UNIVERSIDADE DO OESTE DE SANTA CATARINA – UNOESC
CAMPUS DE SÃO MIGUEL DO OESTE
EZEQUIEL JULIANO MÜLLER
MARCELO JOSÉ BOTH

DELIBRIS:
SISTEMA BIBLIOTECÁRIO UTILIZANDO DEMOISELLE FRAMEWORK

São Miguel do Oeste (SC)
2010

EZEQUIEL JULIANO MÜLLER
MARCELO JOSÉ BOTH

DELIBRIS:
SISTEMA BIBLIOTECÁRIO UTILIZANDO DEMOISELLE FRAMEWORK

Monografia apresentada à Universidade do Oeste de Santa Catarina – Campus de São Miguel do Oeste como requisito parcial à obtenção do grau de Bacharel em Sistemas de Informação.

Orientador: Prof. Esp. Roberson Junior Fernandes Alves

São Miguel do Oeste (SC)
2010

Dedicamos este trabalho aos nossos pais pelo esforço, dedicação e compreensão, em todos os momentos desta e de outras caminhadas e pelo exemplo de vida e família. Aos nossos demais familiares por nos ensinarem a fazer as melhores escolhas, mostrando que honestidade e respeito são essências à vida e que nunca devemos desistir de nossos objetivos.

AGRADECIMENTOS

Agradecemos primeiramente a Deus, por criar o universo, por dar a capacidade e a inteligência às pessoas para desenvolverem e assim complementarem a criação e por iluminar nossos caminhos. Sem o universo e sem os recursos contemporâneos que o homem possui seria complicado desenvolver um projeto, enquanto dissentes e seres humanos na sagaz busca do saber.

Aos nossos pais, pois sem eles não estaríamos aqui, e por terem nos fornecido condições para que nos tornássemos profissionais e homens que somos.

Às nossas famílias que, com muito carinho e apoio não mediram esforços para que vencêssemos mais essa etapa em nossas vidas, estando sempre presentes nos momentos de dificuldade no decorrer desse curso.

Aos amigos, aos que já faziam parte de nossas vidas e àqueles que conhecemos no decorrer do curso. Amigos que estiverem presentes nos momentos de alegria e descontração e também nos momentos de dificuldade, para nos dar o apoio que tanto precisávamos.

Ao professor e orientador Roberson J. F. Alves por seu apoio e incentivo no amadurecimento de nosso conhecimento, tornando possível a execução e conclusão deste trabalho.

Aos profissionais da Serpro, pelo apoio e convite para apresentar este projeto no primeiro encontro sobre o *framework*, possibilitando a divulgação do Delibris para a comunidade *Demoiselle*.

E por último, mas não menos importante, nossos demais professores, que repassaram o conhecimento necessário para que nos tornássemos profissionais capazes e éticos.

RESUMO

Este trabalho tem por objetivo apresentar os métodos de desenvolvimento do sistema para gestão de bibliotecas Delibris, concebido com o intuito de automatizar os processos envolvidos no universo das bibliotecas e oferecer ao usuário final mais comodidade, agilidade e facilidade no acesso às informações. Para isso foi necessário analisar e compreender o funcionamento das práticas de biblioteconomia, fazendo uso do formato MARC (*Machine Readable Cataloging*) para lançamento de informações sobre o acervo bibliográfico, bem como obter conhecimentos mais sólidos sobre orientação a objetos, explorando a modalidade de *software* livre, utilizando ferramentas de desenvolvimento que possuem como ambiente operacional a *Web*, como a linguagem de programação Java, por meio do *framework* integrador *Demoiselle*. O *Demoiselle* é disponibilizado sob a licença LGPL (*Lesser General Public License*) 3 sendo desenvolvido e mantido pelo Serpro (Serviço Federal de Processamento de Dados) e se trata de uma plataforma livre de desenvolvimento de *software* do Governo Brasileiro que visa padronização, qualidade e agilidade no desenvolvimento de novas soluções. Tem como característica integrar diversas tecnologias especialistas como o padrão arquitetural MVC (*Model-View-Controller*, divisão em camadas) e ORM (*Object-Relational Mapping*, mapeamento objeto relacional) para garantir que as informações relacionais armazenadas no Sistema Gerenciador de Banco de Dados *PostgreSQL* sejam utilizadas na forma de objeto.

