

**UNIVERSIDADE FEDERAL DE SANTA CATARINA  
CENTRO TECNOLÓGICO  
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA**

**Framework Java de Apoio ao Desenvolvimento de  
Aplicações Web com Banco de Dados, utilizando Struts,  
Tiles e Hibernate**

David Pedro Willemann

Gustavo Bestetti Ibarra

FLORIANÓPOLIS - 2007

**UNIVERSIDADE FEDERAL DE SANTA CATARINA  
CENTRO TECNOLÓGICO  
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA**

**Framework Java de Apoio ao Desenvolvimento de  
Aplicações Web com Banco de Dados, utilizando Struts,  
Tiles e Hibernate**

Trabalho de Conclusão de Curso apresentado  
como parte dos requisitos para obtenção  
do grau de Bacharel em Ciências da Computação.

David Pedro Willemann  
Gustavo Bestetti Ibarra

Orientador

Vitório Bruno Mazzola

FLORIANÓPOLIS - 2007

# AGRADECIMENTOS

Durante nossa jornada universitária perdemos a conta de quantas vezes fomos iluminados em nossos caminhos. E, ao encontrarmos dificuldades em nossas vidas, lembramos que Deus está nos carregando em seus braços.

Assim, não poderíamos deixar de agradecer primeiramente a Ele por essa conquista em nossas vidas, dentre tantas que ainda lutaremos para conseguir.

Agradecemos em especial, aos nossos amados pais, pelo apoio incondicional e amor eterno, nas dificuldades sendo nosso alicerce, muito obrigado também à família em geral, base de nosso caráter e educação, a todos eles nosso amor e respeito.

Muito obrigado à esposa Sinara e ao filho Ian pela compreensão na ausência, por torcer e dividir os méritos desta conquista. À namorada Morgana pela colaboração e afeto, por entender e ajudar nos maus momentos e ainda estar lá pra curtir os bons.

Ao nosso orientador pelo apoio e contribuição, bem como a todos os professores que de uma forma ou outra, nos ajudaram a ultrapassar esta etapa, obrigado pela riqueza de suas palavras e pela aprendizagem proporcionada.

Agradecemos a banca, por aceitar e colaborar para na finalização desse trabalho bem como a Universidade e seus servidores em geral.

Quando pensamos nos agradecimentos sempre esquecemos de alguns atores participantes dessa nossa história, um grandioso passo em nossas vidas, assim, deixamos registrados o nosso sincero obrigado àqueles que de alguma forma aturam nessa conquista.

# RESUMO

O presente trabalho refere-se ao desenvolvimento de um framework de apoio à criação de sistemas em ambiente Web que utilizam banco de dados para armazenamento de suas informações, com foco nas operações básicas de CRUD (Create Retrieve Update Delete). Nesse sentido o estudo inicia com uma revisão da bibliografia acerca dos conceitos, padrões e metodologias bem como avaliação de ferramentas de desenvolvimento existentes no mercado de software mundial. O referencial teórico utilizado foi o desenvolvimento evolutivo (envolving frameworks). Com o objetivo de desenvolver um framework que seja funcional e ofereça maior produtividade aos desenvolvedores de sistemas de informações sob a plataforma Web, foram integrados os frameworks Struts, Tiles e Hibernate, empregando padrões de projetos. Assim, foi definida uma arquitetura padronizada que oferece aos desenvolvedores pontos de flexão onde as funcionalidades básicas disponibilizadas de forma automática pelo framework podem ser customizadas. Por fim, foi desenvolvida uma aplicação de gerenciamento de controle de empréstimo de equipamentos para o Laboratório de Metrologia (LABMETRO) da Universidade Federal de Santa Catarina, como Prova de Conceito do Framework.

**Palavras chave:** Framework, Java, Struts, Tiles, Hibernate, Desenvolvimento WEB

# LISTA DE FIGURAS

Figura 1 - Modelo MVC.....	39
Figura 2 - Framework de Aplicação.....	44
Figura 3 - Framework de Domínio .....	45
Figura 4 - Metodologia de Desenvolvimento Evolutivo de Frameworks .....	50
Figura 5 - Diagrama de Componentes do Struts.....	52
Figura 6 - Funcionamento Geral do Struts .....	53
Figura 7 - Exemplo de Recortes Tiles: Fonte [30].....	56
Figura 8 - Diagrama de Dependências entre as Camadas BO e DAO.....	61
Figura 9 - Divisão de responsabilidades do framework na camada DAO .....	61
Figura 10 - Utilização do Padrão Template Method no framework .....	62
Figura 11 - Utilização do Padrão Singleton no framework.....	63
Figura 12 - Visão geral da Arquitetura do framework.....	65
Figura 13 - Visão geral do funcionamento do framework .....	66
Figura 14 - Lógicas de Negócio no framework .....	68
Figura 15 - Acesso e Persistência de dados no framework .....	70
Figura 16 - Classe AppActionMapping.....	73
Figura 17 - Classe AppGenericDAO .....	74
Figura 18 - Classe AppFactory .....	75
Figura 19 - Diagrama de Casos de Uso do Administrador.....	78
Figura 20 - Diagrama de Caso de Uso Operador.....	79
Figura 21 - Diagrama de Caso de Uso Usuário .....	79
Figura 22 - Diagrama de Classes Conceituais do SGEL.....	88
Figura 23 - Tela inicial do SGEL.....	91
Figura 24 - Tela Principal do Administrador do SGEL .....	91
Figura 25 - Tela Principal do Operador so SGEL.....	92
Figura 26 - Tela Principal do Usuario Comum do SGEL .....	92
Figura 27 - Lista de Usuários do Sistema do SGEL .....	93
Figura 28 - Cadastro de Usuários do SGEL .....	94
Figura 29 - Consulta de Equipamentos do SGEL.....	95
Figura 30 - Cadastro de Equipamentos do SGEL.....	96
Figura 31 - Lista de Empréstimos do SGEL.....	97
Figura 32 - Novo Empréstimo: Passo 1.....	98
Figura 33 - Novo Empréstimo Passo 2 do SGEL .....	99
Figura 34 - Devolução de Equipamento do SGEL .....	100

# SUMÁRIO

<b>1</b>	<b>INTRODUÇÃO .....</b>	<b>11</b>
1.1	Objetivos .....	12
1.2	Justificativa.....	13
1.3	Organização do Trabalho .....	13
<b>2</b>	<b>REVISÃO DE LITERATURA .....</b>	<b>15</b>
2.1	World Wide Web.....	15
2.2	HTML .....	16
2.3	Linguagem de Programação .....	18
2.3.1	Interpretação e Compilação .....	19
2.4	Orientação a Objetos.....	20
2.4.1	Herança e Polimorfismo .....	21
2.4.2	Sobrecarga .....	22
2.4.3	Sobrescrita .....	22
2.5	Java .....	23
2.5.1	História .....	23
2.5.2	Conceito.....	24
2.5.3	Características .....	25
2.6	Java Servlet API .....	26
2.6.1	Ciclo de Vida de um Servlet.....	27
2.7	Java Server Pages.....	27
2.8	Web Container .....	28
2.9	Internacionalização de Software.....	28
2.10	Banco de Dados .....	29
2.11	UML.....	30
2.12	Características e Padrões de Projeto .....	30
2.12.1	Introdução.....	31
2.12.2	Definição .....	32
2.12.3	Baixo Acoplamento .....	33
2.12.4	Alta Coesão .....	34

2.12.5	Padrão Command.....	36
2.12.6	Padrão Template Method .....	37
2.12.7	Padrão Facade.....	37
2.12.8	Padrão Singleton.....	38
2.12.9	Padrão Abstract Factory.....	38
2.12.10	Modelo MVC.....	38
<b>3</b>	<b>FRAMEWORK .....</b>	<b>41</b>
3.1	Definições.....	41
3.2	Vantagens e Desvantagens .....	42
3.3	Tipos de Frameworks.....	42
3.3.1	Caixa-Branca .....	43
3.3.2	Caixa-Preta .....	43
3.3.3	Caixa Cinza.....	43
3.4	Formas de Utilização .....	43
3.4.1	Framework de Suporte .....	43
3.4.2	Framework de Aplicação.....	44
3.4.3	Framework de Domínio.....	44
3.5	Qualidades de um bom Framework .....	45
3.5.1	Alterabilidade.....	46
3.5.2	Generalidade .....	46
3.5.3	Extensibilidade.....	46
3.5.4	Clareza .....	46
3.5.5	Simplicidade .....	47
3.5.6	Fronteiras .....	47
3.5.7	Ganchos .....	48
3.6	Metodologias .....	48
3.6.1	Dirigidos a Exemplos .....	49
3.6.2	Dirigidos por Flexão.....	49
3.6.3	Evolutivos.....	50
<b>4</b>	<b>FRAMEWORKS UTILIZADOS.....</b>	<b>51</b>
4.1	Struts .....	51
4.1.1	Arquitetura do Struts .....	52

4.1.2	Funcionamento do Modelo Struts .....	53
4.2	Tiles.....	56
4.3	Hibernate .....	57
<b>5</b>	<b>FRAMEWORK DESENVOLVIDO .....</b>	<b>58</b>
5.1	Introdução.....	58
5.2	Ferramentas Utilizadas.....	59
5.2.1	Eclipse .....	59
5.2.2	Tomcat.....	59
5.2.3	MySQL.....	60
5.3	Padrões e Características Empregadas.....	60
5.3.1	Baixo Acoplamento.....	60
5.3.2	Alta Coesão.....	61
5.3.3	Template Method .....	62
5.3.4	Command.....	62
5.3.5	Singleton.....	62
5.3.6	Abstract Factory .....	63
5.3.7	Arquitetura MVC .....	63
5.4	Categorização do Framework.....	64
5.5	Arquitetura do Framework .....	64
5.6	Funcionamento Geral do Framework .....	65
5.7	Camada Model (M).....	67
5.7.1	Camada de Lógica.....	68
5.7.2	Camada de Modelo.....	68
5.7.3	Camada de Acesso a Dados .....	69
5.8	Camada Visão (V) .....	70
5.9	Camada Controle (C) .....	70
5.10	Principais Classes .....	72
5.10.1	AppBaseAction.....	72
5.10.2	AppActionMapping .....	72
5.10.3	AppGenericBO .....	73
5.10.4	AppGenericDAO .....	74
5.10.5	AppGenericVO.....	74
5.10.6	DaoClass Annotation .....	75



5.10.7	BOClass Annotation .....	75
5.10.8	AppFactory.....	75
<b>6</b>	<b>ESTUDE DE CASO - SGEL .....</b>	<b>76</b>
6.1	Introdução.....	76
6.2	Visão Geral do Sistema.....	77
6.3	Atores do Sistema .....	78
6.3.1	Administrador .....	78
6.3.2	Operador.....	79
6.3.3	Usuário .....	79
6.4	Casos de Uso .....	80
6.4.1	Caso de Uso: Cadastro de Usuários .....	80
6.4.2	Caso de Uso: Listar Usuários Cadastrados.....	80
6.4.3	Caso de Uso: Alterar Dados dos Usuários.....	81
6.4.4	Caso de Uso: Excluir Usuários .....	81
6.4.5	Caso de Uso: Cadastrar Equipamentos .....	82
6.4.6	Caso de Uso: Consultar Equipamento por Nome .....	82
6.4.7	Caso de Uso: Alterar Dados dos Equipamentos .....	83
6.4.8	Caso de Uso: Excluir Equipamentos.....	83
6.4.9	Caso de Uso: Emprestar Equipamento.....	84
6.4.10	Caso de Uso: Devolver Equipamento.....	84
6.4.11	Caso de Uso: Notificar Atraso de Devolução .....	85
6.4.12	Caso de Uso: Efetivar Empréstimo de Reserva.....	85
6.4.13	Caso de Uso: Listar Empréstimos.....	85
6.4.14	Caso de Uso: Listar Reservas .....	86
6.4.15	Caso de Uso: Listar Empréstimos em Atraso.....	86
6.4.16	Caso de Uso: Registrar Devolução de Equipamento.....	86
6.4.17	Caso de Uso: Alterar Meus Dados.....	87
6.4.18	Caso de Uso: Listar Meus Empréstimos .....	87
6.5	Classes Conceituais do Sistema.....	88
6.6	Implementação.....	89
6.7	Telas do Sistema .....	91
6.7.1	Tela Inicial .....	91
6.7.2	Tela Principal do Administrador.....	91

6.7.3	Tela Principal do Operador.....	92
6.7.4	Tela Principal do Usuário Comum.....	92
6.7.5	Lista de Usuários do Sistema (Admin).....	93
6.7.6	Cadastro de Usuários.....	94
6.7.7	Consultar Equipamentos.....	95
6.7.8	Cadastro de Equipamentos .....	96
6.7.9	Empréstimos .....	97
6.7.10	Novo Empréstimo (Passo 1).....	98
6.7.11	Novo Empréstimo (Passo 2).....	99
6.7.12	Devolução de Equipamento.....	100
<b>7</b>	<b>CONCLUSÃO .....</b>	<b>101</b>
<b>8</b>	<b>TRABALHOS FUTUROS .....</b>	<b>104</b>
<b>9</b>	<b>REFERÊNCIAS BIBLIOGRÁFICAS.....</b>	<b>105</b>
	<b>ANEXO A – VO’S DO SGEL.....</b>	<b>109</b>
	<b>ANEXO B – CUSTOMIZAÇÕES PARA O SGEL .....</b>	<b>116</b>
	<b>ANEXO C - CONFIGURAÇÕES.....</b>	<b>120</b>
	<b>ANEXO D - CÓDIGOS.....</b>	<b>129</b>
	<b>ANEXO E – ARTIGO.....</b>	<b>141</b>

# 1 Introdução

A história da Internet pode parecer surpreendente para quem a conheceu há pouco tempo. A necessidade da comunicação entre os computadores surgiu remotamente por volta dos anos 60 durante a guerra fria. Era essencial proteger os dados militares de tal forma que, mesmo com um ataque inimigo, os dados fossem preservados. A solução proposta foi uma rede eletrônica de dados, na qual eles deveriam estar distribuídos entre diversos computadores e poderiam ser atualizados no menor espaço de tempo possível. Para atender essa necessidade foi criada a rede ARPANET pelo grupo de pesquisa Advanced Research Projects Agency (ARPA), interligando três computadores no final de 1969. Três anos depois já eram 40 computadores interligados. Mas logo se percebeu que possuir computadores interligados apenas para fins militares não seria interessante.

Assim o meio científico se interessou pela ARPANET como um meio de obter resultados de pesquisas realizadas em outras instituições. Com isso, o número de computadores interligados aumentou muito, ocasionando a separação entre a parte militar e a parte civil. Nos anos 80, a National Science Foundation (NSF) interligou os mais importantes centros científicos à redes menores de universidades, fazendo com que diversas redes fossem unidas. Então, o que era chamado de ARPANET foi batizado de Internet.

Hoje podemos encontrar informações de todos os tipos e áreas publicadas na Internet e numa infinidade de locais, que vão desde uma simples página pessoal até grandes portais de vendas de mercadorias e transações eletrônicas.

Outra coisa que vem acontecendo ao longo dos últimos anos é a utilização da Internet como ambiente para utilização de sistemas. Devido a facilidade que a Internet fornece, no

sentido em que o sistema está disponível em qualquer lugar e a qualquer momento, sem dificuldades de instalação, bastando ter um navegador disponível para acessar a rede.

Algumas tecnologias podem ser usadas para construção de aplicações Web, entre elas Personal Home Page (PHP), Active Server Page (ASP), Microsoft.NET e Java. Dentre elas, a linguagem Java apresenta um potencial muito grande, pois possui diversas formas de utilização. Podendo ser através de Servlet, Java Server Pages (JSP) ou ainda através de templates desenvolvidos em alguma espécie de linguagem de script como, por exemplo, o Velocity, sendo todos executados e processados em um servidor e apenas a página produzida é visualizado no cliente.

Visando o grande crescimento do desenvolvimento de aplicações Web e a necessidade de ter agilidade no desenvolvimento deste tipo de sistema, este trabalho descreve a construção de um arcabouço (framework) em Java que integre algumas tecnologias e frameworks existentes no mercado, concluindo com o desenvolvimento de um sistema de gerenciamento de empréstimo de equipamentos, como forma de avaliar e validar a estrutura e funcionalidades do arcabouço proposto.

## 1.1 Objetivos

O principal objetivo deste trabalho é desenvolver um framework que seja funcional e ofereça maior produtividade aos desenvolvedores de Sistemas de Informações sob a plataforma Web. Não é esperado o desenvolvimento de um framework completo, mas um framework capaz de auxiliar e padronizar o desenvolvimento de aplicação neste domínio.

A elaboração deste trabalho também inclui:

- Estudar o uso de frameworks, suas vantagens, problemas e metodologias de desenvolvimento;
- Estudar alguns frameworks disponíveis para o desenvolvimento Java.

- Desenvolver um sistema utilizando o framework desenvolvido, como forma de prova de conceito.

## 1.2 Justificativa

Desenvolver sistemas para Web é uma tarefa que exige uma série de conhecimentos sobre tecnologias e frameworks que muitas vezes são utilizados de forma errada por diversos desenvolvedores de sistemas.

Os frameworks são estruturas pré-definidas especializados em uma funcionalidade ou em um tipo de uso. Eles devem ser utilizados como base para o desenvolvimento de aplicações e não como ferramentas utilitárias.

Hoje em dia, existe uma infinidade de frameworks Java que auxiliam o desenvolvimento de sistemas. Existem frameworks que controlam fluxo das informações, frameworks para persistência de dados, frameworks para testes unitários até frameworks para controle de interfaces.

Conhecer todos estes frameworks e conseguir utilizá-los de uma forma que as responsabilidades fiquem bem definidas não é uma tarefa simples. Por esses motivos, este trabalho apresenta uma proposta de integração de diversos frameworks e tecnologias Java e Internet como forma de estabelecer um arcabouço de desenvolvimento de sistemas para Web utilizando Java.

## 1.3 Organização do Trabalho

Este trabalho foi organizado em cinco partes. Os três primeiros capítulos são dedicados à parte teórica, o quarto capítulo descreve o conceito principal deste trabalho, ou seja, frameworks. No capítulo cinco descreve-se o framework desenvolvido, e finalizando, o

capítulo seis mostra o desenvolvimento de uma aplicação utilizando o framework desenvolvido, seguido da conclusão sobre o trabalho e trabalhos futuros.

## 2 Revisão de Literatura

Nesta seção são apresentados os conceitos necessários para a fundamentação e desenvolvimento deste trabalho. Para tanto, são abordados definições de Framework, conceitos da Web, HTML, linguagens de programação (JAVA/JSP), considerando sua importância e vantagens, MySQL e também o modelo UML.

### 2.1 World Wide Web

A Web foi criada em um projeto na CERN, mais ou menos no início de 1989, onde Tim Berners-Lee construiu o sistema protótipo que se tornou um modelo do que hoje é a World Wide Web. O intento original do sistema foi tornar mais fácil o compartilhamento de documentos de pesquisas entre os colegas [35].

A World Wide Web (Web ou WWW), traduzido literalmente como “teia do tamanho do mundo”, é uma rede de computadores na Internet que fornece informação em forma de hipertexto. Para ver a informação, pode-se usar um software (navegador) para descarregar informações (documentos ou páginas) de servidores de Internet (sites) e mostrá-los na tela do usuário.

A funcionalidade da Web é baseada em três padrões: a URL, que especifica como cada página de informação recebe um "endereço" único onde pode ser encontrada; HTTP, que especifica como o navegador e servidor enviam informação um ao outro (protocolo); e HTML, um método de codificar a informação de modo que possa ser exibida em uma grande quantidade de dispositivos. Berners-Lee hoje encabeça o World Wide Web Consortium (W3C), que desenvolve e mantém estes padrões e outros de modo a permitir que os computadores na Web armazenem e comuniquem todos os tipos de informação efetivamente.

O termo “Web Services” ou serviços Web apareceu denominando uma nova estrutura para arquitetura de sistemas voltados para a Internet, com especificação inicial pertencente a Microsoft, vem crescendo e sendo usada como sinônimo para denominar serviços disponíveis na Internet capazes de interagir entre si, com troca de informações de forma padronizada, sem requerer intervenção humana. Trata-se de um mecanismo para expor diferentes funcionalidades de um sistema qualquer na Web, permitindo que pessoas e aplicativos possam se comunicar [35].

Os Web Services vêm atraindo o interesse de desenvolvedores que trabalham em todas as plataformas, pois com esta tecnologia é possível que novas aplicações possam interagir com aquelas que já existem e que sistemas desenvolvidos em plataformas diferentes sejam compatíveis.

As páginas Web não passam de documentos de texto simples; apesar de toda sofisticação, elas podem ser produzidas com qualquer editor de texto. A diferença é que as páginas Web contêm algumas marcas especiais para determinar o papel de cada elemento dentro do texto. Alguns elementos são marcados como títulos, outros como parágrafos simples. As marcações são usadas também para indicar os links que levam a outros documentos na rede. Essas marcas são chamadas de tags e estão especificadas dentro da linguagem HTML , utilizada para criar as páginas Web.

## 2.2 HTML

A sigla HTML deriva da expressão inglesa HyperText Markup Language e trata-se de uma linguagem de marcação utilizada para produzir páginas na Internet. Esses códigos podem ser interpretados pelos browsers para exibir as páginas da World Wide Web [13].

A Linguagem de Marcação HiperTexto é uma linguagem simples composta de marcações de formatação e diagramação de hipertexto/hipermídia (informações em texto,



imagens, sons e ações ligadas umas às outras de uma forma complexa e não-sequencial através de chaves relacionadas).

A linguagem do HTML é a linguagem da WWW (Word Wide Web), justamente por essa capacidade de formatação e diagramação de hipertexto/hipermídia. Atualmente existem muitas outras linguagens utilizadas concorrentemente com a HTML (Java, ActiveX, etc...) mas a base da WWW ainda é, de longe, o HTML, que é interpretada por todos os navegadores (browsers) disponíveis (Firefox, Internet Explorer, Mosaic, etc...).

HTML ou linguagem de Marcação é uma linguagem universal e se destina à elaboração de páginas de hiper-texto, como o próprio nome indica. Ela é uma linguagem simples composta de marcações de formatação e diagramação de hipertexto/hipermídia (informações em texto, imagens, sons e ações ligadas umas às outras de uma forma complexa e não-sequencial através de chaves relacionadas). Conceitua-se hiper-texto por certos itens de um documento que contém uma ligação à outra zona do mesmo documento ou, como é mais vulgar, a outros documentos [5].

A principal aplicação do HTML é a criação de páginas na Web que não se trata de uma linguagem de programação. Antes uma espécie de linguagem de formatação, o HTML é um arquivo de texto que é formatado através de uma série de comandos, as tags.

As tags consistem em breves instruções tendo uma marca de início e outro de final, mediante as quais se determinam a formatação do texto, imagens e demais elementos que compõem uma página HTML. As tags tem como início o símbolo < e é fechada com o símbolo />, como exemplo: <marquee> Este é um comando para rolagem de texto em uma página HTML</marquee>.

## 2.3 Linguagem de Programação

Uma linguagem de programação é um método padronizado para expressar instruções para um computador. É um conjunto de regras sintáticas e semânticas usadas para definir um programa de computador. Uma linguagem permite que um programador especifique precisamente sobre quais dados um computador vai atuar, como estes dados serão armazenados ou transmitidos e quais ações devem ser tomadas sob várias circunstâncias.

O conjunto de palavras (tokens), compostos de acordo com essas regras, constitui o código fonte de um software, que é depois traduzido para código de máquina, executado pelo processador.

Uma das principais metas das linguagens de programação é permitir que programadores tenham uma maior produtividade, podendo expressar suas intenções mais facilmente do que quando comparado com a linguagem que um computador entende nativamente (código de máquina).

As linguagens de programação são projetadas para adotar uma sintaxe de nível mais alto, que pode ser mais facilmente entendida por programadores humanos, são ferramentas importantes para que programadores e engenheiros de software possam escrever programas mais organizados e com maior rapidez.

Elas também tornam os programas menos dependentes de computadores ou ambientes computacionais específicos (propriedade chamada de portabilidade). Isto acontece porque programas escritos em linguagens de programação são traduzidos para o código de máquina do computador no qual será executado em vez de ser diretamente executado.

### 2.3.1 Interpretação e Compilação

Uma linguagem de programação pode ser convertida em código de máquina por compilação ou interpretação.

Se o método utilizado traduz todo o texto do programa (código), para só depois executar o programa, então diz-se que o programa foi compilado e que o mecanismo utilizado para a tradução é um compilador (que nada mais é do que um programa). A versão compilada do programa tipicamente é armazenada, de forma que o programa pode ser executado um número indefinido de vezes sem que seja necessária nova compilação, o que compensa o tempo gasto na compilação. Isso acontece com linguagens como Pascal e C (linguagem de programação).

Se o texto do programa é traduzido à medida que vai sendo executado, como em Javascript, Python ou Perl, num processo de tradução de trechos seguidos de sua execução imediata, então diz-se que o programa foi interpretado e que o mecanismo utilizado para a tradução é um interpretador. Programas interpretados são geralmente mais lentos do que os compilados, mas são também geralmente mais flexíveis, já que podem interagir com o ambiente mais facilmente.

Embora haja essa distinção entre linguagens interpretadas e compiladas, as coisas nem sempre são tão simples. Há linguagens compiladas para um código de máquina de uma máquina virtual (sendo esta máquina virtual apenas mais um software, que emula a máquina virtual sendo executado em uma máquina real), como o Java e o Parrot. E também há outras formas de interpretar em que os códigos-fontes, ao invés de serem interpretados linha-a-linha, têm blocos "compilados" para a memória, de acordo com as necessidades, o que aumenta a performance dos programas quando os mesmos módulos são chamados várias vezes, técnica esta conhecida como Just in Time.

## 2.4 Orientação a Objetos

Atualmente existem diversos tipos de linguagem de programação, cada qual com suas particularidades. Diferentes linguagens de programação podem ser agrupadas segundo o paradigma, linguagem procedural, linguagem orientada a objetos, linguagem natural, linguagem de programação em lógica.

A Orientação a Objeto é um paradigma de análise, projeto e programação de sistemas de software baseado na composição e interação entre diversas unidades de software chamadas objetos [18].

A análise e projeto orientados a objetos têm como meta identificar o melhor conjunto de objetos para descrever um sistema de software. O funcionamento deste sistema se dá através do relacionamento e troca de mensagens entre estes objetos.

Hoje existem duas vertentes no projeto de sistemas orientados a objetos. O projeto formal, normalmente utilizando técnicas como a notação UML (Unified Modeling Language) e processos de desenvolvimento como o RUP (Rational Unified Process ou Processo Unificado da Rational); e a programação extrema, que utiliza pouca documentação, programação em pares e testes unitários.

Na programação orientada a objetos, implementa-se um conjunto de classes que definem os objetos presentes no sistema de software. Cada classe determina o comportamento (definidos nos métodos) e estados possíveis (atributos) de seus objetos, assim como o relacionamento com outros objetos [11].

Smalltalk, Perl, Python, Ruby, Php, C++, Java e C# são as linguagens de programação mais importantes com suporte à orientação a objetos. Neste trabalho abordarmos a linguagem Java.

## 2.4.1 Herança e Polimorfismo

A Herança é um princípio da Programação Orientada a Objetos que irá permitir às classes compartilharem atributos e operações baseados em um relacionamento, geralmente generalização [18]. Ela permite a criação de subclasses que herdam atributos e operações da classe pai. Além disso, a herança é um conceito aplicado no momento de criação das classes. Assim sendo usada na intenção de evitar que classes que possuam atributos ou métodos semelhantes sejam repetidamente criados. Para exemplificar observa-se as classes 'aluno' e 'professor', onde ambas possuem atributos como nome, idade e sexo. Nesse caso pode-se criar uma nova classe chamada, por exemplo, 'pessoa', que contenha as semelhanças entre as duas classes, fazendo com que aluno e professor herdem as características de pessoa, desta maneira pode-se dizer que aluno e professor são subclasses de pessoa.

O Polimorfismo é o princípio em que duas ou mais classes derivadas de uma mesma superclasse podem invocar métodos que têm identificações similares (assinatura), mas comportamentos distintos, especializados para cada classe derivada, usando para tanto uma referência a um objeto do tipo da superclasse. A decisão sobre qual o método que deve ser selecionado, de acordo com o tipo da classe derivada, é tomada em tempo de execução, através do mecanismo de ligação tardia.

No caso de polimorfismo, é indispensável que os métodos tenham exatamente a mesma identificação, sendo utilizado o mecanismo de redefinição de métodos. Esse mecanismo de redefinição não pode ser confundido com o mecanismo de sobrecarga de métodos.

Uma observação importante é que, quando o polimorfismo está sendo utilizado, o comportamento que será adotado por um método só será definido durante a execução, mesmo que em geral, esse mecanismo seja um facilitador no desenvolvimento e na compreensão do

código orientado a objetos, há algumas situações onde o resultado da execução pode ser não-intuitivo.

### **2.4.2 Sobrecarga**

A sobrecarga de método é um tipo de polimorfismo, que permite a existência de vários métodos com o mesmo nome, porém com assinaturas sutilmente diferentes, ou seja, variando no número e tipo de argumentos e no valor de retorno, ficando a cargo do compilador escolher de acordo com as listas de argumentos os procedimentos ou métodos a serem executados [18].

A importância de permitir a sobrecarga de métodos não é uma mera conveniência para evitar que se tenha que escolher e usar nomes novos para cada definição, ou para evitar conflitos entre nomes existentes em um determinado escopo, evitando a "poluição" do espaço de nomes usados em programas.

Outra propriedade fundamental da sobrecarga de métodos, é permitir que expressões e nomes definidos por meio do uso de símbolos sobrecarregados possam ser usados em contextos que requerem valores de tipos distintos, permitindo assim que sejam definidas novas abstrações polimórficas.

### **2.4.3 Sobrescrita**

A sobrescrita na programação orientada a objetos é o conceito de redefinição de métodos em classes derivadas.

A redefinição ocorre quando um método cuja assinatura já tenha sido especificada recebe uma nova definição (ou seja, um novo corpo) em uma classe derivada.

## 2.5 Java

### 2.5.1 História

Em 1991, na Sun Microsystems, foi iniciado o Green Project, onde foi iniciado o Java, uma linguagem de programação orientada a objetos. Os mentores do projeto eram Patrick Naughton, Mike Sheridan, e James Gosling. Seu objetivo não era a criação de uma nova linguagem de programação, mas antecipar e planejar a “próxima onda” do mundo digital. Pois acreditavam que em algum tempo haveria uma convergência dos computadores com os equipamentos e eletrodomésticos comumente usados pelas pessoas no seu dia-a-dia.

Provando sua viabilidade, 13 pessoas trabalharam arduamente durante 18 meses, no verão de 1992 eles emergiram de um escritório de Sand Hill Road no Menlo Park com uma demonstração funcional da idéia inicial. O protótipo se chamava \*7 (“StarSeven”), um controle remoto com uma interface gráfica touchscreen. Para o \*7 foi criado um mascote, hoje bem conhecido no mundo Java, o Duke, que tinha o trabalho no \*7 de ser um guia virtual ajudando e ensinando o usuário a utilizar o equipamento. O \*7 tinha a habilidade de controlar diversos dispositivos e aplicações. James Gosling especificou uma nova linguagem de programação para o \*7. Gosling decidiu batizá-la de “Oak”, que significa carvalho [6].