Palavras-chave: Delibris, *Demoiselle, Framework.*

ABSTRACT

This paper aims to present methods used for the developing system for library management called the Delibris, designed aiming automate the processes involved in the world of libraries and offer the end user, more convenience, speed and ease of access to information. For this it was necessary to analyze and understand the practices of library science, making use of the MARC (Machine Readable Cataloging) for storage of information about the collection bibliographic as well as get stronger knowledge about object orientation, exploring the modality of free software, using development tools that have the Web as operating environment, such as the Java programming language, through the integrative framework Demoiselle. The Demoiselle is available under the LGPL (Lesser General Public License) 3 is developed and maintained by Serpro (Federal Data Processing) and it is a free platform for software development of the Brazilian Government focused at standardization, quality and agility in developing new solutions. Its main feature is to integrate various specialists technologies as the architectural pattern MVC (Model-View-Controller, division to layers) and ORM (Object-Relational Mapping, Object relational mapping) to ensure that relational information that is stored in Management System PostgreSQL Database can be the used in the form of objects.

Keywords: Delibris, Demoiselle, Framework.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

AOP	Orientação a Aspectos
API	*Application Programming Interface*
CGLIB	*Code Generation Library*
CRB	Conselho Regional de Biblioteconomia
CRUD	*Create Retrieve Update and Delete*
CONSEGI	Congresso Internacional *Software* Livre e Governo Eletrônico
DAO	*Data Access Object*
DLL	*Dynamic-link-library*
GPL	*General Public License*
HTTP	*Hypertext Transfer Protocol*
IBGE	Instituto Brasileiro de Geografia e Estatística
IDE	*Integrated Development Environment*
JAAS	*Java Authentication and Authorization Service*
J2EE	*Java 2 Enterprise Edition*
JDBC	*Java Database Connectivity*
JEE	*Java Enterprise Edition*
JNDI	*Java Naming and Directory Interface*
JSF	*Java Server Faces*
JTA	*Java Transaction*
LGPL	*Lesser General Public License*
MARC	*Machine Readable Cataloging*
MUNIC	Pesquisa de Informações Básicas Municipais
MVC	*Model-View-Controller*
OO	Orientação a Objetos
OOP	Programação Orientada a Objetos
OR	Objeto Relacional
ORM	*Object Relational Mapping*

POJOS	*Plain Old Java Objects*
SERPRO	Serviço Federal de Processamento de Dados
SGDB	Sistema Gerenciador de Banco de Dados
SNMP	*Simple Network Management Protocol*
SQL	*Structured Query Language*
SVN	*Subversion*
TI	Tecnologia da Informação
UML	*Unified Modeling Language*
UNOESC	Universidade do Oeste de Santa Catarina
XML	*Extensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

O processo de informatização se tornou fundamental em qualquer ramo de atividade. Porém, atualmente, para suprir a demanda dos serviços procurados deve-se ir muito além do que a simples implementação de um aplicativo. A escolha por tecnologias especialistas que estimulem um desenvolvimento eficaz, através de conceitos de padronização, reusabilidade e persistência das informações, se torna um diferencial competitivo e aumenta as chances de se obter sucesso no universo da tecnologia da informação.

A linguagem Java é de longe a mais popular entre as linguagens de programação atuais. Isso se deve muito ao fato de ser considerada livre e pelas suas diferentes variações de plataformas. Perante isso, o governo brasileiro, por meio do Serviço Federal de Processamento de Dados (Serpro), desenvolveu um *framework* voltado para o desenvolvimento de aplicações *Web*, denominado *Demoiselle*, sendo uma analogia ao avião modular criado por Santos Dumont. Este *framework* possui como características, desenvolver *softwares* modulares, escaláveis e de forma ágil, pois muitos *frameworks* consagrados estão acoplados a ele facilitando a sua utilização.

O presente trabalho utilizou-se da linguagem de programação orientada a objetos Java por meio do *framework* integrador *Demoiselle* para o desenvolvimento de um *software* ambientado na *Web*. Sob a ótica de se agregar diversos *frameworks* especialistas e se tratar de uma opção que segue os conceitos de *software* livre, o objetivo foi utilizar o *Demoiselle* para implementar uma aplicação para o setor de biblioteconomia usufruindo dos recursos oferecidos por esta tecnologia.

O sistema desenvolvido tem como característica auxiliar no processamento técnico de catalogação do acervo bibliográfico utilizando o formato MARC (*Machine Readable Cataloging*) possibilitando desta forma o acesso e compartilhamento dos dados entre os demais sistemas que utilizam este formato. Além disso, outra característica do Delibris é oferecer formas de gerenciar o processo de circulação de materiais.

Para a definição do ciclo de desenvolvimento do projeto, optou-se por empregar uma combinação de técnicas da metodologia ágil de desenvolvimento *Scrum*, pois seu foco está nos pontos mais importantes do projeto proporcionando agilidade e dinâmica para o desenvolvimento.

No quesito de revisão bibliográfica, inicialmente, o assunto abordado é biblioteca, em seguida é apresentado um conteúdo sobre o formato MARC, e posteriormente, são apresentados *frameworks* e tecnologias que compõem o *Demoiselle* com o intuito de facilitar o entendimento a cerca do seu funcionamento.

Em virtude de o *Demoiselle* focar a padronização, reuso de código e adotar métodos de qualidade citam-se *frameworks* e Padrões de Projetos (*Design Patterns*). No que tange a persistência dos dados explica-se a ORM (*Object-Relational Mapping*) e, em seguida, cita-se o padrão arquitetural MVC (*Model-View-Controller*), para posteriormente chegar ao *framework* integrador *Demoiselle*, que é o foco principal deste trabalho.

No quesito de organização e engenharia de *software*, destaca-se a UML *(Unified Modeling Language*) que serviu para especificar, documentar e estruturar o desenvolvimento do projeto, bem como a metodologia de desenvolvimento ágil *Scrum*, pelo fato de se utilizar algumas ferramentas que a compõe para auxiliar e definir os processos ao longo da construção do *software*.

Posteriormente, é apresentada a forma de desenvolvimento, compreendendo as metodologias aplicadas e os resultados obtidos durante a execução do projeto, bem como as diretrizes para a continuidade do mesmo.

1.1 OBJETIVOS

A seguir serão descritos os objetivos que direcionam o desenvolvimento do presente trabalho.

1.1.1 Objetivo Geral

Desenvolver, através do estudo do *framework* de desenvolvimento *Demoiselle* do Governo Federal, um sistema para gestão de bibliotecas.

1.1.2 Objetivos Específicos

- Compreender o funcionamento das práticas de biblioteconomia para implantação do *software*;
- Explorar a modalidade de *software* livre;
- Adotar a metodologia ágil *Scrum* para auxiliar o desenvolvimento do projeto;
- Utilizar o paradigma orientado a objetos durante o desenvolvimento do aplicativo *Web*, através do uso de análise de projetos orientados a objetos e a linguagem de programação Java;
- Analisar e compreender o funcionamento do *Demoiselle Framework*;
- Projetar a aplicação para utilizar padrões de projetos, construção em camadas e o padrão de arquitetura MVC (*Model-View-Controller*);
- Reduzir a impedância entre o modelo orientado a objetos e o modelo relacional utilizando-se de mapeamento objeto relacional (ORM) por meio do *Hibernate*;
- Implementar os módulos de catalogação, consulta e circulação de materiais (empréstimo, reserva, renovação e devolução).

1.2 JUSTIFICATIVA/PROBLEMATIZAÇÃO

O mundo tecnológico vem sofrendo constantes mudanças. Estas buscam facilitar a resolução de problemas decorrentes do cotidiano. A cada dia fica mais explícita a necessidade de se informatizar os mais diversos procedimentos, assim oferecendo serviços mais qualificados, como observa Laurindo *et al.* (2001) "Há uma grande expectativa acerca das aplicações da Tecnologia da Informação (TI), que possibilitam novas alternativas de estratégias de negócios e novas possibilidades para as organizações [...]".

Em virtude disso, o Governo Federal, por meio do Serviço Federal de Processamento de Dados (Serpro), desenvolveu um *framework* a fim de prover uma plataforma de desenvolvimento livre e voltada para construção de aplicações robustas, escaláveis e que utilizam a *Web* como ambiente operacional. Este *framework* foi denominado *Demoiselle* em homenagem a Santos Dumont, sendo concebido no conceito de *framework* integrador. Brasil (2009c) argumenta que "Seu objetivo é facilitar a construção de aplicações sem o investimento de tempo em escolha e integração de *frameworks* especialistas, o que resulta no aumento da produtividade e garante a manutenibilidade dos sistemas."

Além do *framework* se tratar de uma boa ferramenta para desenvolvimento integrando tecnologias especialistas e seguindo um padrão nacional para o desenvolvimento de aplicações públicas, pretende-se utilizá-lo na construção de um sistema de gestão de bibliotecas de caráter público. Diante isso, busca-se desenvolver um sistema capaz de automatizar os processos envolvidos no universo das bibliotecas e oferecer ao usuário final mais comodidade, agilidade e facilidade no acesso as informações.

A preocupação com a automatização desses processos cresce na mesma proporção em que surgem novas bibliotecas. Nos últimos anos, tem aumentado a preocupação dos municípios brasileiros em oferecer um maior acesso a este tipo de cultura, como aponta a Pesquisa de Informações Básicas Municipais (MUNIC) de 2005 realizado pelo Instituto Brasileiro de Geografia e Estatística (IBGE), Brasil (2005a) estabelece que "[...] as bibliotecas públicas são os equipamentos mais presentes nos municípios brasileiros. São 6.545 bibliotecas localizadas em 4.726 municípios (85% do total), com uma relação de 1,2 bibliotecas por município."