A seguir era preciso encontrar um mercado para o \*7, sua equipe pensava que uma boa idéia seria controlar televisões e vídeo por demanda com o equipamento. Eles construíram um demo chamado MovieWood, mas infelizmente era muito cedo para que o vídeo por demanda bem como as empresas de TV a cabo pudessem viabilizar o negócio. A idéia que o \*7 tentava vender, hoje já é realidade em programas interativos e também na televisão digital. Permitir ao telespectador interagir com a emissora e com a programação em uma grande rede de cabos

era algo muito visionário e estava muito longe do que as empresas de TV a cabo tinham capacidade de entender e comprar. A idéia certa, na época errada.

Por sorte a Internet expandiu, e rapidamente uma grande rede interativa estava se estabelecendo, era este tipo de rede que a equipe do \*7 estava tentando vender para as empresas de TV a cabo. Assim, de repente, não era mais necessário construir a infra-estrutura para a rede, em um golpe de sorte, ela simplesmente está lá. Gosling foi incumbido de adaptar o Oak para a Internet e em janeiro 1995 foi lançada uma nova versão do Oak que foi rebatizada para Java. A tecnologia Java tinha sido projetada para se mover através de redes de dispositivos heterogêneos, redes como a Internet. As aplicações, então, poderiam ser executadas dentro dos Browsers nos Applets Java e tudo seria disponibilizado pela Internet instantaneamente. Foi o estático HTML dos Browsers que promoveu a rápida disseminação da dinâmica tecnologia Java. A velocidade dos acontecimentos seguintes foi assustadora, o número de usuários cresceu rapidamente, grandes players, como a IBM anunciaram suporte para a tecnologia Java.

A partir de seu lançamento, em maio de 1995, a plataforma Java foi adotada rapidamente, e em 2003 o Java atingiu a marca de 4 milhões de desenvolvedores em todo mundo. Ele continuou e continua crescendo e hoje é um dos padrões para o mercado oferecendo qualidade, performance e segurança. Java tornou-se popular pelo seu uso na Internet e hoje possui seu ambiente de execução presente em Web browsers, mainframes, SOs, celulares, palmtops e cartões inteligentes, entre outros.

## **2.5.2 Conceito**

Java é uma linguagem orientada a objetos, independente da plataforma e segura. A orientação a objetos é empregada no processo de desenvolvimento de software, onde um programa é concebido como um grupo de objetos que trabalham juntos, comunicando-se e



cooperando para solucionar o problema proposto. A neutralidade de plataforma é a capacidade de um programa ser executado sem modificações em diferentes ambientes computacionais, de software e de hardware. Os programas em Java são compilados para um formato chamado bytecode, que podem ser executados por diferentes máquinas virtuais Java, presentes em diferentes plataformas de software de hardware.

O Java trata do gerenciamento de alocação e desalocação de espaço em memória, não sendo necessário que o programador tenha essa preocupação. Essa linguagem não permite o uso de ponteiros, minimizando a geração e o tempo gasto com a depuração de erros.

A sintaxe da linguagem Java é originária da linguagem C e a versão atual da linguagem, denominada Java Tiger, apresenta algumas alterações sintáticas, como número de parâmetros variáveis. Essa linguagem é empregada nas aplicações que utilizam os conceitos de applets, que são executados nas máquinas cliente. Porém as aplicações são limitadas a utilizar apenas um subconjunto da linguagem, de forma a garantir a segurança das aplicações. Nos servlets e JSP é também empregado Java como linguagem base para as aplicações.

### **2.5.3 Características**

- Orientação a objeto - Baseado no modelo de Smalltalk e Simula67.
- Portabilidade - Independência de plataforma - "write once run anywhere!".
- Recursos de Rede - Possui extensa biblioteca de rotinas que facilitam a cooperação com protocolos TCP/IP, como HTTP e FTP.
- Segurança - Pode executar programas via rede com restrições de execução.
- Além disso, podem-se destacar outras vantagens apresentadas pela linguagem:
- Sintaxe similar a Linguagem C/C++.
- Facilidades de Internacionalização - Suporta nativamente caracteres Unicode.

- Simplicidade na especificação, tanto da linguagem como do "ambiente" de execução (JVM).
- É distribuída com um vasto conjunto de bibliotecas (ou APIs).
- Possui facilidades para criação de programas distribuídos e multi-thread (múltiplas linhas de execução num mesmo programa).
- Desalocação de memória automática por processo de coletor de lixo.
- Carga Dinâmica de Código - Programas em Java são formados por uma coleção de classes armazenadas independentemente e que podem ser carregadas no momento de utilização.

## 2.6 Java Servlet API

A API Java Servlet permite aos desenvolvedores de sistemas criarem dinamicamente conteúdos em um servidor Web para exibição em um navegador usando a plataforma Java. O conteúdo que será gerado é normalmente HTML, mas pode ser de qualquer outro tipo, como um XML ou documentos binários e textos.

A API Servlet está disponível através do pacote `javax.servlet` que define e implementa todas as formas de interação entre um container Web e um servlet. Um container Web é essencialmente um componente do servidor Web que tem a função de interagir com um servlet e é responsável por gerenciar o ciclo de vida dos servlets além de mapear uma URL a um servlet e controlar os direitos de acesso a estes recursos.

Um servlet é um objeto que recebe uma requisição (request), processa o que for necessário e gera uma resposta (response).

A especificação original de Servlets foi criada pela Sun Microsystems (a versão 1.0 foi finalizada em junho 1997). Depois disso, outras especificações foram desenvolvidas na Java Community Process, como a versão 2.3 que foi definida JSR (Java Submission Request) 53

que definia além da Servlet 2.3 a especificação JavaServer Page 1.2. A JSR 154 especifica a Servlet 2.4 e 2.5. Desde maio de 2006 a versão mais atual de servlet é a especificação 2.5.

### **2.6.1 Ciclo de Vida de um Servlet**

O ciclo de vida de um servlet consiste nos seguintes passos [DEITEL]:

1. A classe do servlet é carregada quando o container Web é iniciado.
2. O container chama o método `init()` do servlet. Este método é executado uma única vez e é o responsável por inicializar o servlet.
3. Depois da inicialização o servlet está pronto para receber as requisições dos cliente. Cada requisição é processada em uma thread separada. O container chama o método `service` que por sua vez determina qual o tipo de request (GET, POST, etc) e de acordo com o tipo faz a chamada ao método específico `doGet()`, `doPost()`, `doTrace()` ou outro.
4. Finalmente, o container chama o método `destroy()` que é responsável por tirar servlet de serviço. Da mesma forma que o `init()`, o método `destroy` é chamado uma única vez ao longo do ciclo de vida de um servlet.

## **2.7 Java Server Pages**

JSP é uma linguagem de script que mistura HTML e programação java, e da mesma forma que os servlets, é capaz de gerar conteúdos dinâmicos.

Quando uma página JSP é requisitada, o container Web verifica se existe um servlet gerado para esta JSP. Caso não exista, um compilador JSP é executado, e o mesmo produz um servlet que por sua vez é executado pelo container, produzindo dinamicamente o conteúdo de resposta.

## 2.8 Web Container

Um container WEB é um componente de um servidor WEB responsável por rodar os servlets de uma aplicação. Existem vários containers WEB no Mercado, mas um dos mais conhecidos pela comunidade de desenvolvedores é o Apache Tomcat, que é uma implementação Open Source de referência para containers WEB.

Através de arquivos de configurações o tomcat é capaz de gerenciar as aplicações e também pool de conexões com banco de dados.

## 2.9 Internacionalização de Software

As atividades de internacionalização e de localização de softwares estão ocupando cada vez mais, uma importância na indústria de software, devido à globalização e conseqüente interação dos principais mercados mundiais, além do crescimento dos usuários da Internet.

Para ser competitivo no mercado mundial o software precisa se comunicar com o usuário final na sua língua nativa e precisa estar baseado nas suas convenções locais.

A internacionalização é o processo de tornar um programa independente do idioma, permitindo a ligação dinâmica da língua e das convenções locais ao software. O resultado é um programa capaz de suportar múltiplas línguas e convenções.

Os pontos fundamentais para internacionalização e localização incluem:

- Língua:
  - Codificação do texto como diferentes sistemas de escrita (Alfabetos), diferentes sistemas numerais, scripts da esquerda-para-direita e scripts da direita-para-esquerda;
  - Representação gráfica do texto;
  - Áudio;

- Sub-títulos para vídeo.
- Formato de data e tempo, incluindo diferentes calendários.
- Formatação de números.
- Fuso horário.
- Números pré-definidos governamentalmente.
- Números de telefones, endereço e códigos postais internacionais.
- Pesos e medidas.
- Tamanho de papéis.

É necessário preparar o software antes de iniciar a tradução para minimizar as tarefas necessárias e os recursos envolvidos na localização. O objetivo da internacionalização é dispensar a necessidade de trocar os componentes centrais durante o processo de localização e manter os componentes localizáveis separados, considerando requisitos e necessidades de outros países e culturas.

## 2.10 Banco de Dados

Um banco de dados é uma coleção de dados estruturados, que pode ser desde uma simples lista de compras a uma galeria de imagens ou a grande quantidade de informação da sua rede corporativa. Os computadores lidam muito bem com grandes quantidades de dados, o gerenciamento de banco de dados funciona então como a engrenagem central da computação, seja como utilitários independentes ou como partes de outras aplicações.

O MySQL é um dos mais populares sistema de gerenciamento de banco de dados relacional, que armazena dados em tabelas separadas em vez de colocar todos os dados num só local. Isso proporciona velocidade e flexibilidade. SQL é linguagem padrão mais comum usada para acessar banco de dados e é definida pelo Padrão ANSI/ISO SQL [23]. O padrão SQL está evoluindo desde 1986 e existem diversas versões.

O servidor de banco de dados MySQL é extremamente rápido, confiável, e fácil de usar, também tem um conjunto recursos muito práticos. Ele foi desenvolvido originalmente para lidar com bancos de dados muito grandes de maneira muito mais rápida que as soluções existentes e tem sido usada em ambientes de produção de alta demanda por diversos anos de maneira bem sucedida. Apesar de estar em constante desenvolvimento, o Servidor MySQL oferece hoje um rico e proveitoso conjunto de funções. A conectividade, velocidade, e segurança fazem com que o MySQL seja altamente adaptável para acessar bancos de dados na Internet.

## 2.11 UML

A Unified Modeling Language (UML) é uma linguagem de modelagem não proprietária de terceira geração. A UML não é um método de desenvolvimento, o que significa que ela não diz para você o que fazer primeiro e em seguida ou como desenhar seu sistema, mas ele lhe auxilia a visualizar seu desenho e a comunicação entre objetos [24].

Basicamente, a UML permite que desenvolvedores visualizem os produtos de seu trabalho em diagramas padronizados. Junto com uma notação gráfica, a UML também especifica significado, isto é, semântica. É uma notação independente de processos, embora o RUP (Rational Unified Process) tenha sido especificamente desenvolvido utilizando a UML.

## 2.12 Características e Padrões de Projeto

Uma abordagem sobre características importantes no desenvolvimento de sistemas orientados a objetos é considerável, bem como um breve histórico dos Padrões de Projeto (Design Pattern), assim apresentamos em detalhes os padrões que foram utilizados no framework desenvolvido neste trabalho.

## 2.12.1 Introdução

Uma abordagem recente em termos de reutilização de projeto são os Padrões (Patterns), no contexto do desenvolvimento de softwares orientados a objetos. Sendo que a principal questão é como proceder para registrar, para poder reutilizar em um desenvolvimento de software, a experiência de projeto adquirida em desenvolvimentos anteriores [28].

Através da observação de que diferentes partes de um projeto possuíam uma estrutura de classes semelhante, os Padrões de Projeto foram originados. Demonstrando a existência de padrões de solução para problemas de projeto semelhantes, e que se repetiam à medida que se procurava produzir uma estrutura flexível [28].

Os Padrões de Projeto seguem soluções simples para problemas específicos no projeto de software orientado a objetos, usando soluções que foram desenvolvidas e aperfeiçoadas ao longo do tempo. Ele reflete modelagens e recodificações, nunca relatadas, resultando dos esforços dos desenvolvedores por maior reutilização e flexibilidade em seus sistemas de software. Esses padrões capturam estas soluções em uma forma sucinta e facilmente aplicável. Compreendendo-se esses padrões, se alcançam as percepções que tornarão os projetos mais flexíveis, modulares, reutilizáveis e compreensíveis [11].

Arquiteturas orientadas a objetos bem-estruturadas estão cheias de padrões, uma das maneiras de medir a qualidade de um sistema orientado a objetos é avaliar se os desenvolvedores tiveram bastante cuidado com as colaborações comuns entre seus objetos. Levando em consideração tais mecanismos durante o desenvolvimento de um sistema, conduz-se a uma arquitetura menor, mais simplória e muito mais compreensível que aquelas produzidas quando estes padrões não são considerados.

Técnicas já testadas e aprovadas são descritas pelos padrões de projetos, o que tornam mais fácil a reutilização de projetos e arquiteturas bem-sucedidas e ajudam a escolher

alternativas de projeto, evitando outras que poderiam comprometer a reutilização. Além disso, podem melhorar a documentação e a manutenção do sistema, assim como fornecer uma especificação explícita de interações de classes e objetos e o seu objetivo subjacente. Sendo assim, ajudam um projetista a obter um projeto “certo” mais rápido.

## 2.12.2 Definição

Em outras áreas já era reconhecida a importância de padrões na criação de sistemas complexos. Christopher Alexander e seus colegas talvez foram os primeiros a propor a idéia de utilizar uma linguagem de padrões em projetos de edificações e cidades, assim essas idéias, junto com as contribuições de outros, assentaram raízes na comunidade de software orientado a objetos.

A idéia de Christopher Alexander era de que cada padrão descreverá um problema no ambiente e o núcleo da sua solução, assim pode-se usar esta solução várias vezes, sem nunca fazê-lo da mesma maneira. Nossas soluções são expressas em termos de objetos e interfaces ao invés de paredes e portas, mas em ambos os tipos de padrões está a solução para um problema num contexto.

Consideram um padrão de projeto tendo quatro elementos essenciais [11]:

1. **O Nome do padrão:** é uma referência que se pode usar para descrever um problema de projeto, suas soluções e conseqüências em uma ou duas palavras. Dar nome a um padrão é importante, porque aumentam o vocabulário do projeto, permite a conversação com colegas, em nossa documentação e até com nós mesmos, tornando mais fácil pensar sobre projetos e comunicá-los a outras pessoas.
2. **O Problema:** narra quando aplicar o padrão. Explicando o problema e o seu contexto, assim podem descrever problemas de projeto específicos, estruturas



de classes ou objeto sintomáticas de um projeto inflexível. Para facilitar o entendimento de sua aplicabilidade, o problema pode conter uma lista de condições que devem ser satisfeitas para que se faça sentido aplicar o padrão.

3. **Solução:** descreve os elementos que compõem o projeto, seus relacionamentos, suas responsabilidades e colaborações. Ela não descreve um projeto concreto ou uma implementação em particular, mas sim uma descrição abstrata de um problema de projeto e de como um arranjo geral de elementos (classes e objetos no nosso caso) resolve o problema.
4. **Conseqüências:** são os resultados e análises das vantagens e desvantagens da aplicação do padrão. Mesmo que não sejam mencionadas na descrição das decisões de projeto, elas são críticas para a avaliação das alternativas de projeto e para a compreensão dos custos e benefícios da aplicação do padrão. As conseqüências de um padrão incluem o seu impacto sobre a flexibilidade, portabilidade ou extensibilidade de um sistema.

### 2.12.3 Baixo Acoplamento

O acoplamento é uma medida de quão fortemente uma classe está conectada a outras classes, tendo conhecimento das mesmas ou depende delas. Uma classe com acoplamento fraco (ou baixo) não é dependente de muitas outras classes (muitas outras é uma expressão que depende do contexto tratado). Uma classe com acoplamento forte (ou alto) depende de muitas outras classes [34].

- Elas são indesejáveis, pois sofrem de problemas como os a seguir:
- Mudanças em classes relacionadas forcem mudanças locais.
- Maior dificuldade de compreensão isoladamente.

- Mais difíceis para reutilizá-las, pois o seu uso requer a presença adicional das classes das quais ela depende.

O Acoplamento Fraco é um princípio que deve ser levado em conta em todas as decisões de projeto, é um objetivo subjacente que se deve ter sempre em mente, sendo que é um padrão de avaliação, ou seja, um padrão que um projetista utiliza enquanto avalia todas as decisões de projeto.

Ele estimula a atribuição de uma responsabilidade de maneira que a sua localização não aumente o acoplamento até um nível que conduza aos resultados negativos que o acoplamento forte pode produzir. O Acoplamento Fraco também apóia o projeto de classes que são mais independentes e isso reduz o impacto de mudanças, promovendo maior capacidade de reutilização, o que aumenta as oportunidades de se obter uma produtividade mais alta. Além disso, ele não pode ser considerado isoladamente de outros padrões, e necessita ser incluído como um dentre vários princípios de projeto que influenciam uma escolha de atribuição de uma responsabilidade.

Não se deve buscar em excesso o baixo acoplamento, pois algum grau moderado de acoplamento entre as classes são normais e necessários, para criar um sistema orientado a objetos, no qual as tarefas são executadas por uma colaboração entre objetos conectados.

#### **2.12.4 Alta Coesão**

Coesão é uma medida de quão fortemente relacionadas e focalizadas são as responsabilidades de uma classe, em termos de projeto orientado a objetos. Uma classe com responsabilidades altamente relacionadas e que não executa um formidável volume de trabalho tem coesão alta. A importância da aplicação deste conceito está em atribuir uma responsabilidade a uma classe para que a coesão permaneça alta [34].

Uma classe com coesão baixa realiza muitas ações não-relacionadas, ou executa demasiado trabalho. Tais classes não são desejáveis, pois sofrem dos seguintes problemas:

- Difíceis de compreender.
- Difíceis de reutilizar.
- Difíceis de manter.
- Delicadas, pois são constantemente afetadas pelas mudanças.

Classes com coesão baixa representam, em geral, uma abstração de “grande granularidade”, ou então, assumiram responsabilidades que deveriam ter sido delegadas a outros objetos.

Na prática, o nível de coesão sozinho não pode ser considerado isoladamente de outras responsabilidades e de outros princípios. Bem como o Acoplamento Fraco, a Coesão Alta é um princípio que se deve ter em mente durante todas as decisões de projeto; é um objetivo subjacente a ser levado em conta continuamente; é um padrão de avaliação que um projetista utiliza enquanto julga todas as decisões de projeto.

Grady Booch [3] descreve a coesão funcional alta sendo algo existente quando os elementos de um componente (bem como uma classe) trabalham todos em conjunto, para fornecer algum comportamento bem-delimitado.

Alguns cenários que ilustram graus variáveis de coesão funcional são dados a seguir:

- **Coesão Muito Baixa:** uma classe é a única responsável por diversas ações em áreas funcionais muito diferentes.
- **Coesão Baixa:** uma classe é a única com responsabilidades de uma tarefa complexa em uma área funcional.
- **Coesão Alta:** uma classe tem responsabilidades moderadas em uma área funcional e ainda ajudam outras classes para levar as tarefas a termo.

- **Coesão Moderada:** uma classe tem peso leve e responsabilidades exclusivas em umas poucas áreas diferentes que estão claramente relacionadas ao conceito da classe, mas não entre si.

Como regra prática, uma classe com coesão alta tem um número relativamente pequeno de métodos, com funcionalidades altamente relacionadas e não executa muito trabalho. Quando a tarefa for grande, ela colabora com outros objetos para compartilhar o esforço.

### 2.12.5 Padrão Command

O padrão Comando é aquela que especifica a definição de uma classe para cada mensagem ou comando, cada uma com um método com nome e parâmetros pré-determinados (normalmente o método é chamado execute) [18].

O princípio para utilização deste padrão está na criação de uma interface que define a operação a ser executada, adicionalmente pode-se ter uma classe abstrata (Comando) que implemente esta interface. Cada subclasse de Comando tem um único método execute que especifica as ações para aquele comando.

As vantagens desse padrão são:

- Suportar “undo” (desfazer). A operação Execute do Comando pode armazenar o estado para reverter seus efeitos no próprio comando. Sendo que a classe Comando deve ter uma operação adicional “unexecute” que reverte os efeitos de uma chamada prévia para Execute.
- Estruturar um sistema em volta de operações de alto nível construídas sobre operações primitivas.

- O padrão Comando vai separar o objeto que invoca a operação daquele que sabe executá-la.
- Permite que seja criado um “log” (registro) dos comandos que foram executados.

### **2.12.6 Padrão Template Method**

O Padrão Template Method mostra como definir o esqueleto de um algoritmo em uma operação, postergando alguns passos para as subclasses, assim, ele permite que as mesmas redefinam certos passos do algoritmo sem mudar a estrutura do mesmo.

A utilização deste padrão contribui para a definição de uma arquitetura e fluxo de operações que o framework deve executar para determinado contexto, além de oferecer ao desenvolvedor, pontos de extensão, onde poderão ser acrescentadas informações específicas de um domínio de aplicação para que o framework as trate [18].

### **2.12.7 Padrão Facade**

Este padrão define uma “fachada” para um conjunto de classes e interfaces de um subsistema, ele é utilizado para ocultar a complexidade da arquitetura de um subsistema, oferecendo, para quem precisa utilizar as funcionalidades de um determinado módulo, operações prontas, que são responsáveis por controlar a lógica e fazer as chamadas necessárias as classes do subsistema requisitado [18].

A principal idéia deste padrão é facilitar a utilização dos recursos de um subsistema, expondo somente as funcionalidades do domínio do subsistema. A estruturação de um sistema em subsistemas ajuda a reduzir a complexidade. Um objetivo comum de todos os projetos é minimizar a comunicação e as dependências entre subsistemas. Para atingir este objetivo

pode-se introduzir um objeto fachada, o qual fornece uma interface única e simplificada para os recursos e facilidades mais gerais de um subsistema.

### **2.12.8 Padrão Singleton**

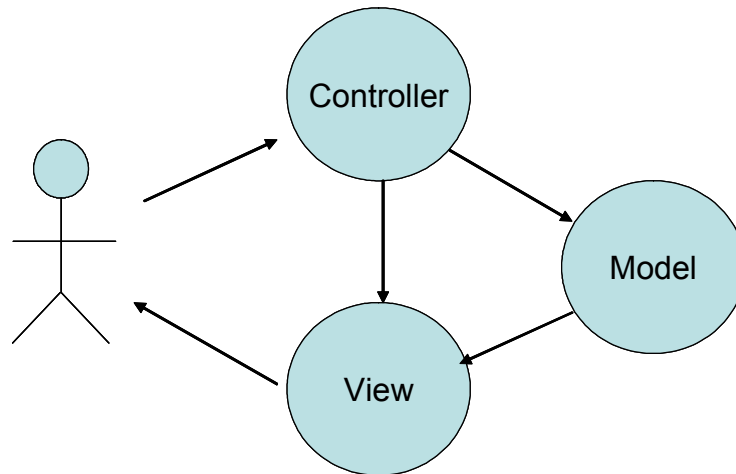
O objetivo desse padrão é assegurar que uma classe tenha uma única instância e prover um ponto de acesso global a esta instância. Ele torna fácil fazer com que seja criado um número fixo, ou um número máximo de instâncias em vez de apenas uma única instância, apenas basta mudar a implementação interna do Singleton [18].

### **2.12.9 Padrão Abstract Factory**

O Padrão Abstract Factory utilizado para a criação dinâmica de objetos de uma determinada família, relacionados ou dependentes, sem especificar suas classes concretas. Sendo muito útil quando se trabalha com interfaces, e deseja-se criar objetos de forma dinâmica das classes que implementam esta interface [18].

### **2.12.10 Modelo MVC**

Segundo Lozano [20], a sigla MVC denota os três papéis no qual todo componente da aplicação deve ser classificado: Model, View e Controller, ou em português: Modelo, Visualização e Controlador. Consiste em um padrão de arquitetura de aplicações que visa separar a lógica da aplicação (Model), da interface do usuário (View) e do fluxo da aplicação (Controller). Permite que a mesma lógica de negócios possa ser acessada e visualizada por várias interfaces.



**Figura 1 - Modelo MVC**

MVC também é utilizado em padrões de projetos de software, porém, MVC abrange mais da arquitetura de uma aplicação do que é típico para um padrão de projeto.

Componentes de modelo são responsáveis pelo armazenamento dos dados durante a sessão do usuário para a execução de regras de negócios, sendo que os componentes de visualização se encarregam da exibição final e de fornecer meios do usuário indicar as operações a serem realizadas pela aplicação. Componentes do controlador são responsáveis em fazer a ponte entre os outros dois tipos de componentes, interpretando os eventos gerados pelos componentes de visualização e disparando a execução do método correspondente ao Modelo.

Usando-se a Web, o emprego de MVC, faz com que se desenvolva uma aplicação bem estruturada e com módulos bem distintos, em que os Web-designers, que são os profissionais responsáveis pela parte visual da aplicação, não precisam entender rotinas de programação que são desenvolvidas por programadores propriamente ditos. O MVC possibilita o armazenamento de objetos na sessão do usuário o que auxilia as regras de negócios e torna a aplicação mais segura, porque não se utiliza cookies nas máquinas do cliente e evita-se o emprego de passagem de parâmetros através de links. Esta última técnica é desaconselhável,

pois qualquer pessoa pode alterar o parâmetro de um link, e com isso ter acesso a áreas denominadas restritas.

Para uma implementação correta, as camadas Model , Controller e View devem ser implementadas de forma que a inversão da ordem não acarrete problemas por dependência, quer dizer, a camada de interface (View) depende de controle (Controller) que implementa um Modelo (Model), mas nunca o inverso.



# 3 Framework

## 3.1 Definições

Várias definições sobre framework são descritas na literatura, mas segundo Gamma "um framework é um conjunto de classes que cooperam entre si provendo assim um projeto reutilizável para um domínio específico de classes de sistema" [GAM].

Um framework ou arcabouço é uma estrutura de suporte definida em que um outro projeto de software pode ser organizado e desenvolvido, quando se analisa o conceito no âmbito do desenvolvimento de software. Um framework pode incluir programas de suporte, bibliotecas de código, linguagens de script e outros softwares para ajudar a desenvolver e juntar diferentes componentes de um projeto de software.

Os Frameworks são projetados com o propósito de facilitar o desenvolvimento de software, habilitando projetistas e programadores a gastarem mais tempo detalhando as exigências de negócio do software do que com detalhes tediosos de baixo nível do sistema.

Características básicas devem ser respeitadas para que projetos de software sejam considerados um framework:

- Precisa ser reutilizável.
- Precisa facilitar o desenvolvimento de sistemas.
- Precisa possuir boa documentação.
- Precisa ser completo para o que se propõe.
- Precisa ser eficiente.

## 3.2 Vantagens e Desvantagens

Utilizando frameworks a principal vantagem é a redução de custos, sendo que já existe uma estrutura definida e que o desenvolvimento pode concentrar-se em implementar as regras específicas do negócio em que o sistema deve atuar. Um framework ainda proporciona uma maior reutilização de códigos e a fatoração de problemas em aspectos comuns a várias aplicações, permite também obter sistemas com códigos menos frágeis e com menos defeitos.

Entretanto, construir um framework é uma tarefa complexa, pois o reuso não acontece por acaso, devendo ser muito bem planejado. Iniciar a construção de um framework sem um bom planejamento pode trazer mais prejuízos do que vantagens.

Com certeza construir uma aplicação tendo que construir um framework demora muito mais do que construir uma aplicação isolada. Isso tudo pelo fato de que quando se constrói um framework devendo planejá-lo de forma que atenda a mais do que uma aplicação, ou seja, atenda a um domínio específico de aplicações e não somente uma. As vantagens de um framework só aparecem a longo prazo, na medida em que a estrutura torna-se consistente e de domínio das equipes de desenvolvimento.

## 3.3 Tipos de Frameworks

Classifica-se um framework de acordo com duas dimensões: como ele é utilizado e onde é utilizado.

Quando tratamos de como um framework pode ser utilizado, analisamos o ponto de como introduzir as particularidades de uma aplicação. Neste sentido temos os frameworks Caixa-Branca, Caixa-Preta e Caixa-Cinza.

### **3.3.1 Caixa-Branca**

Os frameworks de Caixa Branca são baseados na especialização por herança e sobrescrita de métodos, modificando assim as funcionalidades básicas do framework.

### **3.3.2 Caixa-Preta**

São os frameworks focados na composição devendo utilizar as funcionalidades já presentes no framework, ou seja, neste tipo de framework as funcionalidades internas não podem ser vistas nem modificadas e devem-se utilizar as interfaces fornecidas pelo framework. Neste framework as instanciações e composições feitas é o que determinam as particularidades da aplicação.

### **3.3.3 Caixa Cinza**

Os frameworks Caixa Cinza são híbridos, misturam os dois focos: herança e composição, ou seja, são frameworks baseados em herança (caixa branca) com algumas funcionalidades prontas.

## **3.4 Formas de Utilização**

Formas de utilização de um framework são apresentadas à seguir.

### **3.4.1 Framework de Suporte**

São frameworks que fornecem serviços em nível de sistema operacional, como acesso a arquivos, computação distribuída e acesso a dispositivos. Este tipo de framework é mais raros e muito complexo.

### 3.4.2 Framework de Aplicação

São conhecidos também como frameworks horizontais, neste tipo de framework todo o conhecimento encapsulado nele é aplicado a uma enorme gama de aplicações. Ele não provê a solução completa para uma aplicação, mas sim uma estrutura completa para a solução da aplicação. Um exemplo deste tipo são os frameworks para construção de interfaces de sistema (GUI).

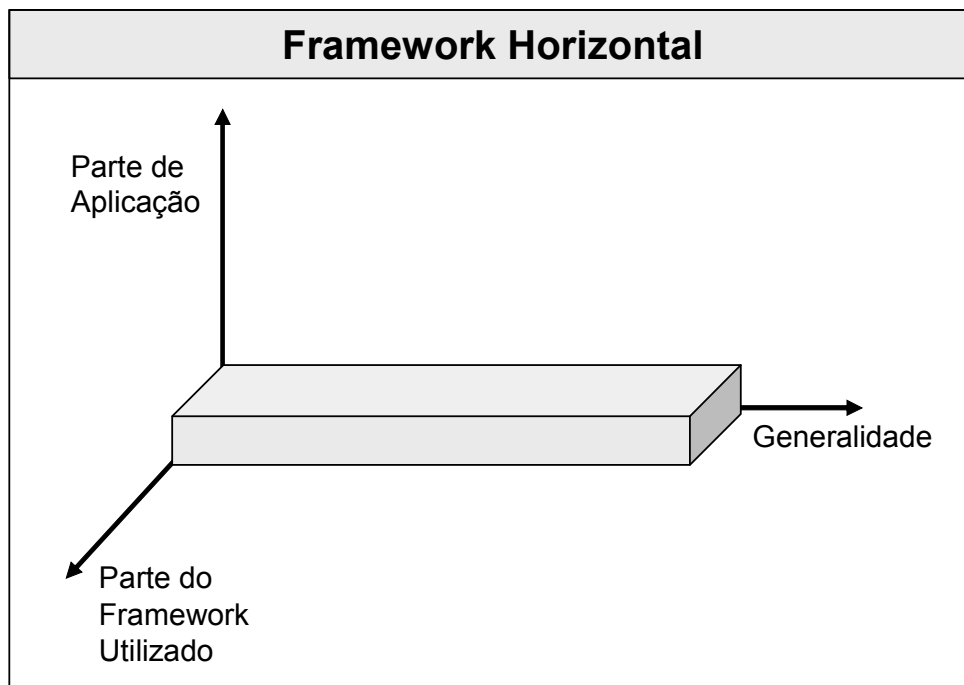
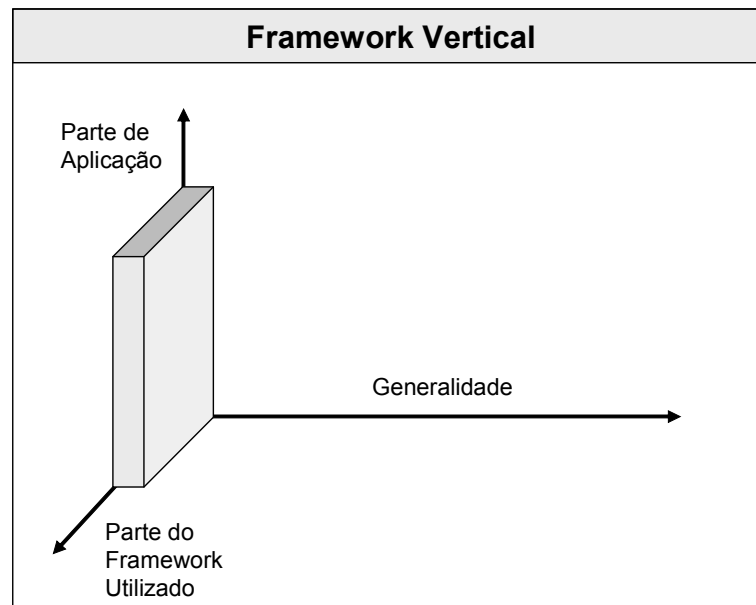


Figura 2 - Framework de Aplicação

### 3.4.3 Framework de Domínio

Conhecidos também como frameworks verticias, neste tipo de framework o conhecimento encapsulado nele provê soluções para um domínio específico de aplicações, embutindo regras comuns ao domínio. Assim, boa parte de uma aplicação desenvolvida com um framework de domínio já possui funcionalidades, um exemplo deste tipo de podemos citar

um framework muito conhecido, o Hibernate, que é utilizado para manipular bancos de dados com java.



**Figura 3 - Framework de Domínio**

### 3.5 Qualidades de um bom Framework

Para que um framework ofereça um bom suporte ao domínio de aplicações a que se propõe, deve buscar implementar algumas características que irão contribuir muito para o aumento da qualidade do framework, entre elas, pode-se citar como principais: a generalidade, alterabilidade e extensibilidade. Para que ocorra isto o projeto do framework deve ser bem elaborado, buscando identificar que partes devem ser mantidas flexíveis para produzir um projeto bem estruturado [15]. Assim, devem-se utilizar os princípios de um projeto orientado a objetos, como o uso da herança para reutilização de interface ao invés de reutilização de código e o uso do polimorfismo na definição das classes e métodos. Suas classes abstratas precisam estar no topo da hierarquia de classes, pois a finalidade destas classes é definir as interfaces a serem herdadas pelas classes concretas das aplicações.

Podendo-se com isto afirmar que o desenvolvimento de um framework é mais complexo que o desenvolvimento de uma aplicação específica. Para um bom funcionamento o framework deve possuir algumas qualidades.

### **3.5.1 Alterabilidade**

Repercute a capacidade do framework de mudar suas funcionalidades em função da necessidade de uma aplicação específica sem que estas alterações resultem em conseqüências imprevistas no conjunto de sua estrutura.

### **3.5.2 Generalidade**

Reflete a capacidade do framework em dar suporte a muitas aplicações diferentes de um mesmo domínio, sendo flexível o suficiente para que as características de alterabilidade e extensibilidade possam ser aplicadas.

### **3.5.3 Extensibilidade**

Repercute a capacidade do framework de ampliar sua funcionalidade sem conseqüências imprevistas no conjunto de sua estrutura, está ligada diretamente à manutenção do framework, permite que sua estrutura evolua por toda sua vida útil, pois à medida que vai sendo utilizado, outros novos recursos vão sendo agregados para que ele se ajuste as novas aplicações a que dá suporte.

### **3.5.4 Clareza**

Os aspectos comportamentais do framework precisam estar encapsulados, não têm necessidade de o desenvolvedor saber todos os detalhes de como os frameworks fazem

alguma coisa para que se possa utilizá-lo. A interface pública das classes do framework deve ser tão simples quanto possível, visto que são nas interfaces públicas que o desenvolvedor estará trabalhando, é importante mantê-las não mais complexas que o necessário para que se possa alcançar a funcionalidade desejada.

O desenvolvedor não deveria ter que sequencialmente chamar três ou quatro métodos no framework para acompanhar um determinado processo, a menos que o processo seja utilizado muitas vezes na parte.

### **3.5.5 Simplicidade**

A estrutura geral do framework deve ser de fácil compreensão para que assim o desenvolvedor possa aprendê-lo em pouco tempo. Obviamente todos os detalhes do projeto não poderão ser aprendidos em poucos dias, no entanto, o desenvolvedor deve estar apto para entender seu funcionamento em pouco tempo, deixando o aprendizado de seus detalhes para o decorrer de sua utilização.

Esta simplicidade é alcançada pelo projeto de interfaces limpas e consistentes, pela utilização de padrões em seu projeto, código, interfaces e nomenclaturas. Todos os objetos pertencentes ao mesmo domínio devem herdar de uma interface comum, de modo que em se conhecendo os métodos e atributos desta interface, bem como seu design e funcionamento, o aprendizado de suas subclasses é facilitado.

### **3.5.6 Fronteiras**

Um framework tem responsabilidades claras e sucintas, e deve atendê-las, nada mais. Todas as funcionalidades exteriores a fronteira do framework devem ser tratadas pelo desenvolvedor. Quando um framework ultrapassa esta fronteira, ele se torna complexo e provavelmente o desenvolvedor ao tentar utiliza-lo precisará implementar código adicional

junto ao framework para conseguir o comportamento desejado. Um framework não fornece a funcionalidade da aplicação, ele fornece o esqueleto sobre o qual a aplicação é construída, sua funcionalidade é responsabilidade do desenvolvedor que o utiliza. Caso o framework deseje prover classes mais especializadas, estas devem ser fornecidas em bibliotecas de classes separadas como subclasses das classes do framework. Isto permitirá ao desenvolvedor escolher usar ou não estas classes, para então fazer uma clara distinção entre o framework e o kit de ferramentas do desenvolvedor.

### **3.5.7 Ganchos**

Um caminho para fornecer capacidade de expansão das funcionalidades do framework é oferecer ao desenvolvedor métodos pelos quais ele possa escrever código afetando o comportamento do framework. Um exemplo é se o framework possuisse um formulário de cadastro qualquer, tendo um método `Salvar()` para gravar as alterações feitas pelo usuário, o framework poderia dispor de um método `AntesDeSalvar()` e `DepoisDeSalvar()`, vinculados ao método `Salvar()`. Estes métodos poderiam prover ao desenvolvedor, locais para escrever código antes e depois das edições do usuário serem salvas. Se o método `Salvar()` respeitar o valor retornado pelo método `AntesDeSalvar()`, o desenvolvedor poderia decidir, por exemplo, cancelar a operação se alguma condição não for respeitada, sem precisar implementar código adicional na classe do framework para prover esta funcionalidade.

## **3.6 Metodologias**

Existem algumas metodologias para o desenvolvimento de frameworks na literatura, assim, descrevemos metodologias de desenvolvimento de framework: dirigidas por exemplos (exemple-driven design), dirigidas por pontos de flexão (hot spot driven design) e os frameworks evolutivos (envolving frameworks).



### **3.6.1 Dirigidos a Exemplos**

Esta metodologia de desenvolvimento, dirigida por exemplos, busca o maior número possível de aplicações já desenvolvidas. Devendo ser realizada uma análise para identificar quais partes das aplicações são comuns e quais delas são específicas. Sendo que as partes comuns devem ser generalizadas de modo que passam a constituir o framework. Enquanto que as partes específicas das aplicações precisam ser desenvolvidas na forma de métodos de gatilho (hook). Esses métodos de gatilho são padrões utilizados por uma aplicação ou framework para fazer chamadas para partes de códigos não existentes no seu desenvolvimento [25].

Para verificar se o framework foi bem desenvolvido, deve-se implementar novamente as aplicações exemplos, mas desta vez utilizando o framework desenvolvido como base. No entanto, este método é mais indicado quando já se tem as aplicações desenvolvidas (ou esteja disposto a desenvolvê-las). E quanto mais aplicações foram utilizadas como exemplos, mais genérico será o framework.

### **3.6.2 Dirigidos por Flexão**

A metodologia dirigida por flexão é necessária para iniciar ao desenvolvimento da arquitetura e implementação do framework para o domínio específico, e com o intuito de manter o framework flexível deve-se buscar os pontos de flexão (hot spots). Os pontos de flexão são regiões de código específicas de cada aplicação, como métodos de resposta a eventos que tem comportamentos diferentes em cada aplicação. Nestes pontos são aplicados vários padrões para manter o código flexível.

Esta metodologia teria uma melhor utilidade em um contexto em que se deseja desenvolver as aplicações de um determinado domínio, mas antes é desenvolvido um

framework para servir como base. Neste caso a implementação das aplicações é o melhor caso de teste.

### 3.6.3 Evolutivos

Frameworks evolutivos é uma metodologia formada por uma linguagem de padrões que devem ser seguidos durante o desenvolvimento do framework, Tem nove padrões propostos que servem como um guia de alto nível, para o desenvolvimento de um framework de forma evolutiva [26]. Nenhum destes padrões são obrigatórios ou devem ser seguidos de forma sequencial. Abaixo uma figura que contém um gráfico com todos os padrões:

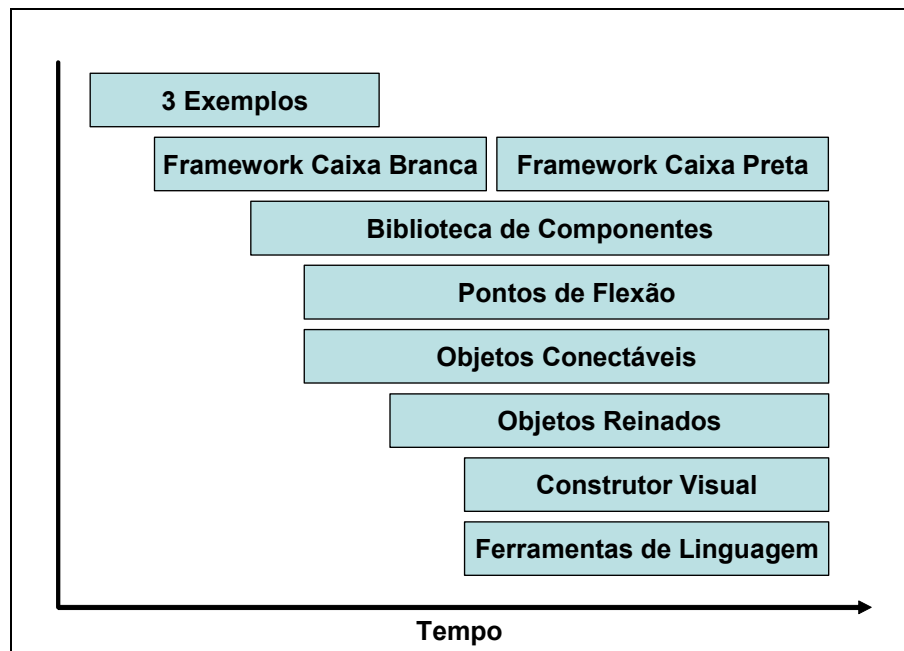


Figura 4 - Metodologia de Desenvolvimento Evolutivo de Frameworks

# 4 Frameworks Utilizados

## 4.1 Struts

O framework Struts foi desenvolvido por Craig McClanahan e doado, em maio de 2002, à Apache Foundation. Atualmente há uma comunidade de desenvolvedores espalhados pelo mundo, trabalhando de forma cooperativa neste projeto [31].

Este é um modelo de código aberto, proposto pela Jakarta, útil para construir aplicativos Web, usando Servlets e JSP. Struts estimula o uso de uma arquitetura de aplicação baseada no paradigma de projeto Model-View-Controller (MVC), conhecida como Modelo 2. Ele fornece seu próprio componente de controle e o integra com outras tecnologias para possibilitar o Modelo e a Visão [14]. Com relação ao Modelo, Struts pode interagir com qualquer tecnologia padrão de acesso a dados. Quanto à Visão, o modelo Struts funciona bem com JSP.

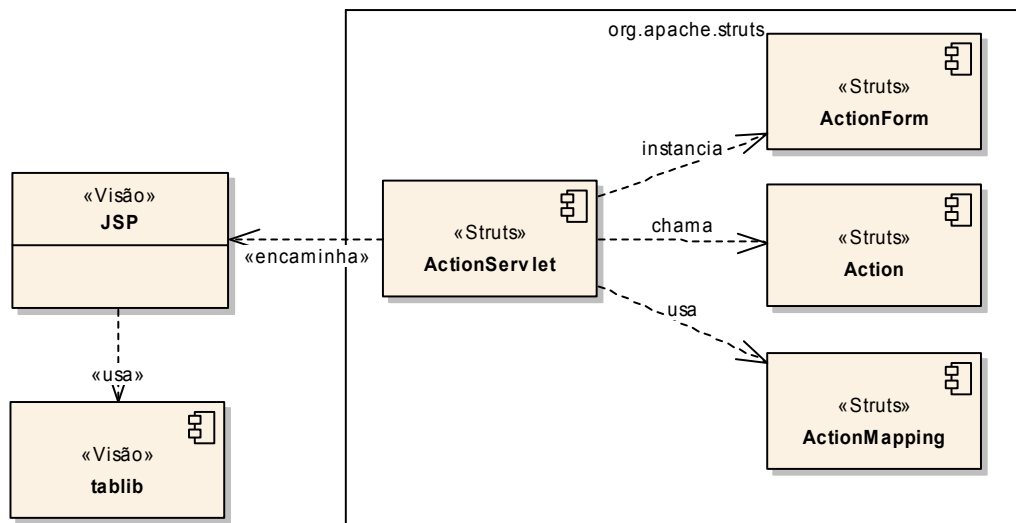
As principais motivações que levaram o Struts a ser utilizado como padrão de mercado são:

- Ele possui garantia de que o Apache Group irá manter o framework (correção de bugs e novos releases);
- Possibilita Integração com a maioria das IDEs de mercado;
- Não “reinventa a roda”, focando os seus esforços em regras de negócio;
- Ele separa a camada de negócio da camada de apresentação;
- Incorpora diversos padrões de projetos (design patterns);
- Cria aplicações padronizadas, facilitando a manutenção;
- Cria aplicações internacionalizadas;

- Possibilita gerar a saída de acordo com o dispositivo usado (HTML, SHTML, WML, etc);
- Aumenta a produtividade.

### 4.1.1 Arquitetura do Struts

Todos os componentes básicos dessa arquitetura são encontrados em um pacote chamado `org.apache.struts.action` como é mostrada na figura abaixo:



**Figura 5 - Diagrama de Componentes do Struts**

Os componentes ActionServlet, ActionForm, Action e ActionMapping são componentes fundamentais da arquitetura Struts, sendo descritos como:

- O ActionServlet (`org.apache.action.ActionServlet`) é o controlador no framework Struts, responsável por criar e interagir com os objetos Action e utilizar o ActionMapping para carregar os mapeamentos.
- O ActionForm (`org.apache.action.ActionForm`) é responsável por tratar os formulários, suas instâncias são preenchidas com os dados do formulário e validados (se necessário) pela invocação do método `validade` antes de um objeto Action ser executado.

- O Action (org.apache.action.Action) chama a lógica de negócio, que em geral, se encontra separada fisicamente em outra camada. Tendo o objetivo de manter a capacidade de reutilização de quaisquer componentes de negócios.

- O ActionMapping (org.apache.action.ActionMapping) contém todos os mapeamentos utilizados pelo controlador para encaminhar as solicitações para os respectivos objetos Action. Seus objetivos são carregados em memória a partir de um arquivo XML chamado struts-config.xml.

### 4.1.2 Funcionamento do Modelo Struts

O funcionamento do Struts pode ser descrito como na figura abaixo, mostrando passo a passo desde a primeira solicitação do usuário, passando pelo gerenciamento e processamento do Struts para essa solicitação, até retornar a resposta ao usuário através de uma nova página html.

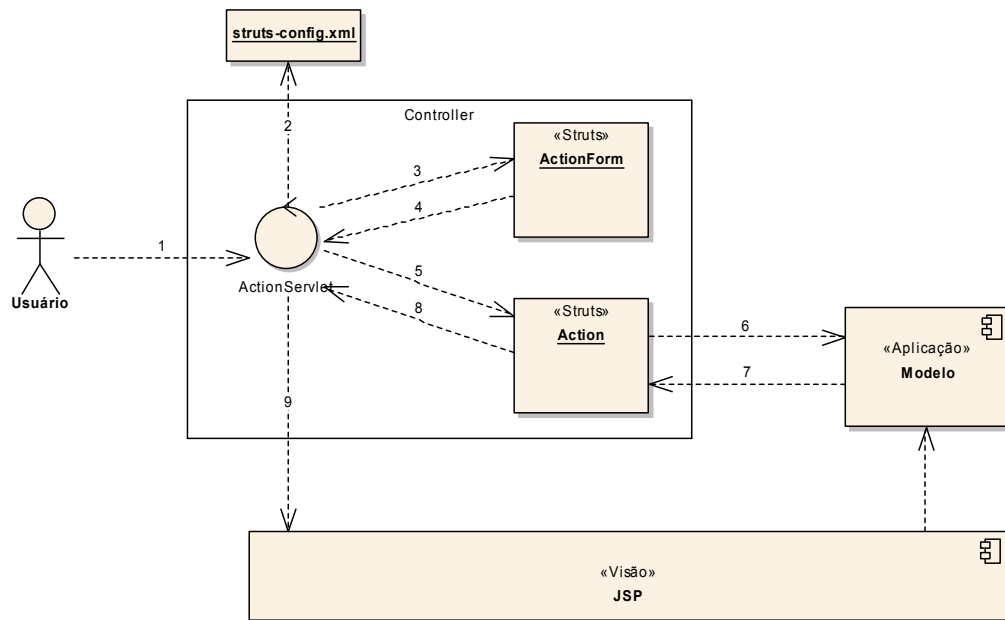


Figura 6 - Funcionamento Geral do Struts

1. O usuário faz uma solicitação através de uma url no browser. Ex: `http://localhost:8080/catalogo.do`. Note que no final da url tem um `.do` que será usado para invocar (na verdade mapear) o servlet controlador do Struts;
2. Se for a primeira solicitação que o container recebeu para esta aplicação, ele irá invocar o método `init()` da `ActionServlet` e irá carregar as configurações do arquivo `struts-config.xml` em estruturas de dados na memória. Vale lembrar que esta passagem só será executada uma única vez, pois nas solicitações subsequentes, o servlet consulta estas estruturas na memória para decidir o fluxo a ser seguido;
3. Baseado no fluxo definido no arquivo `struts-config.xml`, e que neste momento já se encontra carregado em estruturas na memória, o `ActionServlet` identificará qual o `ActionForm` (classe para a validação dos dados) irá invocar. A classe `ActionForm` através do método `validate` irá verificar a integridade dos dados que foram recebidos na solicitação que vem do browser;
4. O controle da aplicação é retomado pelo `ActionServlet`, que checa o resultado da verificação do `ActionForm`. Faltando alguma coisa (campo não preenchido, valor inválido, etc), o usuário recebe um formulário html (geralmente o mesmo que fez a solicitação), informando o motivo do não atendimento da solicitação, para que o usuário possa preencher corretamente os dados para fazer uma nova solicitação. Caso não tenha faltado nenhuma informação, ou seja, todos os dados foram enviados corretamente, o controlador passa para o próximo passo `Action`;
5. O `ActionServlet`, baseado no fluxo da aplicação (estruturas já carregadas em memória) invoca uma classe `Action`. A classe `Action` passará pelo método `execute` que irá delegar a requisição para a camada de negócio;

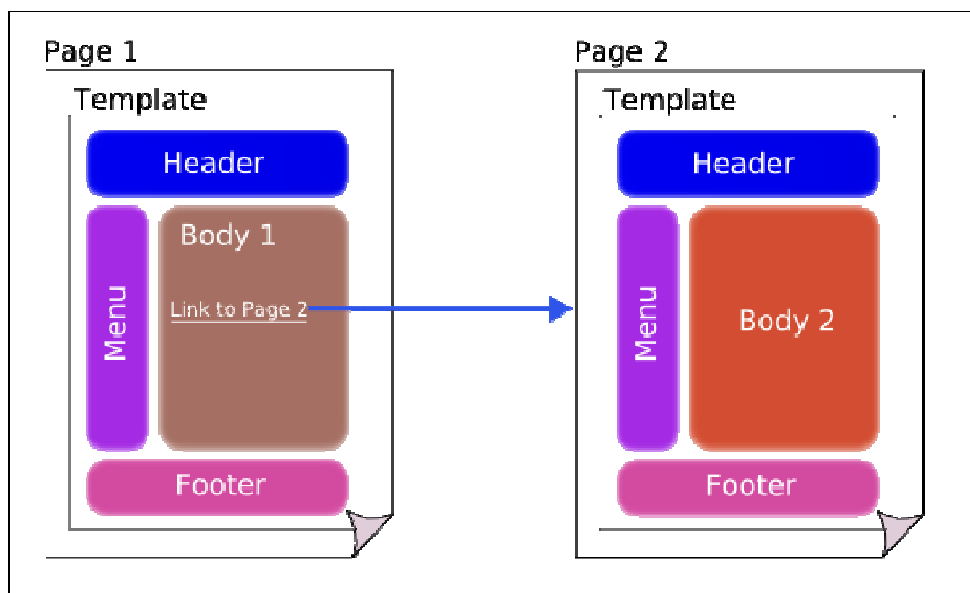
6. A camada de negócio irá executar algum processo (geralmente popular um bean, ou uma coleção). O resultado da execução do processo (objetos já populados) será usado na camada de apresentação para exibir os dados;
7. Quando o controle do fluxo da aplicação voltar ao Action que invocou o processo da camada de negócio, será analisado o resultado e definido qual o mapeamento será utilizado para o fluxo da aplicação. Neste ponto, os objetos que foram “populados” na camada de negócio serão inseridos como atributos na seção do usuário;
8. Se baseando no mapeamento feito pelo o Action, o controlador faz um forward para a JSP para apresentar os dados.
9. Na camada de apresentação (Visão), os objetos que foram inseridos como atributos da sessão do usuário serão consultados para montar a JSP.
10. Chega a JSP como resposta da requisição feita pelo usuário.

O controlador já vem implementado no Struts, no entanto é possível estendê-lo a fim de adicionar funcionalidade. O fluxo da aplicação é programado em um arquivo XML através das ações que serão executadas e as ações são classes base implementadas pela framework seguindo o padrão MVC. Então devemos estendê-las a fim de adicionar a funcionalidade desejada.

A geração da página dinâmica é feita através de tags, sendo que a distribuição Struts inclui um conjunto poderoso de tags JSP, escritas previamente, que integram o framework nas páginas JSP, evitando assim o uso de scriptlets (códigos Java entre `<%>` e `<%>`), deixando o código JSP mais limpo e fácil de manter.

## 4.2 Tiles

A maioria dos sites e sistemas Web obedecem geralmente um padrão de interface, variando apenas alguns pedaços (recortes) entre as páginas visualizadas pelos usuários e é neste contexto que o Tiles se encaixa no desenvolvimento de sistemas Web [30]. Tiles é um framework Open-Source que permite aos desenvolvedores Web aplicarem o padrão de projeto Composite-View. O exemplo ilustrado implica uma situação típica nos sites, onde tem-se uma estrutura padrão entre as páginas, e alguma ação do usuário altera apenas uma parte (recorte) da página.



**Figura 7 - Exemplo de Recortes Tiles: Fonte [30]**

Observa-se que entre uma página e outra somente o conteúdo central é modificado, deste modo, basta definir-se uma estrutura no Tiles com os 4 recortes do layout (Header, Menu, Body e Footer) e então, para cada página modificar somente o recorte definido na Body.



## 4.3 Hibernate

Hibernate é um framework que permite ao desenvolvedor liberta-se de preocupações com a persistência de seus objetos. Com ele, pode-se fazer o mapeamento de objetos para bancos de dados relacionais, persistindo os objetos em tabelas de um banco de dados relacional, permitindo a convivência dos dois paradigmas (relacional e objetos) [21].

O hibernate também fornece uma linguagem própria de seleção, o Hibernate Query Language (HQL) similar ao SQL, mas que possibilita a seleção de objetos ao invés de linhas. O mapeamento objeto-tabela é feito através de arquivos XML. A configuração das características do banco de dados no Hibernate pode ser feita de diversas formas como, por exemplo, através da própria interface do framework ou através de arquivos de configuração XML.

As operações de criação, atualização, remoção e seleção de objetos são feitas através da API do Hibernate que, consultando seus arquivos de mapeamento e configuração, envia os comandos corretos ao banco de dados configurado. Assim, a aplicação fica portátil entre bancos de dados.

# 5 Framework Desenvolvido

Nos capítulos anteriores, foram detalhados vários conceitos e frameworks que serviram de base para o desenvolvimento deste trabalho. Nesta seção abordamos as características e todo o processo de concepção do framework desenvolvido.

## 5.1 Introdução

Como já comentado anteriormente, o objeto deste trabalho é o desenvolvimento de um framework voltado para o desenvolvimento de sistemas Web que utilizam banco de dados como forma de persistência de informações.

Para tanto, o framework provê uma arquitetura em três camadas, obedecendo o padrão MVC de desenvolvimento de aplicações. O foco do framework está em oferecer um mecanismo automático, onde o desenvolvedor de sistemas não necessite codificar muitas linhas, que permita a persistência e recuperação de objetos em algum banco de dados. Neste contexto, o framework provê todas as funcionalidades para o desenvolvedor construir telas que permitam as operações básicas sobre um objeto, como criação, alteração, exclusão e recuperação, comumente chamadas de CRUD (Create Retrieve Update Delete).

Este framework foi desenvolvido em Java, utilizando-se de outros frameworks e ferramentas de código aberto. Nas próximas seções são abordados aspectos do desenvolvimento do framework, como ferramentas, padrões de projeto e frameworks.

## 5.2 Ferramentas Utilizadas

### 5.2.1 Eclipse

Como IDE de desenvolvimento foi utilizado o Eclipse 3.2, que é uma ferramenta muito completa e que oferece ao desenvolvedor muitas funcionalidades das quais aumentam significativamente a produtividade no processo de codificação.

Destacam-se algumas funcionalidades no Eclipse, para o desenvolvimento em Java, que são:

- Auto-complete;
- Organização de imports;
- Auto-formatação de código;
- Mapeamento de referências;
- Compilação imediata;
- Geração de trechos de códigos;
- Incorporação de plugins: TO-DO, TASKS, etc.

O Eclipse é uma ferramenta FREE e Open-Source podendo ser baixado através do site <http://www.eclipse.org>.

### 5.2.2 Tomcat

O Tomcat é um projeto do grupo Jakarta e foi desenvolvido sob a forma de implementação de referência para as API's de servlets e JSP's. Neste trabalho, foi usado como o servidor de páginas. Ele é um projeto FREE e Open-Source e pode ser baixado através do site <http://tomcat.apache.org/>.

### **5.2.3 MySQL**

Para a implementação dos projetos de teste do framework utilizou-se o banco de dados MySQL, pelo fato de ser um banco de dados gratuito e de excelente aceitação pela comunidade de desenvolvedores.

É importante ressaltar que poder-se-ia utilizar qualquer banco de dados relacional, pois a persistência dos dados é tratada utilizando-se o hibernate, que nos oferece esta flexibilidade.

## **5.3 Padrões e Características Empregadas**

Neste projeto utilizamos vários conceitos e boas práticas do mundo de desenvolvimento de softwares. Todos os conceitos foram aplicados visando garantir uma boa qualidade do código, bem como o emprego de padrões de desenvolvimento, facilitando assim, o estudo do framework por outros desenvolvedores. Abordamos ainda alguns conceitos e características empregadas no framework, mostrando também onde e como foram empregados.

### **5.3.1 Baixo Acoplamento**

Neste projeto esta característica pode ser vista quando o framework padroniza a comunicação entre as camadas do sistema, que estão distribuídas em DAO (Data Access Objects, ou camada de acesso a dados), BO (Business Objects, ou camada de lógica de negócios e a camada de controle, onde as 3 camadas possuem suas responsabilidades bem definidas, interagem entre si, mas não são dependentes umas das outras. O acoplamento entre elas é realizado através de uma anotação de classe, que atua como um agente de injeção dinâmica de dependência.

A figura abaixo ilustra este cenário:

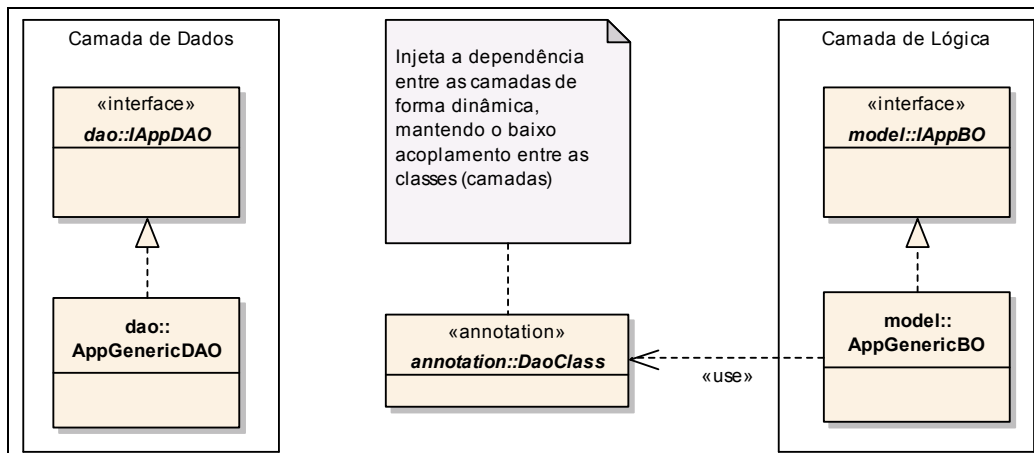


Figura 8 - Diagrama de Dependências entre as Camadas BO e DAO

### 5.3.2 Alta Coesão

Neste projeto esta característica pode ser vista quando o framework define muito claramente as responsabilidades de cada camada da aplicação. As classes de cada camada possuem apenas funcionalidades específicas das suas responsabilidades.

A figura abaixo ilustra este cenário:

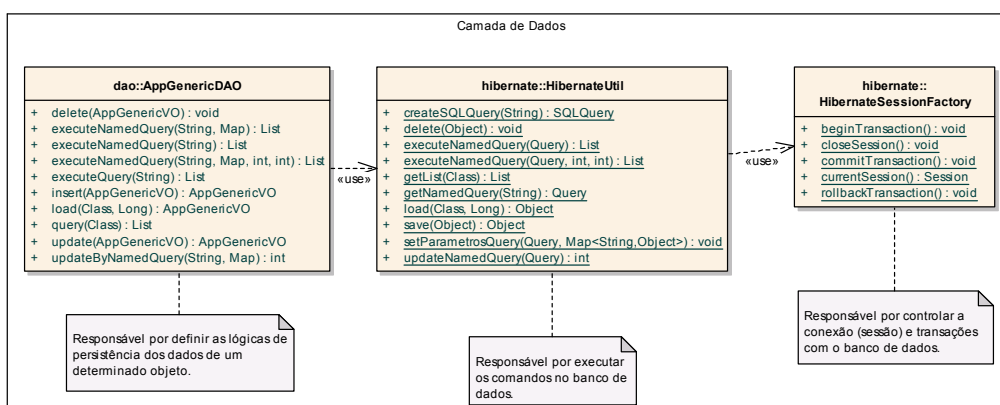


Figura 9 - Divisão de responsabilidades do framework na camada DAO

### 5.3.3 Template Method

Neste projeto este padrão foi utilizado ao definir na classe responsável pela camada de controle da aplicação (AppBaseAction) o método prepareToExecute, que permite ao desenvolvedor acrescentar regras e informações específicas do domínio da aplicação que está desenvolvendo.

O esqueleto da classe AppBaseAction e o fluxo básico de atividades desta classe pode ser observado na figura abaixo:

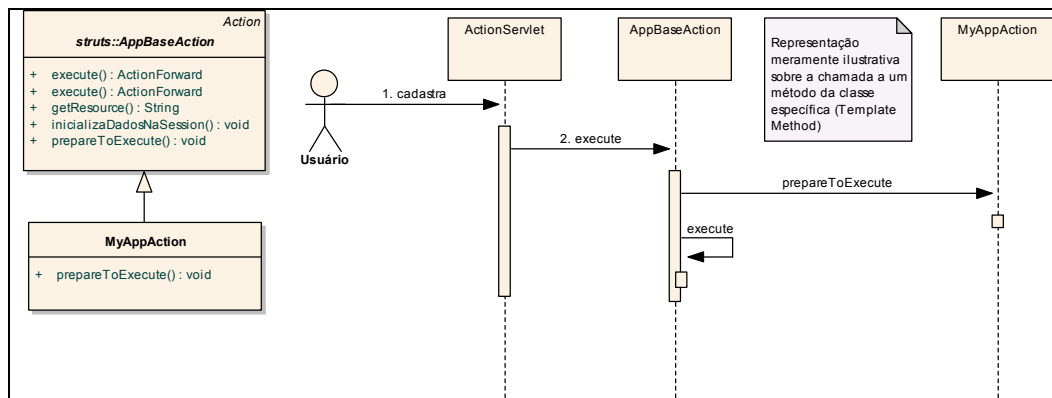


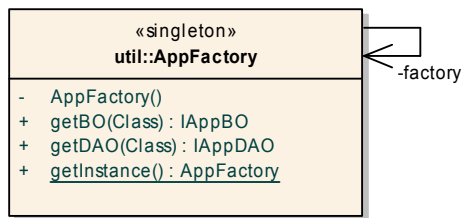
Figura 10 - Utilização do Padrão Template Method no framework

### 5.3.4 Command

Este padrão é utilizado, neste projeto, através do Struts, onde para cada ação disparada pelo usuário é executado uma action que está definida de acordo com o padrão Command.

### 5.3.5 Singleton

Neste projeto este padrão é utilizado juntamente com o padrão Factory (explicado a seguir) na classe AppFactory, que é responsável pela controle e criação de objetos da camada lógica e de acesso a dados.



**Figura 11 - Utilização do Padrão Singleton no framework**

### 5.3.6 Abstract Factory

No trabalho este padrão é utilizado juntamente com o padrão Singleton, para criar instâncias das classes responsáveis pela lógica e persistência dos dados de um determinado objeto de domínio da aplicação. Seu código pode ser visto no anexo D.4 deste documento.

### 5.3.7 Arquitetura MVC

A arquitetura Model View Controller foi utilizada neste projeto através da integração dos frameworks Struts, Tiles e Hibernate e através da definição de uma estrutura que separa claramente e sem dependências as camadas de acesso a dados, lógica de negócio e controle.

Para a camada de controle, foi utilizado o framework Struts, que realiza os controles das requisições e respostas aos clientes (browser). Na camada de visão foi utilizado HTML, JSP e Tiles sob a forma do padrão de projeto Composite View. A camada de modelo ainda ficou separada em duas sub-camadas: a de lógica de negócios e a de persistência de dados.

Na seção 5.5 serão apresentadas com mais profundidade as características e responsabilidades de cada camada, detalhando os pontos importantes do desenvolvimento do framework.

## 5.4 Categorização do Framework

Dentre as classificações básicas de frameworks podemos dizer que o objeto deste trabalho é um framework de domínio do tipo caixa-cinza. Categorizamos como framework de domínio, pois ele pretende atender um domínio específico de aplicações, que são as voltadas para Web com banco de dados.

Dizemos que é caixa-cinza porque ele mescla características do caixa-branca com o caixa-preta. É caixa-branca no sentido em que customizações das ações básicas oferecidas pelo framework podem ser feitas através da herança da classe principal da camada de controle, a `AppBaseAction`.

Da mesma maneira, pode-se dizer que é caixa-preta, pois permite a adição de novas regras de negócio e lógicas de acesso a dados, implementando as interfaces por ele disponibilizadas para cada camada.

Entretanto, seu melhor aproveitamento se dá pela utilização das características de um framework caixa-branca, pois desta forma ele provê seus serviços de forma completa, permitindo ao desenvolvedor customizar apenas algumas funcionalidades através de herança e sobrescrita de métodos.

## 5.5 Arquitetura do Framework

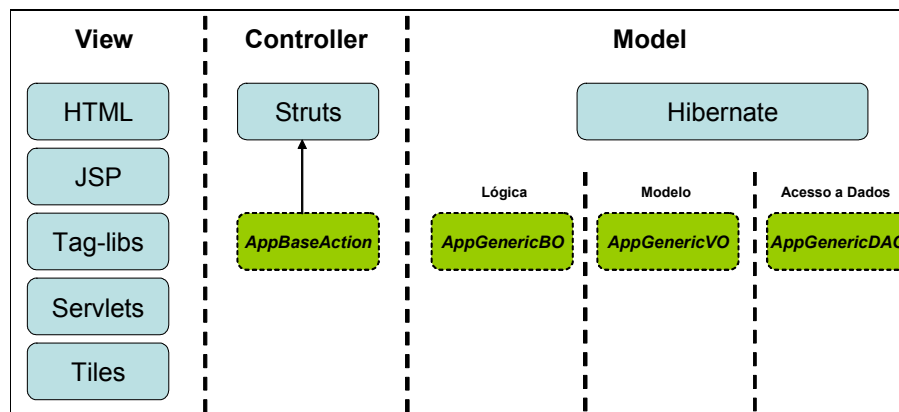
Como foi comentado anteriormente neste trabalho, o framework desenvolvido para esse projeto, obedece à arquitetura MVC, que define três camadas: Modelo, Visão e Controle. No entanto, a camada de Modelo ainda foi subdividida em outras três camadas. Assim, pode-se classificar a arquitetura como:

- **View:** define as interfaces com o usuário (HTML, JSP, Servlets, Tiles)



- **Controller:** controla as requisições e respostas entre cliente (navegador) e servidor.
- **Model:** controla as lógicas de negócio e dados. Está subdividida em:
- **Modelo:** define os objetos de domínio da aplicação
- **Lógica:** define as regras de negócio envolvidas nas operações de CRUD de cada classe do modelo.
- **Persistência:** define as regras de persistência e recuperação de cada classe do modelo.

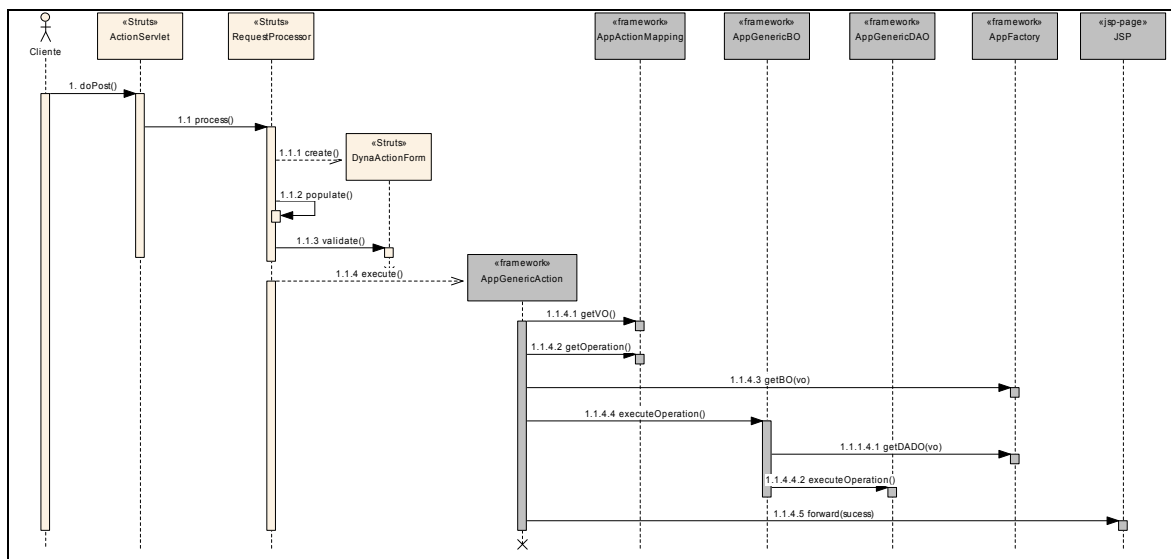
Um esquema pode ser analisado na figura seguinte:



**Figura 12 - Visão geral da Arquitetura do framework**

## 5.6 Funcionamento Geral do Framework

O diagrama de seqüências abaixo ilustra o funcionamento geral do framework:



**Figura 13 - Visão geral do funcionamento do framework**

Quando um usuário faz uma requisição, esta é interceptada pelo Struts através da ActionServlet, que inicia os controles sobre a requisição, criando um RequestProcessor e carregando as configurações do arquivo struts-config.xml.

Com base nesta configurações, uma instancia das classes ActionForm(DynaActionForm), ActionMapping (AppActionMapping) e da classe Action(AppGenericAction) são criadas e pré-configuradas de acordo com os parâmetros especificados no arquivo de configuração.

A validação das informações é feita de forma automática pelo struts através da inferência dos dados existentes na instancia da ActionForm junto às regras definidas pelo desenvolvedor, contidas no arquivo validation.xml.

Uma vez que os parâmetros da requisição foram validados, o Struts, através da implementação do padrão Command, faz uma chamada ao método execute da instancia da Action criada.

Neste momento o framework está instanciado, e começa a executar todo seu fluxo padrão. Ao executar o método execute() da AppGenericAction, o framework realiza algumas

validações e recupera alguns parâmetros das configurações específicas dele através da instancia da classe `AppActionMapping`.

Um dos principais parâmetros é o `vo`, que identifica qual é a classe principal do domínio da aplicação que será tratada na action. Outro parâmetro importante é o `operacao`, que define qual ação será realizada com o objeto (`vo`) recuperado.

Uma vez recuperados estes dois parâmetros, o sistema cria uma instancia da classe `AppGenericBO`, responsável pela implementação das regras para o objeto que está sendo tratado, e faz a chamada ao método específico conforme os parâmetros definidos pelo desenvolvedor.

Na instância da lógica do negócio, são realizadas as regras necessárias para a execução da operação e a criação da classe `AppGenericDAO`, responsável pelo acesso e persistência dos objetos. Em seguida é executado o método da classe `AppGenericDAO` correspondente a operação que está sendo realizada.

Ao final de todo o fluxo é realizado um redirecionamento para a action ou página definida no `struts-config.xml`, colocando no request todas as informações recuperadas através das operações realizadas pelo framework.

Customizações das operações realizadas pelo framework são realizadas basicamente através dos parâmetros configurados pelo desenvolvedor no arquivo `struts-config.xml` e através de especializações das classes `AppGenericBO` e `AppGenericDAO`.

## 5.7 Camada Model (M)

Como comentado anteriormente, esta camada é a responsável por definir os objetos do domínio da aplicação, regras de negócio envolvidas e regras de persistência e recuperação dos dados.

Para uma melhor estruturação do framework e garantia de baixo acoplamento e alta coesão, esta camada foi subdividida em 3 sub-camadas: Modelo, Lógica e Persistência.

### 5.7.1 Camada de Lógica

A camada de lógica é a responsável por todas as regras de negócio envolvidas com determinado objeto do domínio da aplicação, inclusive é nesta camada que se deve desenvolver toda a complexidade e interação com outros objetos e suas lógicas de negócio.

Para ficar mais clara as responsabilidades desta camada, imagine uma situação onde a persistência de um determinado objeto (objeto1) necessite que o estado de outro objeto (objeto2) seja alterado. Neste contexto, no método responsável por salvar o objeto1 deveria fazer uma chamada ao método que altera o estado o objeto2 antes da sua conclusão.

O framework implementa as regras básicas de CRUD (criação, alteração, recuperação e exclusão) de objetos através da classe AppGenericBO. O sufixo BO vem de Business Objects, ou Objetos de Negócios, qualquer regra adicional às operações básicas de um objeto, como o exemplo citado anteriormente, deverá ser realizado através de herança desta classe e sobrescrita do método apropriado.

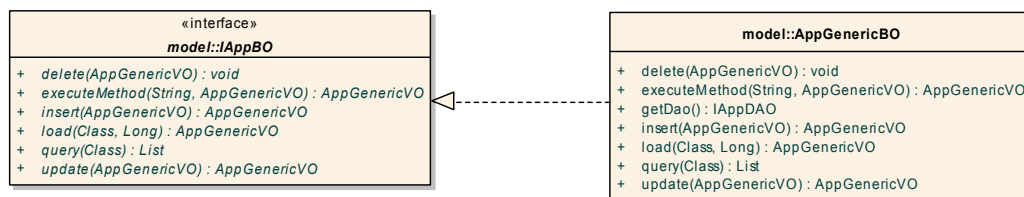


Figura 14 - Lógicas de Negócio no framework

### 5.7.2 Camada de Modelo

Nesta camada é necessário a inclusão das classes que definem os objetos do domínio do negócio, somente com seus atributos e respectivos getters e setters. Os objetos de domínio de negócios são os chamados VO's (Value Objects) podem ser vistos como as tabelas de um

banco de dados. Não necessariamente um objeto corresponderá a uma tabela ou o contrário, uma tabela corresponde a um objeto. Este mapeamento é realizado através das anotações do hibernate, e podem existir vários cenários, como por exemplo, o mapeamento de um objeto para mais de uma tabela.

O framework define uma estrutura padrão de objeto através da classe `AppGenericVO`. Todo objeto de domínio de uma aplicação deve ser definido em uma classe que estenda a classe `AppGenericVO`. O exemplo de um objeto de domínio de negócio pode ser visto abaixo:

```
@Entity
@Table(name = "USER")
@SequenceGenerator(name="SEQ_ID", sequenceName="SEQ_ID")
public class User extends AppGenericVO {

    @Id
    @GeneratedValue(strategy = javax.persistence.GenerationType.AUTO)
    @Column(name = "ID")
    protected Long id;

    @Column(name = "NAME", nullable = false, length = 50)
    private String name;

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

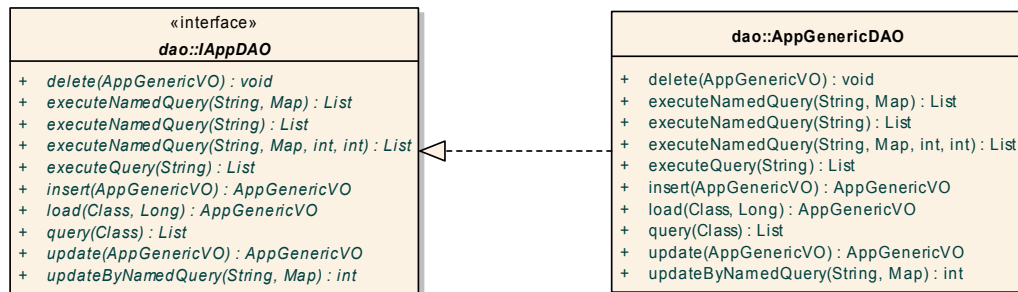
    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

### 5.7.3 Camada de Acesso a Dados

Esta camada, como diz seu nome, é a responsável pelo acesso e persistência dos objetos em um banco de dados, ela mantém a conexão com o banco de dados, realiza as consultas e define os algoritmos para realizar qualquer operação básica de CRUD de objetos.

Esta camada utiliza-se de todos os recursos do framework hibernate para fazer o acesso e persistência dos objetos no banco de dados.



**Figura 15 - Acesso e Persistência de dados no framework**

## 5.8 Camada Visão (V)

A camada de visão é a menos trabalhada neste framework pois com ela existe dificuldade de se padronizar interfaces de sistema. Entretanto, o framework utiliza-se de outros frameworks e bibliotecas para facilitar os controles de interface. Nesta camada são utilizadas bibliotecas para facilitar a exibição de objetos e listas e o framework Tiles para controlar os leiautes da aplicação.

## 5.9 Camada Controle (C)

Esta é a camada responsável por receber as requisições dos usuários e respondê-las de forma adequada, através de chamadas aos objetos da camada de lógica. Para o suporte a camada de controle o framework disponibiliza a classe `AppBaseAction`, que estende a classe `Action` do struts, oferecendo todas as opções necessárias ao controle das operações de CRUD de um objeto.

Para cada operação precisa ser definida uma action no struts, que baseia-se na classe `AppBaseAction`. Esta classe ainda trabalha em conjunto com a classe `AppActionMapping` que

é a responsável pelas opções de configuração da AppBaseAction. Para cada action do framework podem ser definidos os seguintes parâmetros:

Parâmetro	Descrição
<i>vo</i>	Nome completo da classe de domínio que será objeto de uma das operações CRUD
<i>operacao</i>	Define a operação a ser realizada com o objeto de definido no parâmetro vo. As operações disponíveis são: load – carrega um objeto baseado no seu id. update – altera os dados de um objeto insert – insere um novo objeto no banco de dados. delete – exclui um objeto
<i>nextAction</i>	Define a próxima ação a ser disparada depois da ação corrente
<i>limpaForm</i>	limpa os todos os dados do formulário passado a action
<i>executeNamedQuery</i>	Define qual consulta deve ser executada para recuperar informações dos objetos.
<i>metodo</i>	Define a chamada a algum método específico implementado em uma classe especializada pelo desenvolvedor.
<i>carregaLista</i>	Define se deve ser carregado alguma lista no request. É utilizado quando deseja-se montar uma combobox ou lista com objetos recuperados do banco de dados.

A seguir demonstra-se um exemplo de uma configuração de uma action que recupera um objeto do banco de dados baseado em seu id (código):

```
<action path="/carregarequipamento"
  className="br.com.ibarra.web.struts.AppActionMapping"
  name="equipamentoCadForm" scope="request"
  validate="false"
  type="br.com.ibarra.web.struts.actions.AppGenericAction">
  <set-property property="vo"
    value="tcc.sgeem.vo.Equipamento" />
  <set-property property="operacao"
    value="load" />
  <set-property property="nextAction"
    value="/alterarequipamento.do" />
  <forward name="sucesso" path="equipamento.cadastro" />
</action>
```

## 5.10 Principais Classes

### 5.10.1 **AppBaseAction**

Esta classe é a responsável pelo controle e tratamento dos requisitos e respostas entre os usuários e o servidor, ela é uma especialização da classe Action do struts, acrescentados os controles e funcionalidades necessários para o tratamento das operações básicas de CRUD de objetos.

### 5.10.2 **AppActionMapping**

Esta classe permite ao desenvolvedor definir todos os parâmetros referentes às operações automatizadas pelo framework através do complemento de parâmetros nas configurações das actions tratadas pelo Struts.

Os parâmetros padronizados pelo framework são recuperados do arquivo struts-config.xml e passados para a classe AppBaseAction e demais classes da camada de controle, que com base nestes parâmetros realizam os controles necessários as operações básicas de CRUD oferecidas pelo framework.



<i>ActionMapping</i>	
<b>struts::AppActionMapping</b>	
+ aplicaSeguranca() : boolean	
+ findForwardAddDirectory(String, String) : ActionForward	
+ getCarregarLista() : String	
+ getContentControl() : String	
+ getLimpaForm() : boolean	
+ getListaDinamica() : Set<AppGenericVO>	
+ getMetodo() : String	
+ getNamedQuery() : String	
+ getNextAction() : String	
+ getOperacao() : String	
+ getParametrosQuery() : List<String>	
+ getRequestAttributeNamedQuery() : String	
+ getVo() : String	
+ getVOClass() : Class	
+ getVOContentControls() : List<VOContentControl>	
+ getVOInstance() : AppGenericVO	
+ setCarregarLista(String) : void	
+ setContentControl(String) : void	
+ setExecuteNamedQuery(String) : void	
+ setLimpaForm(String) : void	
+ setMetodo(String) : void	
+ setNextAction(String) : void	
+ setOperacao(String) : void	
+ setVo(String) : void	

**Figura 16 - Classe AppActionMapping**

### 5.10.3 AppGenericBO

Esta classe é a que define as regras gerais para tratamento das operações básicas de CRUD de objetos, servindo como uma ponte entre a camada de controle e a camada de acesso a dados.

Através da especialização dela é possível adicionar novas regras e operações de sob determinado objeto. Por exemplo, pode-se criar um método especial para só alterar um determinado valor de um atributo de um objeto e persistir este valor. Este método pode ser configurado através do parâmetro método, disponibilizado pela classe AppActionMapping, para ser chamado automaticamente pelo framework.

## 5.10.4 AppGenericDAO

Nesta classe estão definidas todas as formas de acesso e mecanismos de persistência de um determinado objeto, através da especialização desta classe, pode-se definir novas regras de persistência de um determinado objeto num nível mais baixo, inclusive referenciando-se diretamente a tabelas do banco de dados. Deve utilizar este tipo de especialização mediante necessidade extrema, pois deve-se sempre preferir as customizações em nível de tratamento de objetos, ou seja, na camada de regra de negócios através da especialização da classe AppGenericBO.

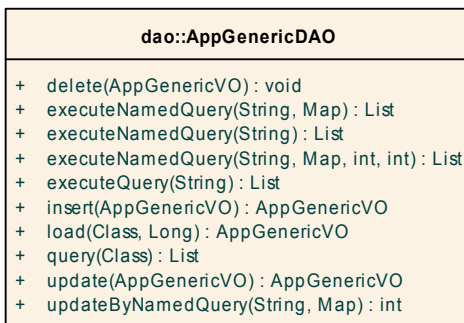


Figura 17 - Classe AppGenericDAO

## 5.10.5 AppGenericVO

Esta é uma classe muito simples do framework e que não apresenta lógica alguma. Ela simplesmente define uma estrutura básica de um objeto de domínio de aplicação, permitindo assim ao framework realizar tratamentos genéricos em objetos distintos (polimorfismo).

Basicamente ela define que para cada classe de domínio precisa existir um atributo id que deverá ser numérico. Este atributo será o utilizado pelo banco de dados como campo de chave primária das tabelas correspondentes aos objetos. Todo o tratamento sobre um objeto é realizado através deste campo.

## 5.10.6 DaoClass Annotation

Com esta classe de anotação é possível definir para uma determinada classe da camada de lógica de negócios qual classe da camada de acesso aos dados ela vai utilizar para realizar suas operações de acesso e persistência dos objetos no banco de dados, fazendo uma injeção de dependência de forma dinâmica mantendo o baixo acoplamento entre as classes de camadas diferentes.

## 5.10.7 BOClass Annotation

Esta anotação possibilita a definição para uma determinada classe de domínio da aplicação (VO) qual classe de lógica de negócios implementa suas operações básicas de CRUD e operações adicionais, fazendo uma injeção de dependência de forma dinâmica mantendo o baixo acoplamento entre as classes de camadas diferentes.

## 5.10.8 AppFactory

Esta classe é uma implementação do padrão Abstract Factory, e permite a criação de objetos tanto da camada de lógica de negócio (BO's) quanto da camada de acesso a dados (DAO's), a partir de uma classe de domínio (VO). Seu mecanismo de definição é baseado nas anotações DaoClass e BOClass, definidas no itens 5.7.6 e 5.7.7 respectivamente.

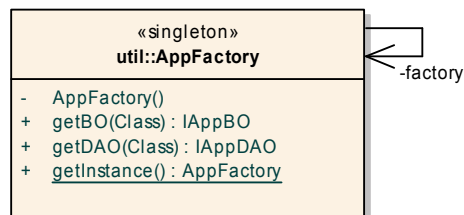


Figura 18 - Classe AppFactory

# 6 Estude de Caso - SGEL

## 6.1 Introdução

A tendência atual de grande parte das indústrias é produzir componentes mecânicos dentro de tolerâncias cada vez mais estreitas. No entanto, é necessário investir em pesquisa e desenvolvimento de sistemas de medição que satisfaçam os níveis de precisão desejados, garantindo a qualidade dimensional dos produtos.

No Laboratório de Metrologia e Automatização (Labmetro), parte do Departamento de Engenharia Mecânica da Universidade Federal de Santa Catarina (UFSC), internacionalmente reconhecido como um dos grandes centros de excelência do país são desenvolvidos trabalhos onde a ênfase está na metrologia mecânica, embora vários princípios estejam envolvidos.

Ele procura abordar aspectos ligados à avaliação da incerteza de medição, em diferentes níveis de profundidade, ao desenvolvimento de técnicas e sistemas de medição avançados. Pois os instrumentos normalmente utilizados em procedimentos de medição, apresentam erros sistemáticos, da mesma forma, o processo de medição é influenciado por erros aleatórios decorrentes de mudanças ambientais, de vibração e da atuação do operador.

Então, o Labmetro, além de adicionar uma componente tecnológica ao trabalho, o suporte e a participação do setor industrial tem viabilizado a concretização das soluções técnicas inéditas de problemas reais da indústria brasileira.

Para tanto, utilizam-se materiais que possibilitem a realização dos trabalhos desenvolvidos no laboratório, sendo eles de grande importância. Portanto, é necessário um instrumento que facilite o gerenciamento de empréstimos dos mesmos, agilizando sua localização e controle sobre a requisição destes.

## 6.2 Visão Geral do Sistema

Há uma grande procura dos diversos materiais disponibilizados no Labmetro, como placas, gama de pastilhas, paquímetro, serras entre outros, porém existe um controle manual realizado em papel que dificulta a atualização e pesquisa dos mesmos, onde a entrega desses materiais é falho.

Devido a este problema, se faz necessário uma ferramenta capaz de facilitar a atualização e a pesquisa, mantendo o controle dos materiais emprestados, evitando furtos e danos.

Assim, a implementação de um software vem como um meio de organizar e armazenar esses dados, sendo uma ferramenta rápida, segura e eficaz sobre a procura e controle desses materiais.

O software terá uma interface Web que controla o processo de empréstimo dos materiais do laboratório. Serão necessários três níveis de acesso: Administrador, Operador e Usuário. Para o acesso ao software SGEL será necessário uma identificação com login e senha. Identificando o nível de acesso Operador, o mesmo terá como funções cadastrar, consultar, excluir, alterar, emprestar, devolver, notificar atraso de devolução de empréstimos e alterar seus dados de cadastro. Já para nível de acesso Usuário poderá somente alterar seu dados de cadastro e listar seus empréstimos. O nível de acesso Administrador terá todas as funções dos outros níveis e também cadastrar, consultar, excluir e alterar usuários.

## 6.3 Atores do Sistema

### 6.3.1 Administrador

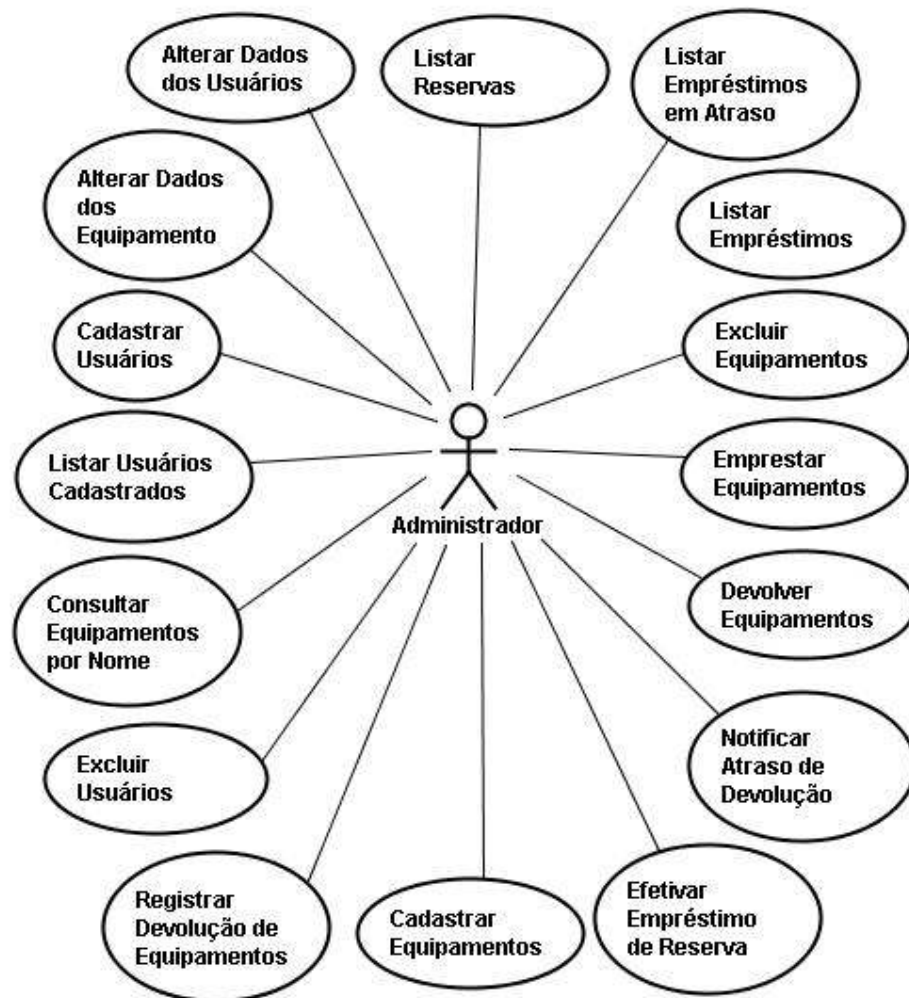


Figura 19 - Diagrama de Casos de Uso do Administrador

### 6.3.2 Operador

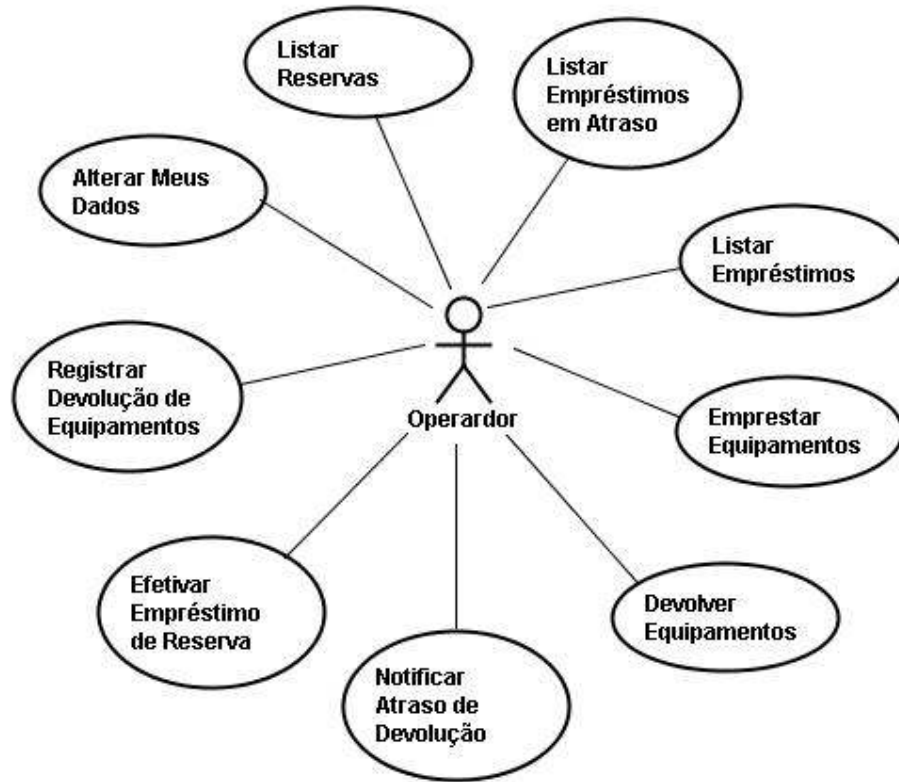


Figura 20 - Diagrama de Caso de Uso Operador

### 6.3.3 Usuário

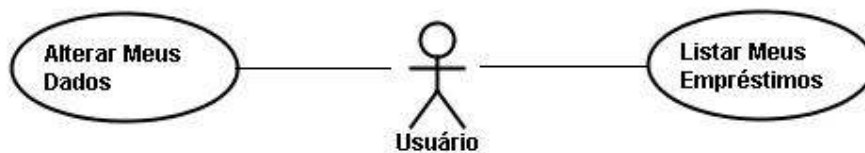


Figura 21 - Diagrama de Caso de Uso Usuário

## 6.4 Casos de Uso

### 6.4.1 Caso de Uso: Cadastro de Usuários

Ator Primário: Administrador

Objetivo: Cadastrar usuários no sistema

Pré-Condições: Usuário autenticado no sistema como Administrador

Pós-Condições: Usuário registrado no sistema.

Fluxo Básico:

1. O Administrador informa: nome, e-mail, login, senha, confirmação de senha, telefone, celular e perfil do usuário.
2. O sistema registra as informações do usuário.

Fluxos Alternativos:

- 1.1 Se o administrador não informar algum dos campos nome, e-mail, login, senha e confirmação de senha descritos no passo
- 1.2 O sistema emite um aviso de que os campos são obrigatórios. Se o administrador informar um valor diferente nos campos senha e confirmação de senha o sistema deve emitir um aviso de que os valores dos campos são diferentes.

### 6.4.2 Caso de Uso: Listar Usuários Cadastrados

Ator Primário: Administrador

Objetivo: Listar usuários cadastrados no sistema

Pré-Condições: Usuário autenticado no sistema como Administrador

Fluxo Básico:



5. O sistema informa uma lista dos usuários cadastrados.

### **6.4.3 Caso de Uso: Alterar Dados dos Usuários**

Ator Primário: Administrador

Objetivo: Alterar os dados dos usuários no sistema

Pré-Condições: Usuário autenticado no sistema como Administrador

Pós-Condições: Dados dos usuários atualizados no sistema

Fluxo Básico:

1. O sistema informa uma lista de usuários cadastrados.
2. O Administrador seleciona o usuário que deseja alterar os dados.
3. O Administrador altera os campos desejados.
4. O sistema atualiza os dados do usuário.

Fluxos Alternativos:

Se o administrador não informar algum dos campos nome, e-mail, login, senha e confirmação de senha, o sistema emite um aviso de que os campos são obrigatórios.

Se o administrador informar um valor diferente nos campos senha e confirmação de senha o sistema deve emitir um aviso de que os valores dos campos são diferentes.

### **6.4.4 Caso de Uso: Excluir Usuários**

Ator Primário: Administrador

Objetivo: Excluir o cadastro dos usuários no sistema

Pré-Condições: Usuário autenticado no sistema como Administrador

Pós-Condições: Cadastro do usuário excluído do sistema

Fluxo Básico:

1. O sistema informa uma lista de usuários cadastrados.

2. O Administrador seleciona o usuário que deseja excluir o cadastro.
3. O sistema exclui o cadastro do usuário.

### **6.4.5 Caso de Uso: Cadastrar Equipamentos**

Ator Primário: Administrador

Objetivo: Cadastrar equipamentos no sistema

Pré-Condições: Usuário autenticado no sistema como Administrador

Pós-Condições: Equipamento registrado no sistema.

Fluxo Básico:

1. O Administrador informa: código, nome, fabricante, modelo, número de série, número de patrimônio, local e observações.
2. Sistema registra as informações do equipamento.

Fluxos Alternativos:

- 1.1 Se o administrador não informar algum dos campos código, nome, fabricante e modelo no passo 1, o sistema emite um aviso de que os campos são obrigatórios.

### **6.4.6 Caso de Uso: Consultar Equipamento por Nome**

Ator Primário: Administrador

Objetivo: Consultar equipamento no sistema

Pré-Condições: Usuário autenticado no sistema como Administrador

Fluxo Básico:

1. O Administrador informa o nome do equipamento.
2. O sistema lista os equipamentos cadastrados com o nome informado.

## **6.4.7 Caso de Uso: Alterar Dados dos Equipamentos**

Ator Primário: Administrador

Objetivo: Alterar os dados de equipamentos no sistema

Pré-Condições: Usuário autenticado no sistema como Administrador

Pós-Condições: Cadastro de equipamentos atualizado no sistema

Fluxo Básico:

1. O sistema informa uma lista de equipamentos cadastrados.
2. O Administrador seleciona o equipamento que deseja alterar o cadastro.
3. O Administrador altera os campos desejados.
4. O sistema atualiza os dados do usuário.

Fluxos Alternativos:

- 1.1 Se o administrador não informar algum dos campos código, nome, fabricante e modelo, o sistema emite um aviso de que os campos são obrigatórios.

## **6.4.8 Caso de Uso: Excluir Equipamentos**

Ator Primário: Administrador

Objetivo: Excluir o cadastro dos equipamentos no sistema

Pré-Condições: Usuário autenticado no sistema como Administrador

Pós-Condições: Cadastro do equipamento excluído do sistema

Fluxo Básico:

1. O sistema informa uma lista de equipamentos cadastrados.
2. O Administrador seleciona o equipamento que deseja excluir o cadastro.
3. O sistema exclui o cadastro do equipamento.

## **6.4.9 Caso de Uso: Emprestar Equipamento**

Atores: Administrador e Operador

Objetivo: Emprestar equipamento para um usuário do sistema

Pré-Condições: Usuário autenticado no sistema como Administrador ou Operador

Pós-Condições: Cadastra o empréstimo do equipamento no sistema

Fluxo Básico:

1. O Administrador ou Operador informa o nome do equipamento para o empréstimo.
2. O sistema lista os equipamentos cadastrados com o nome informado.
3. O Administrador ou Operador informa o equipamento que deseja emprestar.
4. O Administrador ou Operador informa os campos solicitante, data de devolução e observações.
5. O sistema registra o empréstimo.

## **6.4.10 Caso de Uso: Devolver Equipamento**

Atores: Administrador e Operador

Objetivo: Devolver um equipamento emprestado no sistema

Pré-Condições: Usuário autenticado no sistema como Administrador ou Operador

Pós-Condições: Cadastra a devolução do equipamento no sistema

Fluxo Básico:

1. O sistema informa uma lista de equipamentos emprestados.
2. O Administrador ou Operador seleciona o equipamento que deseja devolver.
3. O sistema registra a devolução do equipamento.

### **6.4.11 Caso de Uso: Notificar Atraso de Devolução**

Atores: Administrador e Operador

Objetivo: Notificar um usuário que possui um equipamento emprestado em atraso

Pré-Condições: Usuário autenticado no sistema como Administrador ou Operador

Pós-Condições: Envia um e-mail ao usuário notificando o atraso

Fluxo Básico:

1. O sistema informa uma lista de usuários com equipamentos em atraso.
2. O Administrador ou Operador seleciona o usuário que deseja notificar.
3. O sistema envia um e-mail para o usuário.

### **6.4.12 Caso de Uso: Efetivar Empréstimo de Reserva**

Atores: Administrador e Operador

Objetivo: Cadastrar o empréstimo do equipamento reservado para um usuário.

Pré-Condições: Usuário autenticado no sistema como Administrador ou Operador

Pós-Condições: Cadastra o empréstimo no sistema

Fluxo Básico:

1. O sistema informa uma lista de equipamentos com reserva.
2. Administrador ou Operador seleciona o equipamento que deseja emprestar.
3. O sistema registra o empréstimo.

### **6.4.13 Caso de Uso: Listar Empréstimos**

Atores: Administrador e Operador

Objetivo: Listar empréstimos cadastrados no sistema

Pré-Condições: Usuário autenticado no sistema como Administrador ou Operador

Fluxo Básico:

1. O sistema informa uma lista de empréstimos no sistema.

#### **6.4.14 Caso de Uso: Listar Reservas**

Atores: Administrador e Operador

Objetivo: Listar reservas cadastradas no sistema

Pré-Condições: Usuário autenticado no sistema como Administrador ou Operador

Fluxo Básico:

1. O sistema informa uma lista de reservas no sistema.

#### **6.4.15 Caso de Uso: Listar Empréstimos em Atraso**

Atores: Administrador e Operador

Objetivo: Lista empréstimos em atraso no sistema

Pré-Condições: Usuário autenticado no sistema como Administrador ou Operador

Fluxo Básico:

1. O sistema informa uma lista de empréstimos em atraso no sistema.

#### **6.4.16 Caso de Uso: Registrar Devolução de Equipamento**

Atores: Administrador e Operador

Objetivo: Cadastrar devolução do equipamento no sistema

Pré-Condições: Usuário autenticado no sistema como Administrador ou Operador

Fluxo Básico:

1. O sistema informa uma lista de empréstimos.
2. O Administrador ou Operador seleciona o equipamento que deseja devolver.

3. O sistema registra a devolução.

### **6.4.17 Caso de Uso: Alterar Meus Dados**

Atores: Operador e Usuário

Objetivo: Alterar os meus dados no sistema

Pré-Condições: Usuário autenticado no sistema como Operador ou Usuário

Pós-Condições: Dados do usuário atualizado no sistema

Fluxo Básico:

1. O sistema informa os dados do usuário.
2. O Operador ou Usuário seleciona o que deseja alterar nos dados.
3. O sistema atualiza os dados do usuário.

Fluxos Alternativos:

Se o Operador ou Usuário não informar algum dos campos nome, e-mail, login, senha e confirmação de senha, o sistema emite um aviso de que os campos são obrigatórios.

Se o Operador ou Usuário informar um valor diferente nos campos senha e confirmação de senha o sistema deve emitir um aviso de que os valores dos campos são diferentes.

### **6.4.18 Caso de Uso: Listar Meus Empréstimos**

Atores: Operador e Usuário

Objetivo: Listar meus empréstimos no sistema.

Pré-Condições: Usuário autenticado no sistema como Operador ou Usuário

Fluxo Básico:

1. O sistema informa uma lista de empréstimos.

## 6.5 Classes Conceituais do Sistema

O domínio do SGEL é extremamente simples, e envolve apenas 3 conceitos: Usuário, Equipamento e Empréstimo.

Usuário é toda a pessoa que pode utilizar o sistema. Cada usuário possui um perfil que pode ser Administrador, Operador e Usuário Comum.

Equipamentos são todos os equipamentos do LABMETRO que são passíveis de serem emprestados para seus outras pessoas, sejam funcionários ou não.

Empréstimo define qual usuário está com qual equipamento, mantendo as datas de empréstimo e devolução, além de outras informações.

O diagrama abaixo mostra as associações entre estas classes:

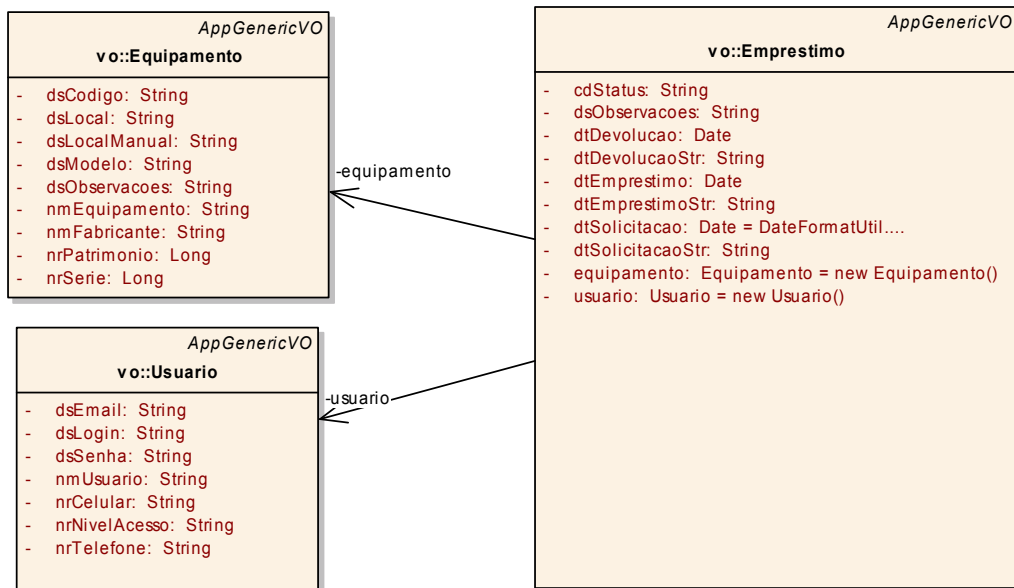


Figura 22 - Diagrama de Classes Conceituais do SGEL



## 6.6 Implementação

A implementação do SGEL foi a parte mais importante e especial deste trabalho, pois neste momento tivemos todas as provas que os conceitos e funcionalidades desenvolvidas no framework funcionavam como esperado.

Para sua implementação partimos da especificação dos casos de uso e modelagem das classes de domínio da aplicação cujo código fonte pode ser visto no Anexo I deste documento.

O segundo passo foi desenvolver uma página JSP para cada caso de uso capaz de apresentar os dados correspondentes aos objetos e informações tratadas nele.

Em seguida foi analisado qual classe de domínio estava envolvida em cada caso de uso, e qual operação deveria ser realizada, tentando sempre encaixar nas operações básicas de CRUD disponibilizadas pelo framework: load, update, insert, delete.

Neste ponto, observamos que a maioria dos casos de uso encaixavam-se diretamente nas funcionalidades providas pelo framework. O resultado da análise pode ser visto na tabela abaixo:

Caso de Uso	Necessitou Especialização?	
	Sim	Não
cadastrar usuários		X
listar usuários cadastrados		X
alterar dados dos usuários		X
excluir usuários		X
cadastrar equipamentos		X
consultar equipamentos (por nome)		X
alterar dados dos equipamentos		X
excluir equipamento		X
emprestar equipamento	X	
listar empréstimos		X
listar reservas		X
listar empréstimos em atraso		X
registrar devolução de equipamento	X	
excluir empréstimo	X	
notificar atraso de devolução	X	

efetivar empréstimo de reserva	X	
alterar meus dados		X
listar meus empréstimos		X

Analisando a tabela acima, podemos perceber que de um total de 18 casos de uso do sistema, apenas 5 precisaram de algum tipo de customização através de especialização das classes do framework o que nos dá um índice de 72,2% de cobertura das funcionalidades pelo framework, caracterizando assim um ganho de produtividade considerável.

Podemos ainda observar que os casos em que foram necessárias algum tipo de customização são todos referente a lógica de empréstimos de equipamentos, que é justamente onde concentra-se o objeto de negócio da aplicação. O nome do sistema em estudo é Sistema de Gerenciamento de Empréstimos do Labmetro, ou seja, o controle é sobre empréstimos, e nada mais natural do que este ponto não ser coberto de maneira completa por um framework cujo domínio é genérico e voltado a aplicações Web com banco de dados.

No entanto esta customização foi simples, e bastou definir regras de negócio específicas para cada caso de uso, que se resumiu na criação de uma classe EmpréstimoBO que especializa AppGenericBO, acrescentando métodos específicos para cada caso de uso.

O anexo II deste documento mostra uma relação dos casos de uso que exigiram alguma customização, o motivo pelo qual não se encaixou nos controles básicos oferecidos pelo framework, e a solução adotada para resolução do problema.

O anexo III mostra todo o código necessário para a customização das funcionalidades do framework de forma a atender aos casos de uso.

O anexo IV mostra o arquivo struts-config, onde são configuradas todas as ações correspondentes aos casos de uso e os parâmetros específicos para utilização do framework em cada caso.

## 6.7 Telas do Sistema

### 6.7.1 Tela Inicial

login:

senha:

S.G.E.E.M

Figura 23 - Tela inicial do SGEL

### 6.7.2 Tela Principal do Administrador

cadastro de usuários 1

equipamentos & materiais 2

empréstimos 3

novo empréstimo 4

devolução 5

sair 6

S.G.E.E.M

Figura 24 - Tela Principal do Administrador do SGEL

### 6.7.3 Tela Principal do Operador



Figura 25 - Tela Principal do Operador so SGEL

### 6.7.4 Tela Principal do Usuário Comum



Figura 26 - Tela Principal do Usuario Comum do SGEL

## 6.7.5 Lista de Usuários do Sistema (Admin)

**SGEL**  
GERENCIAMENTO DE EMPRÉSTIMO  
DE EQUIPAMENTOS DO LABMETRO

- cadastro de usuários 1
- equipamentos & materiais 2
- empréstimos 3
- novo empréstimo 4
- devolução 5
- sair 6

**Usuários Cadastrados no Sistema** [Novo Usuário](#)









ID	Nome	E-mail	Telefone	Celular	Tipo Usuário	Ações
1	ADMINISTRADOR DO SGEEM	root@labmetro.ufsc.br			Administrador	 
6	ANALUCIA VIEIRA FANTIN	avf@labmetro.ufsc.br	32392034		Usuário	 
7	ARMANDO ALBERTAZZI GONCALVES JUNIOR	albertazzi@labmetro.ufsc.br	32392034		Usuário	 
5	DANIEL PEDRO WILLEMANN	willemann@labmetro.ufsc.br		96115885	Usuário	 
2	DAVID PEDRO WILLEMANN	dww@labmetro.ufsc.br	32392161	99197836	Administrador	 
4	FABRICO LUIS BROERING	flb@labmetro.ufsc.br	32392151		Operador	 
3	GUSTAVO BESTTETI IBARRA	gbi@labmetro.ufsc.br		91047574	Administrador	 

Figura 27 - Lista de Usuários do Sistema do SGEL

## 6.7.6 Cadastro de Usuários

The screenshot shows the 'Cadastro de Usuário' (User Registration) page of the SGEL system. The page layout includes a blue header with the SGEL logo and the text 'GERENCIAMENTO DE EMPRÉSTIMO DE EQUIPAMENTOS DO LABMETRO'. A navigation menu on the right lists the following options: 'cadastro de usuários' (1), 'equipamentos & materiais' (2), 'empréstimos' (3), 'novo empréstimo' (4), 'devolução' (5), and 'sair' (6). The main content area is titled 'Cadastro de Usuário' and includes a link for 'Consultar Usuários | Novo Usuário'. The registration form contains the following fields:

- Nome:
- E-mail:
- Login:
- Senha:
- Confirmação de Senha:
- Telefone:
- Celular:
- Perfil do Usuário:

At the bottom of the form are two buttons: 'Salvar' and 'Cancelar'.

Figura 28 - Cadastro de Usuários do SGEL

## 6.7.7 Consultar Equipamentos

The screenshot shows the SGEL (Sistema de Gerenciamento de Empréstimo de Equipamentos do Labmetro) interface. The top left features the SGEL logo and the text "GERENCIAMENTO DE EMPRÉSTIMO DE EQUIPAMENTOS DO LABMETRO". The top right has a navigation menu with the following items:

- cadastro de usuários 1
- equipamentos & materiais 2
- empréstimos 3
- novo empréstimo 4
- devolução 5
- sair 6

The main content area is titled "Consulta Equipamentos" and includes a "Novo Equipamento" link. Below the title is a search form with a text input field labeled "Nome" and a "Consultar" button.

Below the search form is a table displaying equipment records:

ID	Nome	Código	Fabricante	Modelo	Patrimônio	dsLocal	Ações
11	OPTICAL SPECTRUM ANALYSER DISPLAY UNIT	OPT 0011	ANRITSU	MS9030A	0	LABORATORIO	 
12	OPTICAL SPECTRUM ANALYSER OPTICAL UNIT	OPT 0012	ANRITSU	M59701C	0	LABORATORIO	 

Figura 29 - Consulta de Equipamentos do SGEL

## 6.7.8 Cadastro de Equipamentos

The screenshot displays the 'Cadastro de Equipamento' (Equipment Registration) form within the SGEL system. The top navigation bar features the SGEL logo and the text 'GERENCIAMENTO DE EMPRÉSTIMO DE EQUIPAMENTOS DO LABMETRO'. A menu on the right side lists the following options: 'cadastro de usuários' (1), 'equipamentos & materiais' (2), 'empréstimos' (3), 'novo empréstimo' (4), 'devolução' (5), and 'sair' (6). The main content area is titled 'Cadastro de Equipamento' and includes a link for 'Consultar Equipamentos' and a highlighted link for 'Novo Equipamento'. The form contains the following fields: 'Código', 'Nome', 'Fabricante', 'Modelo', 'Nro. de Série', 'Nro. de Patrimônio', 'Local', and 'Observações'. At the bottom of the form are 'Salvar' and 'Cancelar' buttons.

Figura 30 - Cadastro de Equipamentos do SGEL



## 6.7.9 Empréstimos

**SGEL**  
GERENCIAMENTO DE EMPRÉSTIMO  
DE EQUIPAMENTOS DO LABMETRO

cadastro de usuários 1  
equipamentos & materiais 2  
empréstimos 3  
novo empréstimo 4  
devolução 5  
sair 6

**Lista de Empréstimos** Empréstimos | Reservados | Atrasados

ID	Recurso	Usuário	Data	Devolução	Situação	Observações	Ações
1	FURADEIRA MANUAL	DANIEL PEDRO WILLEMANN	18/06/2007	20/06/2007	EMPRESTADO	FURADEIRACOMPLETA	  
2	DREMEL	DAVID PEDRO WILLEMANN	18/06/2007	20/07/2007	EMPRESTADO	NORMAL	  
6	MULTIMETRO DIGITAL	ADMINISTRADOR DO SGEEM	19/06/2007	18/06/2007	EMPRESTADO		  
8	TERMOMETRO DIGITAL	DAVID PEDRO WILLEMANN	19/06/2007	18/06/2007	EMPRESTADO		  

Figura 31 - Lista de Empréstimos do SGEL

## 6.7.10 Novo Empréstimo (Passo 1)

**SGEL**  
GERENCIAMENTO DE EMPRÉSTIMO  
DE EQUIPAMENTOS DO LABMETRO

- cadastro de usuários 1
- equipamentos & materiais 2
- empréstimos 3
- novo empréstimo 4
- devolução 5
- sair 6

### Empréstimo de Equipamentos

Nome

ID	Nome	Código	Fabricante	Modelo	Patrimônio	dsLocal	Ações
11	OPTICAL SPECTRUM ANALYSER DISPLAY UNIT	OPT 0011	ANRITSU	MS9030A.0		LABORATORIO	
12	OPTICAL SPECTRUM ANALYSER OPTICAL UNIT	OPT 0012	ANRITSU	M59701C.0		LABORATORIO	

Figura 32 - Novo Empréstimo: Passo 1

## 6.7.11 Novo Empréstimo (Passo 2)

The screenshot displays the SGEL (Sistema de Gerenciamento de Empréstimo de Equipamentos do Labmetro) interface. The header features the SGEL logo and the text "GERENCIAMENTO DE EMPRÉSTIMO DE EQUIPAMENTOS DO LABMETRO". A navigation menu on the right lists the following options: cadastro de usuários (1), equipamentos & materiais (2), empréstimos (3), novo empréstimo (4), devolução (5), and sair (6). The main content area is titled "Novo Empréstimo" and contains the following form fields:

- Solicitante: ANALUCIA VIEIRA FANTIN (dropdown menu)
- Recurso: OPTICAL SPECTRUM ANALYSER DISPLAY UNIT
- Data para Empréstimo: 21/06/2007
- Data de Devolução: (empty text box)
- Observações: (empty text box)

At the bottom of the form are two buttons: "Cadastrar" and "Cancelar".

Figura 33 - Novo Empréstimo Passo 2 do SGEL

## 6.7.12 Devolução de Equipamento

**SGEL**  
GERENCIAMENTO DE EMPRÉSTIMO  
DE EQUIPAMENTOS DO LABMETRO

- cadastro de usuários 1
- equipamentos & materiais 2
- empréstimos 3
- novo empréstimo 4
- devolução 5
- sair 6

### Devolução de Recurso

ID	Recurso	Usuário	Data	Devolução	Situação	Observações	Ações
1	FURADEIRA MANUAL	DANIEL PEDRO WILLEMANN	18/06/2007	20/06/2007	EMPRESTADO	FURADEIRACOMPLETA	  
2	DREMEL	DAVID PEDRO WILLEMANN	18/06/2007	20/07/2007	EMPRESTADO	NORMAL	  
6	MULTIMETRO DIGITAL	ADMINISTRADOR DO SGEEM	19/06/2007	18/06/2007	EMPRESTADO		  
8	TERMOMETRO DIGITAL	DAVID PEDRO WILLEMANN	19/06/2007	18/06/2007	EMPRESTADO		  

Figura 34 - Devolução de Equipamento do SGEL

## 7 Conclusão

A abordagem de frameworks contribui significativamente para reutilização no desenvolvimento de artefatos de software. Frameworks é uma abordagem que está sendo cada vez mais utilizada, com o objetivo de diminuir o tempo e esforços no desenvolvimento de artefatos de software.

Para se ter um bom projeto, este deve ser bem estruturado e planejado. Padrões de projeto ajudam o projetista no desenvolvimento de projetos melhor estruturados, oferecendo soluções que foram desenvolvidas e aperfeiçoadas ao longo do tempo.

A definição de uma estrutura padronizada para um determinado domínio de aplicações, juntamente com o desenvolvimento uma série de componentes oferecendo funcionalidades prontas para o domínio através da aplicação de padrões de projeto, caracterizam o projeto de um framework.

O framework desenvolvido neste trabalho disponibiliza uma série de funcionalidades prontas para o desenvolvimento de aplicações Web utilizando banco de dados. Seu foco está na separação das camadas de dados, controle e visualização, através da aplicação do padrão de projeto MVC (Model View Controller), e no suporte as operações básicas de CRUD (Create Retrieve Update Delete) sobre objetos.

A utilização da arquitetura MVC oferece vantagens para desenvolvedores como a otimização das habilidades de equipes através de especialidades específicas de cada um, e a redução de custos associados ao desenvolvimento, além de favorecer a extensibilidade e reutilização do código.

Além disso, o desenvolvimento de aplicações baseadas em um framework com uma arquitetura definida, permite uma maior qualidade do código e do projeto, uma vez que a

aplicação é construída através do reuso de componentes e obedecendo uma estrutura bem definida e padronizada.

Outro ponto importante oferecido pelo framework é o suporte às operações básicas de CRUD sobre objetos, que permite um desenvolvimento rápido de funcionalidades básicas de um sistema, que na maioria dos casos somam mais de 50% do projeto.

O desenvolvimento de aplicações utilizando o framework desenvolvido através da aplicação de sua estrutura e utilização de seus componentes, produz aplicações mais robustas e com uma excelente qualidade. Além disso, o tempo de desenvolvimento torna-se muito menor, diminuindo consideravelmente os custos do projeto.

Como prova de conceito e forma de medir a produtividade e reuso dos componentes do framework, foi desenvolvido um sistema de controle de empréstimo de equipamentos para o Laboratório de Metrologia (LABMETRO) da Universidade Federal de Santa Catarina, o qual chamamos SGEL.

Para o desenvolvimento do SGEL, foram realizadas algumas reuniões com os responsáveis pelos empréstimos de equipamentos do LABMETRO com o intuito de definir os casos de uso, objetos conceituais e atores do sistema. Ao final das reuniões, ficou definido o escopo do sistema e seus cenários, totalizando dezoito casos de uso e três atores envolvidos (administrador, operador e usuário comum).

A partir dos casos de uso, iniciou-se o desenvolvimento do sistema, buscando sempre a utilização de todos os recursos oferecidos pelo framework. Os resultados obtidos foram muito animadores e justificam o esforço despendido no desenvolvimento do framework. Dos dezoito casos de uso do sistema, treze deles foram desenvolvidos sem qualquer customização das funcionalidades oferecidas pelo framework, definindo uma cobertura do framework de 72,2% da aplicação construída. Os outros cinco casos de uso do sistema necessitaram de uma pequena especialização das funcionalidades básicas do framework e adição de novas

funcionalidades específicas das regras do SGEL. Estas funcionalidades foram customizadas com a especialização de apenas uma classe do framework, cujo código resumiu-se em 111 linhas.

## 8 Trabalhos Futuros

De fato a metodologia de desenvolvimento adotada para este framework foi a evolutiva. Partimos das experiências de várias outras aplicações, criando componentes extensíveis, componentes reutilizáveis, biblioteca de funções, definindo pontos de extensão e refinando objetos.

Entretanto, é sabido que muitos pontos podem ser melhorados a fim de oferecer cada vez mais funcionalidades e ferramentas de produtividade para o desenvolvimento.

Neste primeiro trabalho focou-se na definição de uma estrutura e de componentes que oferecessem funcionalidades prontas para o domínio do framework. No entanto, não foi focado na construção de ferramentas de geração de código e automatização de procedimentos, nem na definição de todos os ganchos para especialização das funcionalidades.

Desta forma, como trabalhos futuros sugere-se:

- Definição dos pontos de mais pontos de flexão para cada camada da estrutura do framework, como por exemplo, `beforeInsert`, `afterInsert`, `beforeUpdate`, `afterUpdate`, `beforeExecuteMethod`, `afterEecuteMethod`, etc.
- Criação de ferramentas para geração visual de classes de domínio (VO's).
- Criação de ferramentas para geração visual de códigos para especialização das classes do framework.
- Criação de ferramentas para geração automática de páginas para as operações básicas de CRUD.



## 9 Referências Bibliográficas

[1] AMBLER, Scott W. Análise e projeto orientado a objeto – Seu guia para desenvolver sistemas robustos com tecnologia de objetos. Tradução Oswaldo Zanelli. Rio de Janeiro: Infobook, 1998.

[2] ARAGÃO JÚNIOR, Marício. Hibernate 3 Avançado, Boas Práticas, padrões e caching. Disponível em <http://www.guj.com.br/java/tutorial/artigo.181.1.guj>.

[3] BOOCH, Grady; RUMBAUGH, James; JACOBSON, Ivar. UML Guia do Usuário. Rio de Janeiro: Campus. 2001.

[4] CARDOSO, C. HTML Truques espertos. Rio de Janeiro: Axcel Books, 1996.

[5] CASTRO, M. A. S. O que é HTML. São Paulo, 2003. Disponível em: <http://www.icmc.usp.br/ensino/material/html/html.html>.

[6] CAVALHIERI, Marcos A. Programação Java. Disponível em: [http://cognitio.incubadora.fapesp.br/portal/atividades/cursos/instrumentacao/java/ProgramacaoJava\\_01.pdf](http://cognitio.incubadora.fapesp.br/portal/atividades/cursos/instrumentacao/java/ProgramacaoJava_01.pdf).

[7] DAMASCENO JÚNIOR, Américo. Aprendendo JAVA: programação na Internet. São Paulo. Érica. 2001

[8] DEITEL, H. M. Java. How to Program. Prentice Hall 2004.

[9] FURLAN, José Davi; Assumpção Filho, Milton Mira de, editor. Modelagem de objetos através da UML - the unified modeling language. São Paulo: Makron Books: Pearson Education, c1998.

[10] GALHARDO, Raphaela; LIMA, Gleydson. Mapeando Associações com Hibernate. Disponível em: <http://www.portaljava.com/home/modules.php?name=Content>.

- [11] GAMMA, Eric, HELM, Richard, JOHNSON, Ralph, VLISSIDES, John. Padrões de Projeto: soluções reutilizáveis de software orientado a objetos. Porto Alegre: Bookman, 2002.
- [12] H.M. DEITEL; P.J. Deitel. Java – Como programar. São Paulo. Makron Books 2000.
- [13] HTML, HyperText Markup Language. Disponível em: <http://www.w3.org/html/>.
- [14] HUSTED, Ted et al; Struts em ação. Local: Rio de Janeiro: Editora Ciência Moderna Ltda, 2004.
- [15] JOHNSON, R. E. Components, frameworks, patterns. In: ACM SIGSOFT Symposium on Software Reusability. [s.n.], 1997. p. 10–17. Disponível em: <http://citeseer.ist.psu.edu/johnson97components.html>.
- [16] JÚNIOR, Herval Freire de A . Júnior. Mapeamento Objetos em Banco de Dados Relacionais. Disponível em: <http://www.mundooo.com.br/php/modules.php?name=MOOArtigos&pa=showpage&pid=28>.
- [17] KRAEMER, Fabiano; JARDEL VOGT, Jerônimo. Hibernate, um Robusto Framework de Persistência Objeto-Relacional. Disponível em: <http://www.guj.com.br/artigos.jsp>.
- [18] LARMAN, Craig. Utilizando UML Padrões. Uma Introdução a Análise e ao Projeto Orientado a Objetos. Porto Alegre. Bookman 2000.
- [19] LINHARES, Maurício. Introdução ao Hibernate 3. Disponível em: <http://www.guj.com.br/artigos.jsp>.
- [20] LOZANO, Fernando. Desenvolvendo aplicações Web com MVC. Java Magazine, São Paulo, v. 2, n. 20, p. 34-39, fev. 2004.

- [21] LOZANO, Fernando. Persistência Objeto-Relacional com Java. Disponível em : <http://www.lozano.eti.br/>.
- [22] MARTIN, James. Princípios de análise e projeto baseados em objetos. Tradução Cristina Bazán. Rio de Janeiro: Campus, 1994.
- [23] MYSQL, AB. Manual de referência do MySQL. Disponível em: <http://dev.mysql.com/doc/mysql/pt/>.
- [24] POMPILIO, Janaina Estigarribia. Tutorial para a linguagem UML (Unified Modeling Language). Presidente Prudente: [s.n.], 1998.
- [25] PREE, W. Design patterns for object oriented software development.
- [26] ROBERTS, D.; JOHNSON, R. E. Evolving frameworks: A pattern language for developing object-oriented frameworks. In: Pattern Languages of Program Design 3.
- [27] RODRIGUES, Analise Rosa. Um componente de Software para mapeamento objeto relacional. Disponível em [www.upf.br/erbd/index.php?option=com\\_content &task=view &id=20&Itemid=1](http://www.upf.br/erbd/index.php?option=com_content&task=view&id=20&Itemid=1)
- [28] SILVA, Ricardo P. Suporte ao desenvolvimento e uso de frameworks e componentes. Tese para a obtenção do grau de Doutor em Ciências da Computação. Universidade Federal do Rio Grande do Sul, UFRGS. Porto Alegre, 2000.
- [29] SIMON, G. Web sites pessoais. Disponível em: <http://gilbertosimon.sites.uol.com.br/>.
- [30] TILES, Framework Tiles. Disponível em: <http://tiles.apache.org>.
- [31] TODD, Nick et al; Java Server Pages: Guia do Desenvolvedor. Editora: CAMPUS, 2003.
- [32] UNIVERSIDADE FEDERAL DE SANTA CATARINA. Laboratório de Metrologia e Automatização. Disponível em: <http://www.labmetro.ufsc.br>.

[33] WEBSERVICES. Informática. Disponível em: <http://www.terra.com.br/informática/ebusiness/2003/03/24/003.htm>.

[34] WESLEY Addison, 1997. Disponível em: <http://citeseer.ist.psu.edu/roberts96evolving.html>.

[35] WWW, World Wide Web. Disponível em: <http://www.w3.org/MarkUp/historical>.

# Anexo A – VO's do SGEL

## A.1 – USUÁRIO

```
package tcc.sgeem.vo;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
import javax.persistence.Table;

import tcc.sgeem.model.UsuarioBO;
import br.com.ibarra.model.annotation.BoClass;
import br.com.ibarra.vo.AppGenericVO;

@BoClass(value = UsuarioBO.class, singleton = true)
@Entity
@Table(name = "USUARIOS")
@NamedQueries( {
    @NamedQuery(name = "usuario.login", query = "from Usuario where
dsLogin=:p_dsLogin"),
    @NamedQuery(name = "usuario.nome", query = "from Usuario where
nmUsuario like '%:p_nmUsuario%' order by nmUsuario"),
    @NamedQuery(name = "usuario.todos", query = "from Usuario order
by nmUsuario")
})
public class Usuario extends AppGenericVO {

    @Column(name = "NM_USUARIO", nullable = false, length = 200)
    private String nmUsuario;

    @Column(name = "DS_EMAIL", nullable = false, length = 100)
    private String dsEmail;

    @Column(name = "DS_LOGIN", nullable = false, length = 30)
    private String dsLogin;

    @Column(name = "DS_SENHA", nullable = false, length = 30)
    private String dsSenha;

    @Column(name = "NR_TELEFONE", nullable = true, length = 15)
    private String nrTelefone;

    @Column(name = "NR_CELULAR", nullable = true, length = 15)
    private String nrCelular;

    @Column(name = "NR_NIVEL_ACESSO", nullable = false, length = 1)
    private String nrNivelAcesso;

    public String getDsEmail() {
        return dsEmail;
    }

    public void setDsEmail(String dsEmail) {
        this.dsEmail = dsEmail;
    }
}
```

```

    }

    public String getDsLogin() {
        return dsLogin;
    }

    public void setDsLogin(String dsLogin) {
        this.dsLogin = dsLogin;
    }

    public String getDsSenha() {
        return dsSenha;
    }

    public void setDsSenha(String dsSenha) {
        this.dsSenha = dsSenha;
    }

    public String getNmUsuario() {
        return nmUsuario;
    }

    public void setNmUsuario(String nmUsuario) {
        this.nmUsuario = nmUsuario;
    }

    public String getNrCelular() {
        return nrCelular;
    }

    public void setNrCelular(String nrCelular) {
        this.nrCelular = nrCelular;
    }

    public String getNrNivelAcesso() {
        return nrNivelAcesso;
    }

    public void setNrNivelAcesso(String nrNivelAcesso) {
        this.nrNivelAcesso = nrNivelAcesso;
    }

    public String getNrTelefone() {
        return nrTelefone;
    }

    public void setNrTelefone(String nrTelefone) {
        this.nrTelefone = nrTelefone;
    }
}

```

## A.1 – EQUIPAMENTO

```
package tcc.sgeem.vo;
```

```

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
import javax.persistence.Table;

import br.com.ibarra.model.AppGenericBO;
import br.com.ibarra.model.annotation.BoClass;
import br.com.ibarra.vo.AppGenericVO;

@BoClass(value = AppGenericBO.class, singleton = true)
@Entity
@Table(name = "EQUIPAMENTOS")
@NamedQueries( {
    @NamedQuery(name = "equipamento.nome", query = "from
Equipamento where nmEquipamento like :p_nmEquipamento order by
nmEquipamento"),
    @NamedQuery(name = "equipamento.todos", query = "from
Equipamento order by nmEquipamento")
})
public class Equipamento extends AppGenericVO {

    @Column(name = "DS_CODIGO", nullable = false, length = 30)
    private String dsCodigo;

    @Column(name = "NM_EQUIPAMENTO", nullable = false, length = 200)
    private String nmEquipamento;

    @Column(name = "NM_FABRICANTE", nullable = false, length = 200)
    private String nmFabricante;

    @Column(name = "DS_MODELO", nullable = false, length = 200)
    private String dsModelo;

    @Column(name = "NR_SERIE", nullable = false, length = 30)
    private Long nrSerie;

    @Column(name = "NR_PATRIMONIO", nullable = false, length = 30)
    private Long nrPatrimonio;

    @Column(name = "DS_LOCAL", nullable = true, length = 100)
    private String dsLocal;

    @Column(name = "DS_OBS", nullable = true, length = 2000)
    private String dsObservacoes;

    @Column(name = "DS_LOCAL_MANUAL", nullable = true, length = 500)
    private String dsLocalManual;

    public String getDsCodigo() {
        return dsCodigo;
    }

    public void setDsCodigo(String dsCodigo) {
        this.dsCodigo = dsCodigo;
    }

    public String getDsModelo() {
        return dsModelo;
    }
}

```

```

public void setDsModelo(String dsModelo) {
    this.dsModelo = dsModelo;
}

public String getDsLocal() {
    return dsLocal;
}

public void setDsLocal(String dsLocal) {
    this.dsLocal = dsLocal;
}

public String getDsLocalManual() {
    return dsLocalManual;
}

public void setDsLocalManual(String dsLocalManual) {
    this.dsLocalManual = dsLocalManual;
}

public String getDsObservacoes() {
    return dsObservacoes;
}

public void setDsObservacoes(String dsObservacoes) {
    this.dsObservacoes = dsObservacoes;
}

public String getNmEquipamento() {
    return nmEquipamento;
}

public void setNmEquipamento(String nmEquipamento) {
    this.nmEquipamento = nmEquipamento;
}

public String getNmFabricante() {
    return nmFabricante;
}

public void setNmFabricante(String nmFabricante) {
    this.nmFabricante = nmFabricante;
}

public Long getNrPatrimonio() {
    return nrPatrimonio;
}

public void setNrPatrimonio(Long nrPatrimonio) {
    this.nrPatrimonio = nrPatrimonio;
}

public Long getNrSerie() {
    return nrSerie;
}

public void setNrSerie(Long nrSerie) {
    this.nrSerie = nrSerie;
}

```



```
}
```

## A.1 – EMPRESTIMO

```
package tcc.sgeem.vo;

import java.util.Date;

import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
import javax.persistence.Table;
import javax.persistence.Transient;

import tcc.sgeem.model.EmprestimoBO;
import br.com.ibarra.model.annotation.BoClass;
import br.com.ibarra.util.DateFormatUtil;
import br.com.ibarra.vo.AppGenericVO;

@BoClass(value = EmprestimoBO.class, singleton = true)
@Entity
@Table(name = "EMPRESTIMOS")
@NamedQueries( { @NamedQuery(name = "emprestimo.emprestados", query = "from
Emprestimo where cdStatus='EMPRESTADO' order by dtEmprestimo"),
    @NamedQuery(name = "emprestimo.usuario", query = "from
Emprestimo where cdStatus='EMPRESTADO' and usuario.dsLogin =
:p_usuario_login order by dtEmprestimo"),
    @NamedQuery(name = "emprestimo.atrasados", query = "from
Emprestimo where cdStatus='EMPRESTADO' and TO_DAYS(dtDevolucao) <
TO_DAYS(NOW()) order by dtEmprestimo"),
    @NamedQuery(name = "emprestimo.reservados", query = "from
Emprestimo where cdStatus='RESERVADO' order by id "),
    @NamedQuery(name = "emprestimo.reservas.equipamento", query =
"from Emprestimo where equipamento.id = :p_equipamento_id and
(cdStatus='RESERVADO' or cdStatus='EMPRESTADO)'),
    @NamedQuery(name = "emprestimo.equipamento", query = "from
Emprestimo where equipamento.id = :p_equipamento_id and
cdStatus='EMPRESTADO'"),
    @NamedQuery(name = "reservas.equipamento", query = "from
Emprestimo where equipamento.id = :p_equipamento_id and
cdStatus='RESERVADO'") })
public class Emprestimo extends AppGenericVO {

    @ManyToOne(cascade = { CascadeType.PERSIST })
    @JoinColumn(name = "USUARIO_ID", nullable = false)
    private Usuario usuario = new Usuario();

    @ManyToOne(cascade = { CascadeType.PERSIST })
    @JoinColumn(name = "EQUIPAMENTO_ID", nullable = false)
    private Equipamento equipamento = new Equipamento();

    @Column(name = "DT_SOLICITACAO", nullable = true)
    private Date dtSolicitacao = DateFormatUtil.getDataAtual();
```

```

@Transient
private String dtSolicitacaoStr;

@Column(name = "DT_EMPRESTIMO", nullable = true)
private Date dtEmprestimo;

@Transient
private String dtEmprestimoStr;

@Column(name = "DT_DEVOLUCAO", nullable = true)
private Date dtDevolucao;

@Transient
private String dtDevolucaoStr;

@Column(name = "DS_OBSERVACOES", length = 500)
private String dsObservacoes;

@Column(name = "CD_STATUS", length = 30)
private String cdStatus;

public String getCdStatus() {
    return cdStatus;
}

public void setCdStatus(String cdStatus) {
    this.cdStatus = cdStatus;
}

public String getDsObservacoes() {
    return dsObservacoes;
}

public void setDsObservacoes(String dsObservacoes) {
    this.dsObservacoes = dsObservacoes;
}

public Date getDtDevolucao() {
    return dtDevolucao;
}

public void setDtDevolucao(Date dtDevolucao) {
    this.dtDevolucao = dtDevolucao;
}

public String getDtDevolucaoStr() {
    return DateFormatUtil.formatDataStr(this.dtDevolucao);
}

public void setDtDevolucaoStr(String dtDevolucaoStr) {
    this.dtDevolucaoStr = dtDevolucaoStr;
    this.dtDevolucao = DateFormatUtil.parseDataStr(dtDevolucaoStr);
}

public Date getDtEmprestimo() {
    return dtEmprestimo;
}

public void setDtEmprestimo(Date dtEmprestimo) {
    this.dtEmprestimo = dtEmprestimo;
}

```

```

    }

    public String getDtEmprestimoStr() {
        return DateFormatUtil.formatDataStr(this.dtEmprestimo);
    }

    public void setDtEmprestimoStr(String dtEmprestimoStr) {
        this.dtEmprestimoStr = dtEmprestimoStr;
        this.dtEmprestimo =
DateFormatUtil.parseDataStr(dtEmprestimoStr);
    }

    public Date getDtSolicitacao() {
        return dtSolicitacao;
    }

    public void setDtSolicitacao(Date dtSolicitacao) {
        this.dtSolicitacao = dtSolicitacao;
    }

    public String getDtSolicitacaoStr() {
        return DateFormatUtil.formatDataStr(this.dtSolicitacao);
    }

    public void setDtSolicitacaoStr(String dtSolicitacaoStr) {
        this.dtSolicitacaoStr = dtSolicitacaoStr;
        this.dtSolicitacao =
DateFormatUtil.parseDataStr(dtSolicitacaoStr);
    }

    public Equipamento getEquipamento() {
        return equipamento;
    }

    public void setEquipamento(Equipamento equipamento) {
        this.equipamento = equipamento;
    }

    public Usuario getUsuario() {
        return usuario;
    }

    public void setUsuario(Usuario usuario) {
        this.usuario = usuario;
    }

    public boolean isAtrasado() {
        if (this.dtDevolucao != null) {
            return
DateFormatUtil.getDataAtual().after(this.dtDevolucao);
        } else {
            return false;
        }
    }
}

```

# ANEXO B – Customizações para o SCEL

## B.1 - EMPRESTAR EQUIPAMENTO

Problema: Antes de se registrar um empréstimo é necessário verificar se o equipamento já não está emprestado para outro usuário, e caso estiver deve-se registrar uma reserva para o equipamento.

Solução: Sobrescrita do método *insert* da classe *EmprestimoBO*.

Código:

```
@Override
public AppGenericVO insert(AppGenericVO vo) throws BOException {

    try {
        Emprestimo emprestimo = (Emprestimo) vo;

        Map<String, Object> param = new HashMap<String, Object>();
        param.put("p equipamento id",
emprestimo.getEquipamento().getId());
        List<Emprestimo> reservas =
            this.getDao().executeNamedQuery("emprestimo.reservas.equipamento",
param);

        if (reservas != null && reservas.size()>0) {
            emprestimo.setCdStatus("RESERVADO");
        } else {
            emprestimo.setCdStatus("EMPRESTADO");
        }
        return super.insert(vo);

    } catch (DAOException e) {
        throw new BOException(e);
    }
}
```

## B.2 - DEVOLVER EQUIPAMENTO

Problema: Quando um equipamento for devolvido deve-se verificar se existem reservas para o equipamento que está sendo devolvido. Caso existam, os usuários que reservaram o equipamento deverão ser notificados através do envio de um e-mail.

Solução: criação do método *devolver* na classe *EmprestimoBO*, e configuração do parâmetro método no action correspondente ao caso de uso.

Código:

```
public AppGenericVO devolver(AppGenericVO vo) throws BOException {
    try {
        Emprestimo emprestimo = (Emprestimo) vo;
        if ("EMPRESTADO".equals(emprestimo.getCdStatus())) {
            emprestimo.setCdStatus("DEVOLVIDO");
            emprestimo = (Emprestimo) this.update(emprestimo);
            Map<String, Object> param = new HashMap<String,
Object>();
            param.put("p equipamento_id",
emprestimo.getEquipamento().getId());
            List<Emprestimo> reservas =
                this.getDao().executeNamedQuery("reservas.equipamento", param);
            for (Emprestimo e : reservas) {
                String mensagem = getMensagemLiberacao(e);
                this.sendNotification(e.getUsuario(), mensagem);
            }
        }
        return emprestimo;
    } catch (DAOException e) {
        throw new BOException(e);
    }
}
```

## B.3 – EXCLUSÃO DE EMPRÉSTIMO

Problema: Quando um empréstimo for excluído deve-se verificar se existem reservas para o equipamento objeto do empréstimo. Caso existam, os usuários que reservaram o equipamento deverão ser notificados através do envio de um e-mail.

Solução: sobrescrita do método *delete* da classe *EmprestimoBO*.

Código:

```
@Override
public void delete(AppGenericVO vo) throws BOException {
    try {
        Emprestimo emprestimo = (Emprestimo) vo;
        if ("EMPRESTADO".equals(emprestimo.getCdStatus())) {
```

```

        Map<String, Object> param = new HashMap<String,
Object>();
        param.put("p_equipamento_id",
emprestimo.getEquipamento().getId());
        List<Emprestimo> reservas =
        this.getDao().executeNamedQuery("reservas.equipamento", param);
        for (Emprestimo e : reservas) {
            String mensagem = getMensagemLiberacao(e);
            this.sendNotification(e.getUsuario(), mensagem);
        }
        super.delete(vo);
    } catch (DAOException e) {
        throw new BOException(e);
    }
}

```

## B.4 – NOTIFICAR ATRASO NA DEVOLUÇÃO

Problema: quando um empréstimo está com sua data de devolução ultrapassada, o sistema deve permitir que o usuário envie um e-mail ao usuário que está com o equipamento, informando a situação de atraso.

Solução: criação do método *notificarAtraso* na classe *EmprestimoBO*, e configuração do parâmetro método no action correspondente ao caso de uso.

Código:

```

public AppGenericVO notificarAtraso(AppGenericVO vo) throws BOException {
    Emprestimo emprestimo = (Emprestimo) vo;
    if ("EMPRESTADO".equals(emprestimo.getCdStatus())) {
        String mensagem = getMensagemLiberacao(emprestimo);
        this.sendNotification(emprestimo.getUsuario(), mensagem);
    }
    return emprestimo;
}

```

## B.5 – EFETIVAR EMPRÉSTIMO DE RESERVA

Problema: Quando um equipamento for devolvido e houver empréstimos para o mesmo, o sistema deve permitir que o operador efetive uma reserva, mudando o status de RESERVADO para EMPRESTADO.

Solução: criação do método *efetivarEmprestimo* na classe *EmprestimoBO*, e configuração do parâmetro método no action correspondente ao caso de uso.

Código:

```
public AppGenericVO efetivarEmprestimo(AppGenericVO vo) throws BOException
{
    try {
        Emprestimo emprestimo = (Emprestimo) vo;
        if (!"EMPRESTADO".equals(emprestimo.getCdStatus())) {

            Map<String, Object> param = new HashMap<String,
Object>();
            param.put("p_equipamento_id",
emprestimo.getEquipamento().getId());
            List<Emprestimo> emprestimos =
                this.getDao().executeNamedQuery("emprestimo.equipamento", param);
            if (emprestimos == null || emprestimos.size() == 0) {
                emprestimo.setCdStatus("EMPRESTADO");
                emprestimo = (Emprestimo) this.update(emprestimo);
            } else {
                throw new BOException("O recurso já está emprestado
para o usuário " + emprestimos.get(0).getUsuario().getNmUsuario());
            }
            return emprestimo;
        } catch (DAOException e) {
            throw new BOException(e);
        }
    }
}
```

# ANEXO C - CONFIGURAÇÕES

## C.1 – ARQUIVO DE CONFIGURAÇÃO STRUTS-CONFIG.XML

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<!DOCTYPE struts-config PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 1.3//EN"
    "http://struts.apache.org/dtds/struts-config_1_3.dtd">

<struts-config>

    <form-beans>
        <form-bean name="genericForm"
            type="org.apache.struts.validator.DynaValidatorForm" />

        <form-bean name="usuarioForm"
            type="org.apache.struts.validator.DynaValidatorForm">
            <form-property
                name="PROPERTY_OBJECT_INTROSPECTION"
                initial="tcc.sgeem.vo.Usuario"
                type="java.lang.String" />
        </form-bean>
        <form-bean name="usuarioCadForm"
            extends="usuarioForm"
            type="org.apache.struts.validator.DynaValidatorForm">
            <form-property name="nextAction"
                initial="/incluirusuario.do"
                type="java.lang.String" />
            <form-property name="dsSenha2"
                type="java.lang.String" />
        </form-bean>

        <form-bean name="equipamentoForm"
            type="org.apache.struts.validator.DynaValidatorForm">
            <form-property
                name="PROPERTY_OBJECT_INTROSPECTION"
                initial="tcc.sgeem.vo.Equipamento"
                type="java.lang.String" />
            <form-property name="p_nmEquipamento"
                initial="" type="java.lang.String" />
        </form-bean>
        <form-bean name="equipamentoCadForm"
            extends="equipamentoForm"
            type="org.apache.struts.validator.DynaValidatorForm">
            <form-property name="nextAction"
                initial="/incluir Equipamento.do"
                type="java.lang.String" />
        </form-bean>
        <form-bean name="equipamentoConForm"
            extends="equipamentoForm"
            type="org.apache.struts.validator.DynaValidatorForm">
```



```

        <form-property name="p_nmEquipamento"
            initial="" type="java.lang.String" />
    </form-bean>

    <form-bean name="emprestimoForm"
        type="org.apache.struts.validator.DynaValidatorForm">
        <form-property
            name="PROPERTY_OBJECT_INTROSPECTION"
            initial="tcc.sgeem.vo.Emprestimo"
            type="java.lang.String" />
    </form-bean>
    <form-bean name="emprestimoCadForm"
        extends="emprestimoForm"
        type="org.apache.struts.validator.DynaValidatorForm">
        <form-property name="nextAction"
            initial="/incluiremprestimo.do"
            type="java.lang.String" />
        <form-property name="equipamento_id"
            type="java.lang.Long" />
        <form-property name="usuario_id"
            type="java.lang.Long" />
    </form-bean>
    <form-bean
        name="emprestimoEquipamentoConForm"
        extends="emprestimoForm"
        type="org.apache.struts.validator.DynaValidatorForm">
        <form-property name="p_nmEquipamento"
            initial="" type="java.lang.String" />
        <form-property name="p_usuario_login"
            initial="" type="java.lang.String" />
    </form-bean>

</form-beans>

<global-exceptions />
<global-forwards />

<action-mappings>

    <action path="/menu" name="genericForm"
        validate="false"
        className="br.com.ibarra.web.struts.AppActionMapping"
        type="tcc.sgeem.web.struts.MenuAction">
        <forward name="menu0"
            path="/menu0.jsp" />
        <forward name="menu1"
            path="/menu1.jsp" />
        <forward name="menu2"
            path="/menu2.jsp" />
    </action>

    <action path="/logout" name="genericForm"
        validate="false"
        className="br.com.ibarra.web.struts.AppActionMapping"
        type="br.com.ibarra.web.struts.LogoutAction">
        <forward name="sucesso"
            path="mainLayout" />
    </action>

    <action path="/usuarios"
        name="usuarioForm" validate="false"

```

```

        className="br.com.ibarra.web.struts.AppActionMapping"
        type="br.com.ibarra.web.struts.actions.AppGenericAction">
        <set-property property="vo"
            value="tcc.sgeem.vo.Usuario" />
        <set-property
            property="executeNamedQuery"
            value="usuario.todos#listaUsuarios" />
        <forward name="sucesso"
            path="usuario.consulta" />
    </action>
    <action path="/novousuario"
        className="br.com.ibarra.web.struts.AppActionMapping"
        name="usuarioCadForm" scope="request"
        validate="false"
        type="br.com.ibarra.web.struts.actions.AppGenericAction">
        <set-property property="vo"
            value="tcc.sgeem.vo.Usuario" />
        <set-property property="limpaForm"
            value="true" />
        <set-property property="nextAction"
            value="/incluirusuario.do" />
        <forward name="sucesso"
            path="usuario.cadastro" />
    </action>
    <action path="/incluirusuario"
        input="usuario.cadastro"
        className="br.com.ibarra.web.struts.AppActionMapping"
        name="usuarioCadForm" scope="request"
        validate="true"
        type="br.com.ibarra.web.struts.actions.AppGenericAction">
        <set-property property="vo"
            value="tcc.sgeem.vo.Usuario" />
        <set-property property="operacao"
            value="insert" />
        <set-property property="nextAction"
            value="/alterarusuario.do" />
        <forward name="sucesso"
            path="usuario.cadastro" />
    </action>
    <action path="/alterarusuario"
        input="usuario.cadastro"
        className="br.com.ibarra.web.struts.AppActionMapping"
        name="usuarioCadForm" scope="request"
        validate="true"
        type="br.com.ibarra.web.struts.actions.AppGenericAction">
        <set-property property="vo"
            value="tcc.sgeem.vo.Usuario" />
        <set-property property="operacao"
            value="update" />
        <set-property property="nextAction"
            value="/alterarusuario.do" />
        <forward name="sucesso"
            path="usuario.cadastro" />
    </action>
    <action path="/carregarusuario"
        className="br.com.ibarra.web.struts.AppActionMapping"
        name="usuarioCadForm" scope="request"
        validate="false"
        type="br.com.ibarra.web.struts.actions.AppGenericAction">
        <set-property property="vo"
            value="tcc.sgeem.vo.Usuario" />

```

```

        <set-property property="operacao"
            value="load" />
        <set-property property="nextAction"
            value="/alterarusuario.do" />
        <forward name="sucesso"
            path="usuario.cadastro" />
    </action>
    <action path="/excluirusuario"
        className="br.com.ibarra.web.struts.AppActionMapping"
        name="usuarioForm" scope="request"
        validate="false"
        type="br.com.ibarra.web.struts.actions.AppGenericAction">
        <set-property property="vo"
            value="tcc.sgeem.vo.Usuario" />
        <set-property property="operacao"
            value="delete" />
        <forward name="sucesso"
            path="/usuarios.do" />
    </action>

    <action path="/carregarmeusdados"
        className="br.com.ibarra.web.struts.AppActionMapping"
        name="usuarioCadForm" scope="request"
        validate="false"
        type="tcc.sgeem.web.struts.UsuarioLoadAction">
        <set-property property="vo"
            value="tcc.sgeem.vo.Usuario" />
        <set-property property="nextAction"
            value="/alterarmeusdados.do" />
        <forward name="sucesso"
            path="usuario.cadastro" />
    </action>
    <action path="/alterarmeusdados"
        input="usuario.cadastro"
        className="br.com.ibarra.web.struts.AppActionMapping"
        name="usuarioCadForm" scope="request"
        validate="true"
        type="br.com.ibarra.web.struts.actions.AppGenericAction">
        <set-property property="vo"
            value="tcc.sgeem.vo.Usuario" />
        <set-property property="operacao"
            value="update" />
        <set-property property="nextAction"
            value="/alterarmeusdados.do" />
        <forward name="sucesso"
            path="usuario.cadastro" />
    </action>

    <action input="equipamento.consulta"
        path="/equipamentos" name="equipamentoConForm"
        validate="false"
        className="br.com.ibarra.web.struts.AppActionMapping"
        scope="request"
        type="br.com.ibarra.web.struts.actions.AppGenericAction">
        <set-property property="vo"
            value="tcc.sgeem.vo.Equipamento" />
        <set-property
            property="executeNamedQuery"
            value="equipamento.nome#listaEquipamentos#p_nmEquipamento" />
        <forward name="sucesso"

```

```

        path="equipamento.consulta" />
</action>
<action path="/novoequipamento"
        className="br.com.ibarra.web.struts.AppActionMapping"
        name="equipamentoCadForm" scope="request"
        validate="false"
        type="br.com.ibarra.web.struts.actions.AppGenericAction">
    <set-property property="vo"
        value="tcc.sgeem.vo.Equipamento" />
    <set-property property="limpaForm"
        value="true" />
    <set-property property="nextAction"
        value="/incluir Equipamento.do" />
    <forward name="sucesso"
        path="equipamento.cadastro" />
</action>
<action path="/incluir Equipamento"
        input="equipamento.cadastro"
        className="br.com.ibarra.web.struts.AppActionMapping"
        name="equipamentoCadForm" scope="request"
        validate="true"
        type="br.com.ibarra.web.struts.actions.AppGenericAction">
    <set-property property="vo"
        value="tcc.sgeem.vo.Equipamento" />
    <set-property property="operacao"
        value="insert" />
    <set-property property="nextAction"
        value="/alterar Equipamento.do" />
    <forward name="sucesso"
        path="equipamento.cadastro" />
</action>
<action path="/alterar Equipamento"
        input="equipamento.cadastro"
        className="br.com.ibarra.web.struts.AppActionMapping"
        name="equipamentoCadForm" scope="request"
        validate="true"
        type="br.com.ibarra.web.struts.actions.AppGenericAction">
    <set-property property="vo"
        value="tcc.sgeem.vo.Equipamento" />
    <set-property property="operacao"
        value="update" />
    <set-property property="nextAction"
        value="/alterar Equipamento.do" />
    <forward name="sucesso"
        path="equipamento.cadastro" />
</action>
<action path="/carregar Equipamento"
        className="br.com.ibarra.web.struts.AppActionMapping"
        name="equipamentoCadForm" scope="request"
        validate="false"
        type="br.com.ibarra.web.struts.actions.AppGenericAction">
    <set-property property="vo"
        value="tcc.sgeem.vo.Equipamento" />
    <set-property property="operacao"
        value="load" />
    <set-property property="nextAction"
        value="/alterar Equipamento.do" />
    <forward name="sucesso"
        path="equipamento.cadastro" />
</action>
<action path="/excluir Equipamento"

```

```

        className="br.com.ibarra.web.struts.AppActionMapping"
        name="equipamentoForm" scope="request"
        validate="false"
        type="br.com.ibarra.web.struts.actions.AppGenericAction">
        <set-property property="vo"
            value="tcc.sgeem.vo.Equipamento" />
        <set-property property="operacao"
            value="delete" />
        <forward name="sucesso"
            path="/equipamentos.do" />
    </action>

    <action path="/emprestimo"
        name="emprestimoEquipamentoConForm"
        validate="false"
        className="br.com.ibarra.web.struts.AppActionMapping"
        scope="request"
        type="br.com.ibarra.web.struts.actions.AppGenericAction">
        <set-property property="vo"
            value="tcc.sgeem.vo.Emprestimo" />
        <set-property
            property="executeNamedQuery"
            value="equipamento.nome#listaEquipamentos#p_nmEquipamento" />
        <forward name="sucesso"
            path="emprestimo.equipamento.consulta" />
    </action>
    <action path="/listaemprestimos"
        name="emprestimoForm" validate="false"
        className="br.com.ibarra.web.struts.AppActionMapping"
        type="br.com.ibarra.web.struts.actions.AppGenericAction">
        <set-property property="vo"
            value="tcc.sgeem.vo.Emprestimo" />
        <set-property
            property="executeNamedQuery"
            value="emprestimo.emprestados#listaEmprestimos" />
        <forward name="sucesso"
            path="emprestimo.consulta" />
    </action>
    <action path="/meusemprestimos"
        name="emprestimoEquipamentoConForm"
        validate="false"
        className="br.com.ibarra.web.struts.AppActionMapping"
        type="tcc.sgeem.web.struts.MeusEmprestimosAction">
        <set-property property="vo"
            value="tcc.sgeem.vo.Emprestimo" />
        <set-property
            property="executeNamedQuery"
            value="emprestimo.usuario#listaEmprestimos#p_usuario_login" />
        <forward name="sucesso"
            path="emprestimo.meus.consulta" />
    </action>
    <action
        path="/listaemprestimosparadevolucao"
        name="emprestimoForm" validate="false"
        className="br.com.ibarra.web.struts.AppActionMapping"
        type="br.com.ibarra.web.struts.actions.AppGenericAction">
        <set-property property="vo"
            value="tcc.sgeem.vo.Emprestimo" />
        <set-property

```

```

        property="executeNamedQuery"
        value="emprestimo.emprestados#listaEmprestimos" />
    <forward name="sucesso"
        path="emprestimo.consulta.devolucao" />
</action>

<action path="/listareservas"
    name="emprestimoForm" validate="false"
    className="br.com.ibarra.web.struts.AppActionMapping"
    type="br.com.ibarra.web.struts.actions.AppGenericAction">
    <set-property property="vo"
        value="tcc.sgeem.vo.Emprestimo" />
    <set-property
        property="executeNamedQuery"
        value="emprestimo.reservados#listaEmprestimos" />
    <forward name="sucesso"
        path="emprestimo.consulta" />
</action>
<action path="/listaatrasados"
    name="emprestimoForm" validate="false"
    className="br.com.ibarra.web.struts.AppActionMapping"
    type="br.com.ibarra.web.struts.actions.AppGenericAction">
    <set-property property="vo"
        value="tcc.sgeem.vo.Emprestimo" />
    <set-property
        property="executeNamedQuery"
        value="emprestimo.atrasados#listaEmprestimos" />
    <forward name="sucesso"
        path="emprestimo.consulta" />
</action>
<action path="/novoemprestimo"
    className="br.com.ibarra.web.struts.AppActionMapping"
    name="emprestimoCadForm" scope="request"
    validate="false"
    type="tcc.sgeem.web.struts.EmprestimoNewAction">
    <set-property property="vo"
        value="tcc.sgeem.vo.Emprestimo" />
    <set-property property="limpaForm"
        value="true" />
    <set-property property="nextAction"
        value="/incluiremprestimo.do" />
    <forward name="sucesso"
        path="emprestimo.cadastro" />
</action>
<action path="/incluiremprestimo"
    input="emprestimo.cadastro"
    className="br.com.ibarra.web.struts.AppActionMapping"
    name="emprestimoCadForm" scope="request"
    validate="true"
    type="br.com.ibarra.web.struts.actions.AppGenericAction">
    <set-property property="vo"
        value="tcc.sgeem.vo.Emprestimo" />
    <set-property property="operacao"
        value="insert" />
    <forward name="sucesso"
        path="emprestimo.cadastro.resposta" />
</action>
<action path="/excluiremprestimo"
    className="br.com.ibarra.web.struts.AppActionMapping"
    name="emprestimoCadForm" scope="request"
    validate="false"

```

```

        type="br.com.ibarra.web.struts.actions.AppGenericAction">
        <set-property property="vo"
            value="tcc.sgeem.vo.Emprestimo" />
        <set-property property="operacao"
            value="delete" />
        <forward name="sucesso"
            path="/listaemprestimos.do" />
    </action>
    <action path="/devolveremprestimo"
        input="emprestimo.cadastro"
        className="br.com.ibarra.web.struts.AppActionMapping"
        name="emprestimoCadForm" scope="request"
        validate="false"
        type="br.com.ibarra.web.struts.actions.AppGenericAction">
        <set-property property="vo"
            value="tcc.sgeem.vo.Emprestimo" />
        <set-property property="operacao"
            value="load" />
        <set-property property="metodo"
            value="devolver" />
        <forward name="sucesso"
            path="/listaemprestimos.do" />
    </action>
    <action path="/notificaratrasoemprestimo"
        input="emprestimo.cadastro"
        className="br.com.ibarra.web.struts.AppActionMapping"
        name="emprestimoCadForm" scope="request"
        validate="false"
        type="br.com.ibarra.web.struts.actions.AppGenericAction">
        <set-property property="vo"
            value="tcc.sgeem.vo.Emprestimo" />
        <set-property property="operacao"
            value="load" />
        <set-property property="metodo"
            value="notificarAtraso" />
        <forward name="sucesso"
            path="/listaemprestimos.do" />
    </action>
    <action path="/efetivaremprestimo"
        input="emprestimo.cadastro"
        className="br.com.ibarra.web.struts.AppActionMapping"
        name="emprestimoCadForm" scope="request"
        validate="false"
        type="br.com.ibarra.web.struts.actions.AppGenericAction">
        <set-property property="vo"
            value="tcc.sgeem.vo.Emprestimo" />
        <set-property property="operacao"
            value="load" />
        <set-property property="metodo"
            value="efetivarEmprestimo" />
        <forward name="sucesso"
            path="/listaemprestimos.do" />
    </action>
</action-mappings>

<controller
    processorClass="org.apache.struts.tiles.TilesRequestProcessor"
    bufferSize="4096" />

<message-resources
    parameter="ApplicationResources" />

```

```
<plug-in
  className="org.apache.struts.validator.ValidatorPlugIn">
  <set-property property="pathnames"
    value="/org/apache/struts/validator/validator-
rules.xml,/WEB-INF/my-validator-rules.xml,/WEB-INF/validation.xml" />
</plug-in>

<plug-in
  className="org.apache.struts.tiles.TilesPlugin">
  <set-property
    property="definitions-config"
    value="/WEB-INF/tiles-defs.xml" />
  <set-property
    property="definitions-parser-validate"
    value="false" />
  <set-property property="moduleAware"
    value="true" />
</plug-in>
</struts-config>
```



# ANEXO D - CÓDIGOS

## D.1 – CÓDIGO DA CLASSE

### APPBASEACTION.JAVA

```
package br.com.ibarra.web.struts;

import java.util.HashMap;
import java.util.Iterator;
import java.util.List;
import java.util.Map;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.commons.beanutils.BeanUtils;
import org.apache.commons.beanutils.DynaProperty;
import org.apache.struts.action.Action;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.action.DynaActionForm;

import br.com.ibarra.model.AppGenericBO;
import br.com.ibarra.model.IAppBO;
import br.com.ibarra.util.AppFactory;
import br.com.ibarra.util.DateFormatUtil;
import br.com.ibarra.vo.AppGenericVO;
import br.com.ibarra.vo.User;
import br.com.ibarra.web.util.RequestUtil;

public abstract class AppBaseAction extends Action implements IAppAction {

    @Override
    public ActionForward execute(ActionMapping mapping, ActionForm form,
        HttpServletRequest request, HttpServletResponse response) throws Exception
    {
        inicializaDadosNaSession(request);
        limpaForm(mapping, form);
        setNextAction((AppActionMapping) mapping, form, request);
        prepareToExecute(mapping, form, request, response);
        return execute((AppActionMapping) mapping, (DynaActionForm)
            form, request, response);
    }

    public void prepareToExecute(ActionMapping mapping, ActionForm form,
        HttpServletRequest request, HttpServletResponse response) throws Exception
    {
    }
}
```

```

    public ActionForward execute(AppActionMapping mapping, DynaActionForm
form, HttpServletRequest request, HttpServletResponse response) throws
Exception {
        recuperaListasDinamicas(mapping, request);
        executaNamedQuery(mapping, form, request);
        AppGenericVO vo = executaOperacao(mapping, form, request,
response);
        vo = executaMetodo(mapping, form, request, response, vo);
        if (permittedContent(vo, mapping, request)) {
            return execute(mapping, form, request, response, vo);
        } else {
            return mapping.findForward(StrutsConfig.FOWARD.ERRO);
        }
    }

    private void setNextAction(AppActionMapping mapping, ActionForm f,
HttpServletRequest request) {
        if (mapping.getNextAction() != null &&
mapping.getNextAction().trim().length() > 0) {
            request.setAttribute("nextAction",
mapping.getNextAction());
            DynaActionForm form = (DynaActionForm) f;
            form.set("nextAction", mapping.getNextAction());
        }
    }

    private void recuperaListasDinamicas(AppActionMapping mapping,
HttpServletRequest request) throws Exception {
        for (AppGenericVO listaVO : mapping.getListaDinamica()) {
            List lista =
AppFactory.getInstance().getBO(User.class).query(listaVO.getClass());
            String nomeLista = "lista" +
listaVO.getClass().getSimpleName();
            request.setAttribute(nomeLista, lista);
        }
    }

    private void executaNamedQuery(AppActionMapping mapping,
DynaActionForm form, HttpServletRequest request) throws Exception {
        if (mapping.getNamedQuery() != null &&
mapping.getNamedQuery().trim().length() > 0) {
            List lista = null;
            if (mapping.getParametrosQuery() == null) {
                lista =
AppFactory.getInstance().getDAO(AppGenericBO.class).executeNamedQuery(mappi
ng.getNamedQuery());
            } else {
                Map<String, Object> parametros = new
HashMap<String, Object>();
                for (String p : mapping.getParametrosQuery()) {
                    parametros.put(p, form.get(p));
                    form.set(p, form.get(p));
                }
                lista =
AppFactory.getInstance().getDAO(AppGenericBO.class).executeNamedQuery(mappi
ng.getNamedQuery(), parametros);
            }

            request.setAttribute(mapping.getRequestAttributeNamedQuery(), lista);

```

```

    }
}

    private AppGenericVO executaOperacao(AppActionMapping mapping,
DynaActionForm f, HttpServletRequest request, HttpServletResponse response)
throws Exception {
    AppGenericVO vo = mapping.getVOInstance();
    if (vo != null) {
        if (mapping.getOperacao().equalsIgnoreCase("load")) {
            vo = carregaObjeto(mapping, f, request, response);
        } else if
(mapping.getOperacao().equalsIgnoreCase("update")) {
            vo = executeUpdate(mapping, f, request, response);
        } else if
(mapping.getOperacao().equalsIgnoreCase("insert")) {
            vo = executeInsert(mapping, f, request, response);
        } else if
(mapping.getOperacao().equalsIgnoreCase("delete")) {
            executeDelete(mapping, f, request, response);
        } else {
            BeanUtils.copyProperties(vo,
RequestUtil.getMapParametros(request));
        }
    }
    return vo;
}

    private AppGenericVO executaMetodo(AppActionMapping mapping,
DynaActionForm f, HttpServletRequest request, HttpServletResponse response,
AppGenericVO vo) throws Exception {
    if (!"".equals(mapping.getMetodo())) {
        vo = executeMethod(mapping, f, request, response, vo);
    }
    return vo;
}

    private void limpaForm(ActionMapping m, ActionForm f) throws
Exception {
    AppActionMapping mapping = (AppActionMapping) m;
    DynaActionForm form = (DynaActionForm) f;
    if (mapping.getLimpaForm()) {
        Iterator it = form.getMap().keySet().iterator();
        while (it.hasNext()) {
            String key = (String) it.next();
            form.set(key, null);
        }
    }
}

    private AppGenericVO carregaObjeto(AppActionMapping mapping,
DynaActionForm form, HttpServletRequest request, HttpServletResponse
response) throws Exception {
    AppGenericVO vo = mapping.getVOInstance();
    vo =
AppFactory.getInstance().getBO(vo.getClass()).load(vo.getClass(), (Long)
form.get("id"));
    request.setAttribute("vo", vo);
    valueObjectToForm(vo, form);
    return vo;
}
}

```

```

protected void valueObjectToForm(AppGenericVO vo, DynaActionForm
form) {
    try {
        Map mapValores = BeanUtils.describe(vo);

        Iterator it = mapValores.keySet().iterator();
        while (it.hasNext()) {
            try {
                String key = (String) it.next();
                DynaProperty prop =
form.getDynaClass().getDynaProperty(key);
                Class type = prop.getType();
                String valor = (String) mapValores.get(key);
                if (type == String.class) {
                    form.set(key, valor);
                } else if (type == Long.class) {
                    form.set(key, Long.valueOf(valor));
                }
            } catch (Exception e) {
                System.out.println(e.getMessage());
            }
        }

    } catch (Exception e) {
        e.printStackTrace();
    }
}

protected AppGenericVO executeUpdate(AppActionMapping mapping,
DynaActionForm f, HttpServletRequest request, HttpServletResponse response)
throws Exception {
    AppGenericVO vo = mapping.getVOInstance();
    Class clazz = mapping.getVOInstance().getClass();
    Long id = (Long) f.get("id");
    AppGenericVO voBD =
AppFactory.getInstance().getBO(clazz).load(clazz, id);
    Map mapProperties = f.getMap();
    mapProperties.remove("id");
    Iterator it = mapProperties.keySet().iterator();
    while (it.hasNext()) {
        String key = (String) it.next();
        BeanUtils.copyProperty(voBD, key,
mapProperties.get(key));
    }
    vo = voBD;
    IAppBO bo = AppFactory.getInstance().getBO(clazz);
    vo = bo.update(vo);
    request.setAttribute("vo", vo);
    valueObjectToForm(vo, f);
    f.set("id", id);
    return vo;
}

protected AppGenericVO executeInsert(AppActionMapping mapping,
DynaActionForm f, HttpServletRequest request, HttpServletResponse response)
throws Exception {
    AppGenericVO vo = mapping.getVOInstance();
    BeanUtils.copyProperties(vo,
RequestUtil.getMapParametros(request));
    this.executeInsert(mapping, f, request, response, vo);
}

```

```

        return vo;
    }

    protected AppGenericVO executeInsert(AppActionMapping mapping,
    DynaActionForm f, HttpServletRequest request, HttpServletResponse response,
    AppGenericVO vo) throws Exception {
        IAppBO bo = AppFactory.getInstance().getBO(vo.getClass());
        vo = bo.insert(vo);
        request.setAttribute("vo", vo);
        valueObjectToForm(vo, f);
        f.set("id", vo.getId());
        return vo;
    }

    protected void executeDelete(AppActionMapping mapping, DynaActionForm
    f, HttpServletRequest request, HttpServletResponse response) throws
    Exception {
        AppGenericVO vo = mapping.getVOInstance();
        vo = carregaObjeto(mapping, f, request, response);
        IAppBO bo =
    AppFactory.getInstance().getBO(mapping.getVOInstance().getClass());
        bo.delete(vo);
    }

    protected AppGenericVO executeMethod(AppActionMapping mapping,
    DynaActionForm f, HttpServletRequest request, HttpServletResponse response,
    AppGenericVO vo) throws Exception {
        BeanUtils.copyProperties(vo,
    RequestUtil.getMapParametros(request));
        IAppBO bo =
    AppFactory.getInstance().getBO(mapping.getVOClass());
        vo = bo.executeMethod(mapping.getMetodo(), vo);
        request.setAttribute("vo", vo);
        valueObjectToForm(vo, f);
        f.set("id", vo.getId());
        return vo;
    }

    private boolean permittedContent(Object obj, AppActionMapping mapping,
    HttpServletRequest request) throws Exception {
        if (!mapping.aplicaSeguranca()) {
            return true;
        } else {
            for (VOContentControl control :
    mapping.getVOContentControls()) {
                String vSession =
    request.getSession().getAttribute(control.getSessionAtributo()).toString();
                String vObject = BeanUtils.getProperty(obj,
    control.getPropriedade());
                if (!vSession.equalsIgnoreCase(vObject)) {
                    return false;
                }
            }
        }
        return true;
    }

    public void inicializaDadosNaSession(HttpServletRequest request) {
        String dtAtual = DateFormatUtil.getDataAtualStr();
        request.getSession().setAttribute("data_de_hoje", dtAtual);
    }

```

```

        public String getResource(String path) {
            try {
                return
this.getServlet().getServletContext().getRealPath(path);
            } catch (Exception e) {
                e.printStackTrace();
            }
            return null;
        }
    }
}

```

## D.2 – CÓDIGO DA CLASSE

### APPGENERICABO.JAVA

```

package br.com.ibarra.model;

import java.lang.reflect.Method;
import java.util.List;

import br.com.ibarra.dao.AppGenericDAO;
import br.com.ibarra.dao.IAppDAO;
import br.com.ibarra.dao.annotation.DaoClass;
import br.com.ibarra.exception.BOException;
import br.com.ibarra.exception.DAOException;
import br.com.ibarra.util.AppFactory;
import br.com.ibarra.vo.AppGenericVO;

@DaoClass(AppGenericDAO.class)
public class AppGenericBO implements IAppBO {

    public IAppDAO getDao() throws DAOException {
        return AppFactory.getInstance().getDAO(this.getClass());
    }

    public AppGenericVO insert(AppGenericVO vo) throws BOException {
        try {
            vo = getDao().insert(vo);
        } catch (DAOException e) {
            throw new BOException(e);
        } catch (Exception e) {
            throw new BOException(e);
        }
        return vo;
    }

    public AppGenericVO update(AppGenericVO vo) throws BOException {
        try {
            // AppGenericVO voAlterado = getDao().load(vo.getClass(),
            // vo.getId());
            // Map map = BeanUtils.describe(vo);
            // map.remove("id");
            // BeanUtils.copyProperties(voAlterado, map);
            vo = getDao().update(vo);
        } catch (DAOException e) {

```

```

        throw new BOException(e);
    } catch (Exception e) {
        throw new BOException(e);
    }
    return vo;
}

public void delete(AppGenericVO vo) throws BOException {
    try {
        getDao().delete(vo);
    } catch (DAOException e) {
        throw new BOException(e);
    }
}

public List query(Class clazz) throws BOException {
    List retorno;
    try {
        retorno = getDao().query(clazz);
    } catch (DAOException e) {
        throw new BOException(e);
    } catch (Exception e) {
        throw new BOException(e);
    }
    return retorno;
}

public AppGenericVO load(Class objectClass, Long id) throws
BOException {
    AppGenericVO vo;
    try {
        vo = getDao().load(objectClass, id);
    } catch (DAOException e) {
        throw new BOException(e);
    } catch (Exception e) {
        throw new BOException(e);
    }
    return vo;
}

public AppGenericVO executeMethod(String methodName, AppGenericVO vo)
throws BOException {
    try {
        Class partypes[] = new Class[1];
        partypes[0] = AppGenericVO.class;
        Method method = this.getClass().getMethod(methodName,
partypes);
        Object arglist[] = new Object[1];
        arglist[0] = vo;
        AppGenericVO retobj = (AppGenericVO) method.invoke(this,
arglist);
        return retobj;
    } catch (Exception e) {
        e.printStackTrace();
    }
    return null;
}
}

```

## D.3 – CÓDIGO DA CLASSE

### APPGENERICADA.O.JAVA

```
package br.com.ibarra.dao;

import java.util.List;
import java.util.Map;

import org.hibernate.Query;

import br.com.ibarra.exception.DAOException;
import br.com.ibarra.exception.PersistenceException;
import br.com.ibarra.persistence.hibernate.HibernateUtil;
import br.com.ibarra.vo.AppGenericVO;

public class AppGenericDAO implements IAppDAO {

    public AppGenericVO insert(AppGenericVO vo) throws DAOException {
        Long id;
        try {
            id = (Long) HibernateUtil.save(vo);
        } catch (PersistenceException e) {
            throw new DAOException(e);
        }
        vo.setId(id);
        return vo;
    }

    public AppGenericVO update(AppGenericVO vo) throws DAOException {
        try {
            HibernateUtil.save(vo);
        } catch (PersistenceException e) {
            throw new DAOException(e);
        }
        return vo;
    }

    public void delete(AppGenericVO vo) throws DAOException {
        try {
            HibernateUtil.delete(vo);
        } catch (PersistenceException e) {
            throw new DAOException(e);
        }
    }

    public List query(Class clazz) throws DAOException {
        try {
            return HibernateUtil.getList(clazz);
        } catch (PersistenceException e) {
            throw new DAOException(e);
        }
    }
}
```



```

    public AppGenericVO load(Class objectClass, Long id) throws
    DAOException {
        try {
            return (AppGenericVO) HibernateUtil.load(objectClass,
id);
        } catch (PersistenceException e) {
            throw new DAOException(e);
        }
    }

    public List executeNamedQuery(String namedQuery, Map param) throws
    DAOException {
        try {
            Query query = HibernateUtil.getNamedQuery(namedQuery);
            HibernateUtil.setParametrosQuery(query, param);
            return HibernateUtil.executeNamedQuery(query);
        } catch (PersistenceException e) {
            throw new DAOException(e);
        }
    }

    public List executeNamedQuery(String namedQuery) throws DAOException
    {
        try {
            Query query = HibernateUtil.getNamedQuery(namedQuery);
            return HibernateUtil.executeNamedQuery(query);
        } catch (PersistenceException e) {
            throw new DAOException(e);
        }
    }

    public List executeNamedQuery(String namedQuery, Map param, int
    first, int max) throws DAOException {
        try {
            Query query = HibernateUtil.getNamedQuery(namedQuery);
            HibernateUtil.setParametrosQuery(query, param);
            return HibernateUtil.executeNamedQuery(query, first,
max);
        } catch (PersistenceException e) {
            throw new DAOException(e);
        }
    }

    public int updateByNamedQuery(String namedQuery, Map param) throws
    DAOException {
        try {
            Query query = HibernateUtil.getNamedQuery(namedQuery);
            HibernateUtil.setParametrosQuery(query, param);
            return HibernateUtil.updateNamedQuery(query);
        } catch (PersistenceException e) {
            throw new DAOException(e);
        }
    }

    public List executeQuery(String sql) throws DAOException {
        try {
            Query query = HibernateUtil.createSQLQuery(sql);
            return query.list();
        } catch (PersistenceException e) {
            throw new DAOException(e);
        }
    }
}

```

```
}
```

## D.4 – CÓDIGO DA CLASSE APPFACTORY.JAVA

```
package br.com.ibarra.util;

import java.util.HashMap;
import java.util.Map;

import br.com.ibarra.dao.AppGenericDAO;
import br.com.ibarra.dao.IAppDAO;
import br.com.ibarra.dao.annotation.DaoClass;
import br.com.ibarra.exception.BOException;
import br.com.ibarra.exception.DAOException;
import br.com.ibarra.model.AppGenericBO;
import br.com.ibarra.model.IAppBO;
import br.com.ibarra.model.annotation.BoClass;

public class AppFactory {

    private static AppFactory factory = new AppFactory();

    private Map<Class, IAppDAO> daoCache = new HashMap<Class, IAppDAO>();

    private Map<Class, IAppBO> boCache = new HashMap<Class, IAppBO>();

    private AppFactory() {
    }

    public static AppFactory getInstance() {
        return factory;
    }

    public IAppBO getBO(Class clazz) throws BOException {
        if (boCache.containsKey(clazz)) {
            return boCache.get(clazz);
        } else {
            if (!clazz.isAnnotationPresent(BoClass.class))
                throw new BOException("Anotação @BoClass ausente");
            BoClass boClassInfo = (BoClass)
clazz.getAnnotation(BoClass.class);
            IAppBO bo = null;
            try {
                bo = boClassInfo.value().newInstance();
                if (boClassInfo.singleton())
                    boCache.put(clazz, bo);
            } catch (Exception e) {
                bo = new AppGenericBO();
                boCache.put(clazz, bo);
            }
            return bo;
        }
    }

    public IAppDAO getDAO(Class clazz) throws DAOException {
```

```

        if (daoCache.containsKey(clazz)) {
            return daoCache.get(clazz);
        } else {
            if (!clazz.isAnnotationPresent(DaoClass.class))
                throw new DAOException("Anotação @DaoClass
ausente");

            DaoClass daoClassInfo = (DaoClass) clazz
                .getAnnotation(DaoClass.class);
            IAppDAO dao = null;
            try {
                dao = daoClassInfo.value().newInstance();
                if (daoClassInfo.singleton())
                    daoCache.put(clazz, dao);
            } catch (Exception e) {
                dao = new AppGenericDAO();
                daoCache.put(clazz, dao);
            }
            return dao;
        }
    }
}

```



# Anexo E – Artigo

## Framework de apoio ao desenvolvimento de aplicações web com banco de dados, utilizando Struts, Tiles e Hibernate

David Pedro Willemann (davidpw@inf.ufsc.br)

Gustavo Bestetti Ibarra (gibarra@inf.ufsc.br)

**Universidade Federal de Santa Catarina  
Departamento de Informática e Estatística  
Florianópolis/SC, Julho de 2007.**

### Resumo

O presente artigo refere-se ao desenvolvimento de um framework de apoio a criação de sistemas em ambiente web que utilizam banco de dados para armazenamento de suas informações, com foco nas operações básicas de CRUD (Create Retrieve Update Delete), oferecendo uma maior produtividade no desenvolvimento destes tipos de sistemas.

### 1 Introdução

A história da Internet pode parecer surpreendente para quem a conheceu há pouco tempo. A necessidade da comunicação entre os computadores surgiu remotamente por volta dos anos 60 durante a guerra fria com o fim de proteger dados militares através da distribuição das informações em diversos computadores espalhados pelo mundo. Para atender essa necessidade foi criada a rede ARPA-NET pelo grupo de pesquisa Advanced Research Projects Agency (ARPA), interligando três computadores no final de 1969. Três anos depois já eram 40 computadores interligados. Mas logo se percebeu que possuir computadores interligados apenas para fins militares não seria interessante.

Assim o meio científico se interessou pela ARPA-NET como um meio de obter resultados de pesquisas realizadas em outras instituições. Com isso, o número de computadores interligados aumentou muito, ocasionando a separação entre a parte militar e a parte civil. Nos anos 80 a National Science Foundation (NSF) interligou os mais importantes centros científicos à redes menores de

universidades, fazendo com que diversas redes fossem unidas. Então, o que era chamado de ARPANET foi batizado de Internet.

Hoje podemos encontrar informações de todos os tipos e áreas publicadas na Internet e numa infinidade de locais, que vão desde uma simples página pessoal até grandes portais de vendas de mercadorias e transações eletrônicas.

Outra coisa que vem acontecendo ao longo dos últimos anos é a utilização da internet como ambiente para utilização de sistemas. Devido a facilidade que a Internet fornece, no sentido em que o sistema está disponível em qualquer lugar e a qualquer momento, sem dificuldades de instalação, bastando ter um navegador disponível para acessar a internet.

Visando o este nicho de e a necessidade de ter agilidade no desenvolvimento deste tipo de sistema, este artigo descreve a construção de um arcabouço (framework) em Java que integre algumas tecnologias e frameworks existentes no mercado, e mostra como foi testado e avaliado através do desenvolvimento de uma aplicação sob a forma de prova de conceito.

## **2 Padrões de Projeto**

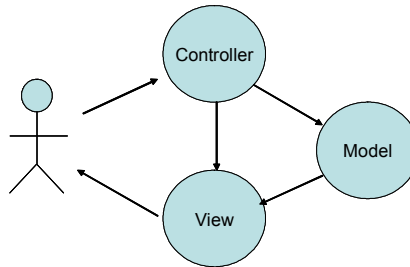
Os Padrões de Projeto são soluções simples para problemas específicos no projeto de software orientado a objetos, usando soluções que foram desenvolvidas e aperfeiçoadas ao longo do tempo.

Arquiteturas orientadas a objetos bem-estruturadas estão cheias de padrões, uma das maneiras de medir a qualidade de um sistema orientado a objetos é avaliar se os desenvolvedores tiveram bastante cuidado com as colaborações comuns entre seus objetos. Levando em consideração tais mecanismos durante o desenvolvimento de um sistema, conduz-se a uma arquitetura menor, mais simplória e muito mais compreensível que aquelas produzidas quando estes padrões não são considerados.

Um bom framework deve estar repleto de padrões de projetos, visando sempre algumas características básicas para um bom framework: baixo acoplamento, alta coesão, reusabilidade, pontos de flexão e extensibilidade.

## **3 Modelo MVC**

Segundo Lozano [2004], a sigla MVC denota os três papéis no qual todo componente da aplicação deve ser classificado: Model, View e Controller, ou em português: Modelo, Visualização e Controlador. Consiste em um padrão de arquitetura de aplicações que visa separar a lógica da aplicação (Model), da interface do usuário (View) e do fluxo da aplicação (Controller). Permite que a mesma lógica de negócios possa ser acessada e visualizada por várias interfaces.



**Figura 35 - Modelo MVC**

MVC também é utilizado em padrões de projetos de software, porém, MVC abrange mais da arquitetura de uma aplicação do que é típico para um padrão de projeto.

Componentes de modelo são responsáveis pelo armazenamento dos dados durante a sessão do usuário para a execução de regras de negócios, sendo que os componentes de visualização se encarregam da exibição final e de fornecer meios do usuário indicar as operações a serem realizadas pela aplicação. Componentes do controlador são responsáveis em fazer a ponte entre os outros dois tipos de componentes, interpretando os eventos gerados pelos componentes de visualização e disparando a execução do método correspondente ao Modelo.

## **4 Framework**

Várias definições sobre framework são descritas na literatura, mas segundo Gamma "um framework é um conjunto de classes que cooperam entre si provendo assim um projeto reutilizável para um domínio específico de classes de sistema".

Um framework ou arcabouço é uma estrutura de suporte definida em que um outro projeto de software pode ser organizado e desenvolvido, quando se analisa o conceito no âmbito do desenvolvimento de software. Um framework pode incluir programas de suporte, bibliotecas de código, linguagens de script e outros softwares para ajudar a desenvolver e juntar diferentes componentes de um projeto de software.

Os Frameworks são projetados com o propósito de facilitar o desenvolvimento de software, habilitando projetistas e programadores a gastarem mais tempo detalhando as exigências de negócio do software do que com detalhes tediosos de baixo nível do sistema.

Um bom framework deve apresentar as seguintes qualidades: generalidade, alterabilidade, extensibilidade, clareza, completude, simplicidade, fronteiras e ganchos.

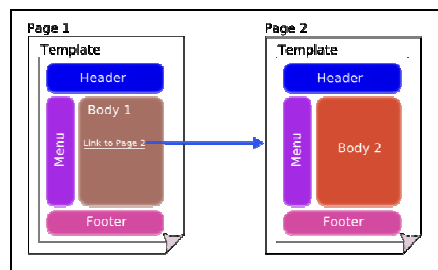
## 5 Struts

O framework Struts foi desenvolvido por Craig McClanahan e doado, em maio de 2002, à Apache Foundation. Atualmente há uma comunidade de desenvolvedores espalhados pelo mundo, trabalhando de forma cooperativa neste projeto [31].

Este é um modelo de código aberto, proposto pela Jakarta, útil para construir aplicativos Web, usando Servlets e JSP. Struts estimula o uso de uma arquitetura de aplicação baseada no paradigma de projeto Model-View-Controller (MVC). Ele fornece seu próprio componente de controle e o integra com outra tecnologias para possibilitar o Modelo e a Visão [14]. Com relação ao Modelo, Struts pode interagir com qualquer tecnologia padrão de acesso a dados. Quanto à Visão, o modelo Struts funciona bem com JSP.

## 6 Tiles

A maioria dos sites e sistemas Web obedecem geralmente um padrão de interface, variando apenas alguns pedaços (recortes) entre as páginas visualizadas pelos usuários e é neste contexto que o Tiles se encaixa no desenvolvimento de sistemas Web [30]. Tiles é um framework Open-Source que permite aos desenvolvedores Web aplicarem o padrão de projeto Composite-View. O exemplo ilustrado implica uma situação típica nos sites, onde tem-se uma estrutura padrão entre a páginas, e alguma ação do usuário altera apenas uma parte (recorte) da página.



**Figura 36 - Exemplo de Recortes Tiles: Fonte [30]**

Observa-se que entre uma página e outra somente o conteúdo central é modificado, deste modo, basta definir-se uma estrutura no Tiles com os 4 recortes do layout (Header, Menu, Body e Footer) e então, para cada página modificar somente o recorte definido na Body.

## 7 Hibernate

Hibernate é um framework que permite ao desenvolvedor liberta-se de preocupações com a persistência de seus objetos. Com ele, pode-se fazer o mapeamento de objetos para bancos de dados



relacionais, persistindo os objetos em tabelas de um banco de dados relacional, permitindo a convivência dos dois paradigmas (relacional e objetos) [21].

O hibernate também fornece uma linguagem própria de seleção, o Hibernate Query Language (HQL) similar ao SQL, mas que possibilita a seleção de objetos ao invés de linhas. O mapeamento objeto-tabela é feito através de arquivos XML. A configuração das características do banco de dados no Hibernate pode ser feita de diversas formas como, por exemplo, através da própria interface do framework ou através de arquivos de configuração XML.

As operações de criação, atualização, remoção e seleção de objetos são feitas através da API do Hibernate que, consultando seus arquivos de mapeamento e configuração, envia os comandos corretos ao banco de dados configurado. Assim, a aplicação fica portátil entre bancos de dados.

## 8 Framework Desenvolvido

Como já comentado anteriormente, o objetivo deste artigo é descrever o desenvolvimento de um framework voltado para o desenvolvimento de sistemas Web que utilizam banco de dados como forma de persistência de informações. Para tanto, o framework provê uma arquitetura em três camadas, obedecendo o padrão MVC de desenvolvimento de aplicações. O foco do framework está em oferecer um mecanismo automático, onde o desenvolvedor de sistemas não necessite codificar muitas linhas, que permita a persistência e recuperação de objetos em algum banco de dados. Neste contexto, o framework provê todas as funcionalidades para o desenvolvedor construir telas que permitam as operações básicas sobre um objeto, como criação, alteração, exclusão e recuperação, comumente chamadas de CRUD (Create Retrieve Update Delete).

Este framework foi desenvolvido em Java, utilizando-se de outros frameworks e ferramentas de código aberto e aplicando os conceitos de padrões de projetos e arquitetura MVC. Desta forma o framework define as três camadas básica do modelo MVC: Modelo, Visão e Controle. No entanto, a camada de Modelo ainda foi subdividida em outras três camadas. Assim, pode-se classificar a arquitetura como:

**View:** define as interfaces com o usuário (HTML, JSP, Servlets, Tiles)

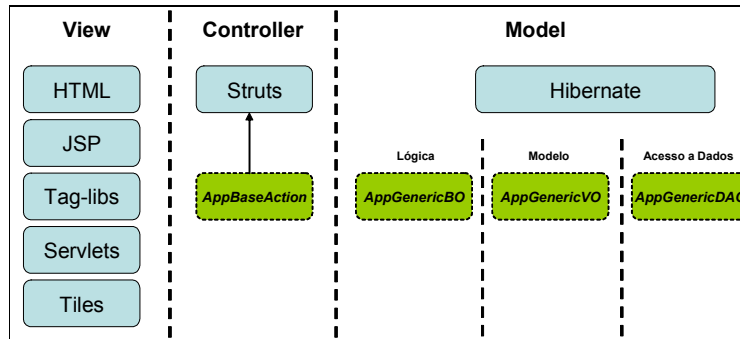
**Controller:** controla as requisições e respostas entre cliente (navegador) e servidor.

**Model:** controla as lógicas de negócio e dados. Está subdividida em:

**Modelo:** define os objetos de domínio da aplicação

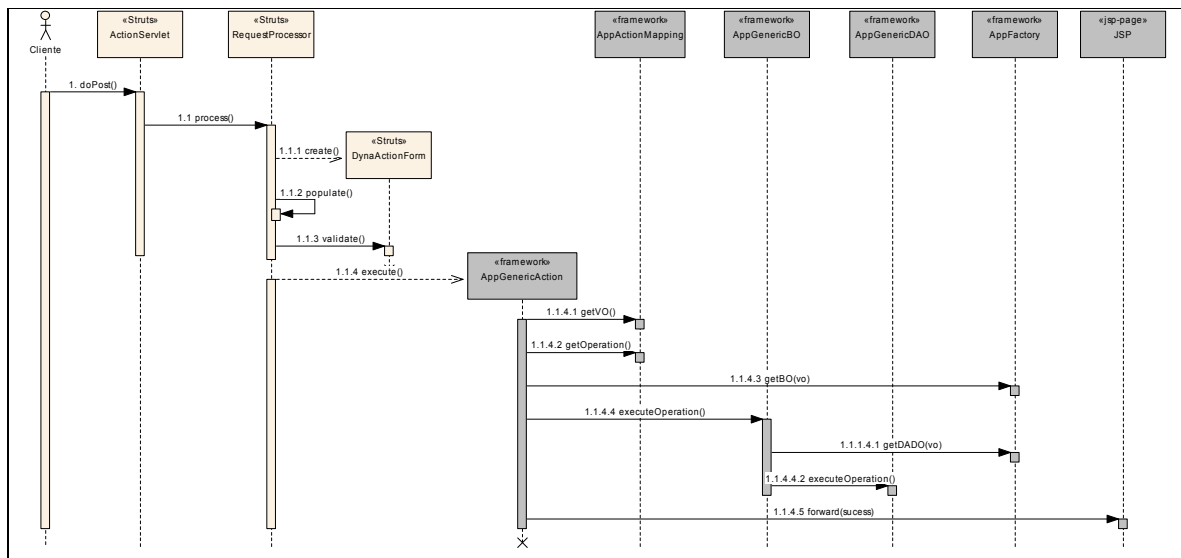
**Lógica:** define as regras de negócio envolvidas nas operações de CRUD de cada classe do modelo.

**Persistência:** define as regras de persistência e recuperação de cada classe do modelo.



**Figura 37 - Visão geral da Arquitetura do framework**

O diagrama abaixo ilustra uma operação básica realizada pelo framework, mostrando assim toda a forma de comunicação entre as diversas camadas definidas pelo framework.



**Figura 38 - Visão geral do funcionamento do framework**

Analisando o diagrama podemos observar:

Quando um usuário faz uma requisição, esta é interceptada pelo Struts através da ActionServlet, que inicia os controles sobre a requisição, criando um RequestProcessor e carregando as configurações do arquivo struts-config.xml.

Com base nesta configurações, uma instancia das classes ActionForm(DynaActionForm), ActionMapping (AppActionMapping) e da classe Action(AppGenericAction) são criadas e pré-configuradas de acordo com os parâmetros especificados no arquivo de configuração.

A validação das informações é feita de forma automática pelo struts através da inferência dos dados existentes na instancia da ActionForm junto às regras definidas pelo desenvolvedor, contidas no arquivo validation.xml.

Uma vez que os parâmetros da requisição foram validados, o Struts, através da implementação do padrão Command, faz uma chamada ao método execute da instancia da Action criada.

Neste momento o framework está instanciado, e começa a executar todo seu fluxo padrão. Ao executar o método execute() da AppGenericAction, o framework realiza algumas validações e recupera alguns parâmetros das configurações específicas dele através da instancia da classe AppActionMapping.

Um dos principais parâmetros é o vo, que identifica qual é a classe principal do domínio da aplicação que será tratada na action. Outro parâmetro importante é o operacao, que define qual ação será realizada com o objeto (vo) recuperado.

Uma vez recuperados estes dois parâmetros, o sistema cria uma instancia da classe AppGenericBO, responsável pela implementação das regras para o objeto que está sendo tratado, e faz a chamada ao método específico conforme os parâmetros definidos pelo desenvolvedor.

Na instância da lógica do negócio, são realizadas as regras necessárias para a execução da operação e a criação da classe AppGenericDAO, responsável pelo acesso e persistência dos objetos. Em seguida é executado o método da classe AppGenericDAO correspondente a operação que está sendo realizada.

Ao final de todo o fluxo é realizado um redirecionamento para a action ou página definida no struts-config.xml, colocando no request todas as informações recuperadas através das operações realizadas pelo framework.

Customizações das operações realizadas pelo framework são realizadas basicamente através dos parâmetros configurados pelo desenvolvedor no arquivo struts-config.xml e através de especializações das classes AppGenericBO e AppGenericDAO.

A seguir estão descritas algumas informações sobre cada camada da arquitetura.

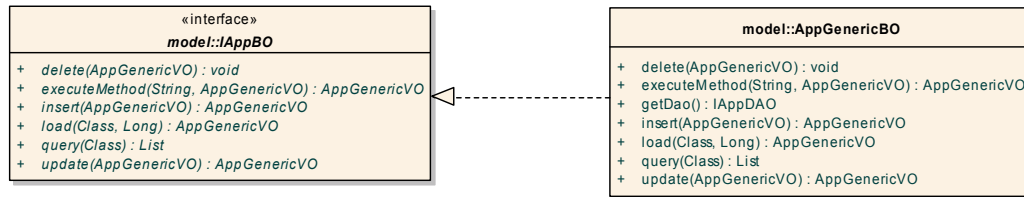
## **1 Camada Model (M)**

Esta camada é a responsável por definir os objetos do domínio da aplicação, regras de negócio envolvidas e regras de persistência e recuperação dos dados. Para uma melhor estruturação do framework e garantia de baixo acoplamento e alta coesão, esta camada foi subdividida em 3 sub-camadas: Modelo, Lógica e Persistência.

### **1.1 Camada Lógica**

A camada de lógica é a responsável por todas as regras de negócio envolvidas com determinado objeto do domínio da aplicação, inclusive é nesta camada que se deve desenvolver toda a complexidade e interação com outros objetos e suas lógicas de negócio.

O framework implementa as regras básicas de CRUD (criação, alteração, recuperação e exclusão) de objetos através da classe AppGenericBO. O sufixo BO vem de *Business Object*, ou Objetos de Negócios. Qualquer regra adicional às operações básicas de um objeto deverá ser implementada através de herança desta classe e sobrescrita do método apropriado ou criação de novo método.



**Figura 39 - Lógicas de Negócio no framework**

### 1.2 Camada de Modelo

Nesta camada é necessário a definição pelo desenvolvedor das classes que definem os objetos do domínio do negócio, somente com seus atributos e respectivos getters e setters. Os objetos de domínio de negócios são os chamados VO's (*Value Objects*) que podem ser vistos como as tabelas de um banco de dados. Não necessariamente um objeto corresponderá a uma tabela ou o contrário, uma tabela corresponde a um objeto. Este mapeamento é realizado através das anotações do hibernate, e podem existir vários cenários, como por exemplo, o mapeamento de um objeto para mais de uma tabela.

O framework define uma estrutura padrão de objeto através da classe AppGenericVO. Todo objeto de domínio de uma aplicação deve ser definido em uma classe que estenda a classe AppGenericVO. O exemplo de um objeto de domínio de negócio pode ser visto abaixo:

```

@Entity
@Table(name = "USER")
@SequenceGenerator(name="SEQ_ID", sequenceName="SEQ_ID")
public class User extends AppGenericVO {

    @Id
    @GeneratedValue(strategy = javax.persistence.GenerationType.AUTO)
    @Column(name = "ID")
    protected Long id;

    @Column(name = "NAME", nullable = false, length = 50)
    private String name;

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

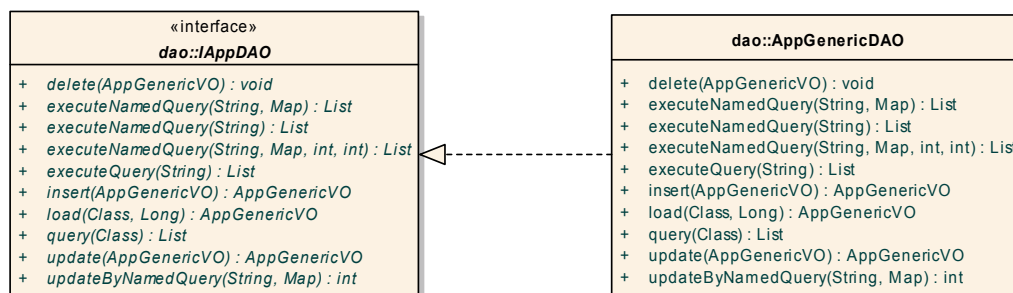
    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
  
```

### 1.3 Camada de Acesso a Dados

Esta camada, como diz seu nome, é a responsável pelo acesso e persistência dos objetos em um banco de dados. Ela mantém a conexão com o banco de dados, realiza as consultas e define os algoritmos para realizar qualquer operação básica de CRUD de objetos. Esta camada utiliza-se de todos os recursos do framework hibernate para fazer o acesso e persistência dos objetos no banco de

dados. O framework implementa todas as funcionalidades básicas através da classe AppGenericDAO. O sufixo DAO vem de *Data Access Objects*.



**Figura 40 - Acesso e Persistência de dados no framework**

## 2 Camada de Visão (V)

A camada de visão é a menos trabalhada neste framework pois com ela existe dificuldade de se padronizar interfaces de sistema. Entretanto, o framework utiliza-se de outros frameworks e bibliotecas para facilitar os controles de interface. Nesta camada são utilizadas bibliotecas para facilitar a exibição de objetos e listas e o framework Tiles para controlar os leiautes da aplicação.

## 3 Camada de Controle (C)

Esta é a camada responsável por receber as requisições dos usuários e respondê-las de forma adequada, através de chamadas aos objetos da camada de lógica. Para o suporte a camada de controle o framework disponibiliza a classe AppBaseAction, que estende a classe Action do struts, oferecendo todas as opções necessárias ao controle das operações de CRUD de um objeto.

Para cada operação precisa ser definida uma action no struts, que baseia-se na classe AppBaseAction. Esta classe ainda trabalha em conjunto com a classe AppActionMapping que é a responsável pelas opções de configuração da AppBaseAction. Para cada action do framework podem ser definidos os seguintes parâmetros:

Parâmetro	Descrição
<i>vo</i>	Nome completo da classe de domínio que será objeto de uma das operações CRUD
<i>operacao</i>	Define a operação a ser realizada com o objeto de definido no parâmetro <i>vo</i> . As operações disponíveis são: load – carrega um objeto baseado no seu id. update – altera os dados de um objeto insert – insere um novo objeto no banco de dados. delete – exclui um objeto
<i>nextAction</i>	Define a próxima ação a ser disparada depois da ação corrente
<i>limpaForm</i>	limpa os todos os dados do formulário passado a action
<i>executeNamedQuery</i>	Define qual consulta deve ser executada para recuperar informações dos objetos.
<i>metodo</i>	Define a chamada a algum método específico implementado em uma classe especializada pelo desenvolvedor.
<i>carregaLista</i>	Define se deve ser carregado alguma lista no request. É utilizado quando deseja-se montar uma combobox ou lista com objetos recuperados do banco de dados.

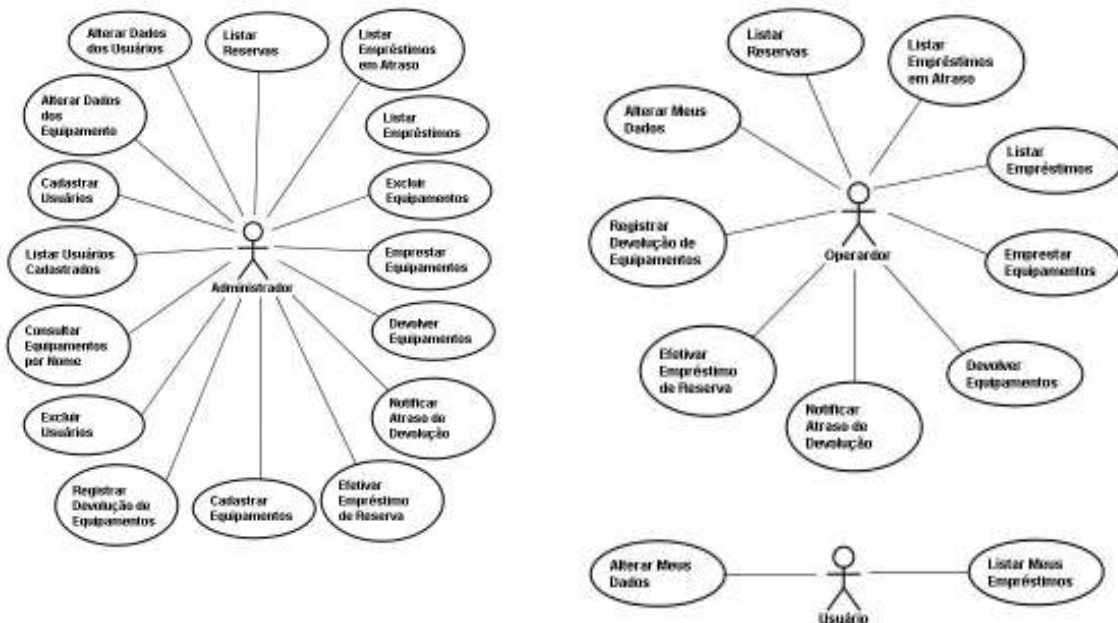
A seguir demonstra-se um exemplo de uma configuração de uma action que recupera um objeto do banco de dados baseado em seu id (código):

```
<action path="/carregarequipamento"
  className="br.com.ibarra.web.struts.AppActionMapping"
  name="equipamentoCadForm" scope="request"
  validate="false"
  type="br.com.ibarra.web.struts.actions.AppGenericAction">
  <set-property property="vo"
    value="tcc.sgeem.vo.Equipamento" />
  <set-property property="operacao"
    value="load" />
  <set-property property="nextAction"
    value="/alterarequipamento.do" />
  <forward name="sucesso" path="equipamento.cadastro" />
</action>
```

## 9 Validação do Framework – Estude de Caso

Para validar o framework foi desenvolvido uma aplicação de controle de empréstimo de equipamentos para o LABMETRO, que é um laboratório de metrologia da UFSC. Nesta aplicação foram definidas várias funcionalidades e todas elas foram implementadas através do framework desenvolvido, como forma de prova de conceito do framework.

Esta aplicação foi batizada com o nome SGEL (Sistema de Ggerenciamento de Empréstimos do Labmetro), e totalizou 18 casos de uso, sendo utilizados por 3 atores: Administrados, Operador e Usuário. Abaixo estão ilustrados os casos de uso e suas relações com os atores.



Podemos considerar que a implementação do SGEL foi a parte mais importante e especial deste trabalho, pois neste momento tivemos todas as provas que os conceitos e funcionalidades desenvolvidas no framework funcionavam como esperado.

Para sua implementação partimos da especificação dos casos de uso e modelagem das classes de domínio da aplicação cujo código fonte pode ser visto no Anexo I deste documento.

O segundo passo foi desenvolver uma página JSP para cada caso de uso capaz de apresentar os dados correspondentes aos objetos e informações tratadas nele.

Em seguida foi analisado qual classe de domínio estava envolvida em cada caso de uso, e qual operação deveria ser realizada, tentando sempre encaixar nas operações básicas de CRUD disponibilizadas pelo framework: load, update, insert, delete.

Neste ponto, observamos que a maioria dos casos de uso encaixavam-se diretamente nas funcionalidades providas pelo framework. O resultado da análise pode ser visto na tabela abaixo:

Caso de Uso	Necessitou Especialização?	
	Sim	Não
cadastrar usuários		X
listar usuários cadastrados		X
alterar dados dos usuários		X
excluir usuários		X
cadastrar equipamentos		X
consultar equipamentos (por nome)		X
alterar dados dos equipamentos		X
excluir equipamento		X
emprestar equipamento	X	
listar empréstimos		X
listar reservas		X
listar empréstimos em atraso		X
registrar devolução de equipamento	X	
excluir empréstimo	X	
notificar atraso de devolução	X	
efetivar empréstimo de reserva	X	
alterar meus dados		X
listar meus empréstimos		X

Analisando a tabela acima, podemos perceber que de um total de 18 casos de uso do sistema, apenas 5 precisaram de algum tipo de customização através de especialização das classes do framework o que nos dá um índice de 72,2% de cobertura das funcionalidades pelo framework, caracterizando assim um ganho de produtividade considerável.

Podemos ainda observar que os casos em que foram necessárias algum tipo de customização são todos referente a lógica de empréstimos de equipamentos, que é justamente onde concentra-se o objeto de negócio da aplicação. O nome do sistema em estudo é Sistema de Gerenciamento de Empréstimos do Labmetro, ou seja, o controle é sobre empréstimos, e nada mais natural do que este ponto não ser coberto de maneira completa por um framework cujo domínio é genérico e voltado a aplicações Web com banco de dados.

No entanto esta customização foi simples, e bastou definir regras de negócio específicas para cada caso de uso, que se resumiu na criação de uma classe `EmprestimoBO` que especializa `AppGenericBO`, acrescentando métodos específicos para cada caso de uso.

## 10 Conclusão

A abordagem de frameworks contribui significativamente para reutilização no desenvolvimento de artefatos de software. Frameworks é uma abordagem que está sendo cada vez mais utilizada, com o objetivo de diminuir o tempo e esforços no desenvolvimento de artefatos de software. Para se ter um bom projeto, este deve ser bem estruturado e planejado. Padrões de projeto ajudam o projetista no desenvolvimento de projetos melhor estruturados, oferecendo soluções que foram desenvolvidas e aperfeiçoadas ao longo do tempo.

A definição de uma estrutura padronizada para um determinado domínio de aplicações, juntamente com o desenvolvimento uma série de componentes oferecendo funcionalidades prontas para o domínio através da aplicação de padrões de projeto, caracterizam o projeto de um framework.

O framework que trata este artigo disponibiliza uma série de funcionalidades prontas para o desenvolvimento de aplicações Web utilizando banco de dados. Seu foco está na separação das camadas de dados, controle e visualização, através da aplicação do padrão de projeto MVC (Model View Controller), e no suporte as operações básicas de CRUD (Create Retrieve Update Delete) sobre objetos.

A utilização da arquitetura MVC oferece vantagens para desenvolvedores como a otimização das habilidades de equipes através de especialidades específicas de cada um, e a redução de custos associados ao desenvolvimento, além de favorecer a extensibilidade e reutilização do código.

Além disso, o desenvolvimento de aplicações baseadas em um framework com uma arquitetura definida, permite uma maior qualidade do código e do projeto, uma vez que a aplicação é construída através do reuso de componentes e obedecendo uma estrutura bem definida e padronizada.

Outro ponto importante oferecido pelo framework é o suporte às operações básicas de CRUD sobre objetos, que permite um desenvolvimento rápido de funcionalidades básicas de um sistema, que na maioria dos casos somam mais de 50% do projeto.

O desenvolvimento de aplicações utilizando o framework desenvolvido através da aplicação de sua estrutura e utilização de seus componentes, produz aplicações mais robustas e com uma excelente qualidade. Além disso, o tempo de desenvolvimento torna-se muito menor, diminuindo consideravelmente os custos do projeto.

Comprovamos tudo isso através do desenvolvimento do SGEL, como comentado no tópico anterior, onde conseguimos resultados muito animadores e que justificam todo o esforço despendido no desenvolvimento do framework. Dos dezoito casos de uso do SGEL, treze deles foram desenvolvidos sem qualquer customização das funcionalidades oferecidas pelo framework, definindo



uma cobertura do framework de 72,2% da aplicação construída. Os outros cinco casos de uso do sistema necessitaram de uma pequena especialização das funcionalidades básicas do framework e adição de novas funcionalidades específicas das regras do SGEL. Estas funcionalidades foram customizadas com a especialização de apenas uma classe do framework, cujo código resumiu-se em 111 linhas.

Com tudo isso, observamos que o framework realmente atingiu seus objetivos, definindo uma arquitetura consistente e oferecendo uma maior produtividade no desenvolvimento de sistemas.

## 11 Referências

- [1] AMBLER, Scott W. Análise e projeto orientado a objeto – Seu guia para desenvolver sistemas robustos com tecnologia de objetos. Tradução Oswaldo Zanelli. Rio de Janeiro: Infobook, 1998.
- [2] ARAGÃO JÚNIOR, Marício. Hibernate 3 Avançado, Boas Práticas, padrões e caching. Disponível em <http://www.guj.com.br/java/tutorial/artigo.181.1.guj>.
- [3] BOOCH, Grady; RUMBAUGH, James; JACOBSON, Ivar. UML Guia do Usuário. Rio de Janeiro: Campus. 2001.
- [4] CARDOSO, C. HTML Truques espertos. Rio de Janeiro: Axcel Books, 1996.
- [5] CASTRO, M. A. S. O que é HTML. São Paulo, 2003. Disponível em: <http://www.icmc.usp.br/ensino/material/html/html.html>.
- [6] CAVALHIERI, Marcos A. Programação Java. Disponível em: [http://cognitio.incubadora.fapesp.br/portal/atividades/cursos/instrumentacao/java/ProgramacaoJava\\_01.pdf](http://cognitio.incubadora.fapesp.br/portal/atividades/cursos/instrumentacao/java/ProgramacaoJava_01.pdf).
- [7] DAMASCENO JÚNIOR, Américo. Aprendendo JAVA: programação na Internet. São Paulo. Érica. 2001
- [8] DEITEL, H. M. Java. How to Program. Prentice Hall 2004.
- [9] FURLAN, José Davi; Assumpção Filho, Milton Mira de, editor. Modelagem de objetos através da UML - the unified modeling language. São Paulo: Makron Books: Pearson Education, c1998.
- [10] GALHARDO, Raphaela; LIMA, Gleydson. Mapeando Associações com Hibernate. Disponível em: <http://www.portaljava.com/home/modules.php?name=Content>.
- [11] GAMMA, Eric, HELM, Richard, JOHNSON, Ralph, VLISSIDES, John. Padrões de Projeto: soluções reutilizáveis de software orientado a objetos. Porto Alegre: Bookman, 2002.
- [12] H.M. DEITEL; P.J. Deitel. Java – Como programar. São Paulo. Makron Books 2000.
- [13] HTML, HyperText Markup Language. Disponível em: <http://www.w3.org/html/>.
- [14] HUSTED, Ted et al; Struts em ação. Local: Rio de Janeiro: Editora Ciência Moderna Ltda, 2004.

- [15] JOHNSON, R. E. Components, frameworks, patterns. In: ACM SIGSOFT Symposium on Software Reusability. [s.n.], 1997. p. 10–17. Disponível em: <http://citeseer.ist.psu.edu/johnson97components.html>.
- [16] JÚNIOR, Herval Freire de A. . Júnior. Mapeamento Objetos em Banco de Dados Relacionais. Disponível em: <http://www.mundooo.com.br/php/modules.php?name=MOOArtigos&pa=showpage&pid=28>.
- [17] KRAEMER, Fabiano; JARDEL VOGT, Jerônimo. Hibernate, um Robusto Framework de Persistência Objeto-Relacional. Disponível em: <http://www.guj.com.br/artigos.jsp>.
- [18] LARMAN, Craig. Utilizando UML Padrões. Uma Introdução a Análise e ao Projeto Orientado a Objetos. Porto Alegre. Bookman 2000.
- [19] LINHARES, Maurício. Introdução ao Hibernate 3. Disponível em: <http://www.guj.com.br/artigos.jsp>.
- [20] LOZANO, Fernando. Desenvolvendo aplicações Web com MVC. Java Magazine, São Paulo, v. 2, n. 20, p. 34-39, fev. 2004.
- [21] LOZANO, Fernando. Persistência Objeto-Relacional com Java. Disponível em : <http://www.lozano.eti.br/>.
- [22] MARTIN, James. Princípios de análise e projeto baseados em objetos. Tradução Cristina Bazán. Rio de Janeiro: Campus, 1994.
- [23] MYSQL, AB. Manual de referência do MySQL. Disponível em: <http://dev.mysql.com/doc/mysql/pt/>.
- [24] POMPILIO, Janaina Estigarribia. Tutorial para a linguagem UML (Unified Modeling Language). Presidente Prudente: [s.n.], 1998.
- [25] PREE, W. Design patterns for object oriented software development.
- [26] ROBERTS, D.; JOHNSON, R. E. Evolving frameworks: A pattern language for developing object-oriented frameworks. In: Pattern Languages of Program Design 3.
- [27] RODRIGUES, Analise Rosa. Um componente de Software para mapeamento objeto relacional. Disponível em [www.upf.br/erbd/index.php?option=com\\_content &task=view &id=20&Itemid=1](http://www.upf.br/erbd/index.php?option=com_content&task=view&id=20&Itemid=1)
- [28] SILVA, Ricardo P. Suporte ao desenvolvimento e uso de frameworks e componentes. Tese para a obtenção do grau de Doutor em Ciências da Computação. Universidade Federal do Rio Grande do Sul, UFRGS. Porto Alegre, 2000.
- [29] SIMON, G. Web sites pessoais. Disponível em: <http://gilbertosimon.sites.uol.com.br/>.
- [30] TILES, Framework Tiles. Disponível em: <http://tiles.apache.org>.
- [31] TODD, Nick et al; Java Server Pages: Guia do Desenvolvedor. Editora: CAMPUS, 2003.
- [32] UNIVERSIDADE FEDERAL DE SANTA CATARINA. Laboratório de Metrologia e Automação. Disponível em: <http://www.labmetro.ufsc.br>.

[33] WEBSERVICES. Informática. Disponível em: <http://www.terra.com.br/informática/ebusiness/2003/03/24/003.htm>.

[34] WESLEY Addison, 1997. Disponível em: <http://citeseer.ist.psu.edu/roberts96evolving.html>.

[35] WWW, World Wide Web. Disponível em: <http://www.w3.org/MarkUp/historical>.