A Biblioteca Pública de São Miguel do Oeste, preocupada em possuir um maior controle de seu acervo bem como oferecer um serviço de qualidade aos seus leitores, busca automatizar os processos bibliotecários. Hoje a mesma dispõe de um bom acervo de livros (aproximadamente 30 mil exemplares), mas o controle sobre esse acervo é feito de forma manual o que acarreta uma série de problemas que compromete as práticas de biblioteconomia.

Frente a essas necessidades, o que se pretende é oferecer, inicialmente, um sistema que explore a modalidade de *software* livre e que seja capaz de automatizar os processos de catalogação, consulta e circulação de materiais (empréstimo, reserva, renovação e devolução) do acervo bibliotecário, proporcionando assim uma melhoria na qualidade dos serviços e não desperdiçando tanto material impresso no armazenamento dos dados, além de diversificar os serviços providos ao usuário final. Pretende-se, com o presente estudo, explorar ao máximo as potencialidades do *Demoiselle Framework*, sendo assim, para todo o processo de desenvolvimento será levado em consideração, seus objetivos base, tais como a padronização, reuso de código e métodos, baixo acoplamento, divisão em camadas, componentização e padrões de projetos.

2 REVISÃO DA LITERATURA

A presente revisão literária citará bibliotecas, abordando suas definições e processos de informatização. Também é apresentado um conteúdo sobre o formato MARC e posteriormente cita-se *software* livre, fazendo um comparativo com o movimento *open source*. Em seguida, são abordados *frameworks* e padrões de projetos (*Design Patterns*) que auxiliam no desenvolvimento e na reutilização de código e estruturas. Ao final é citada a linguagem de modelagem UML e a metodologia ágil *Scrum*.

A revisão literária tem como objetivo fundamentar alguns *frameworks* e tecnologias que compõe o *Demoiselle*, neste sentido, também é abordado o mapeamento objeto/relacional (ORM), a ferramenta para realizar este mapeamento o *Hibernate* e o padrão arquitetural MVC (*Model-View-Controller*).

2.1 BIBLIOTECAS

Inicialmente será abordado o assunto bibliotecas, em que se apresenta uma definição e os processos de informatização que as bibliotecas púbicas sofrem ao longo dos últimos anos.

2.1.1 Tipos de Bibliotecas

Uma correta definição para biblioteca é a grande coleção de arquivos onde estão presentes desde livros até mídias digitais ou acervos culturais diversos. Estes possuem a finalidade de propiciar o acesso a informação, pesquisa e desenvolvimento humano (CRUZ; MENDES; WEITZEL, 2004).

Sobre uma definição e os tipos de bibliotecas Pinho e Machado (2003) estabelece:

> A biblioteca, ou seu sentido, refere-se também à grande variedade de coleções bibliográficas e aos diferentes fins e usuários. A maioria das nações desenvolvidas dispõe de bibliotecas de vários tipos: nacionais, universitárias, públicas, escolares e especializadas. Quase sempre, estão interligadas nacionalmente e, por meio de associações profissionais e de acordos estabelecidos, desenvolvem programas de cooperação e intercâmbio extensivos a outros países. Alem dessas, existem as inúmeras particulares, que se tornaram objeto de estudo histórico, devido ao grau de importância dado à leitura e para o historiador tomar conhecimento sobre o que se lia em determinada época.

Existem outras definições, como bibliotecas de conservação, sendo assim designadas por armazenar documentos raros, que muitas vezes podem ser acessíveis apenas por especialistas e bibliotecas de consumo que são as públicas, abertas e acessíveis a todos os leitores (PINHO; MACHADO, 2009).

2.1.2 Bibliotecas Públicas e a Informatização

Em uma definição sobre biblioteca pública, Cruz, Mendes, Weitzel (2004, p.11) estabelecem que: “Biblioteca pública é a que tem por finalidade servir as massas, [...] atende [...] uma coletividade e, principalmente, uma coletividade local ou regional, destinando-se ao público em geral ou determinadas categorias”.

As bibliotecas no Brasil, ao longo dos últimos anos tem sido alvo de pesquisas e tentativas de incentivos para uma solidificação da leitura e procura de conhecimento. Ainda assim as bibliotecas públicas no Brasil convivem com vários empecilhos, tais como: falta de modernização, principalmente com *softwares* de gestão, estrutura computacional e acesso a Internet, também existe a falta de acervo adequado, processos de atualização e auxilio com recursos para manutenção dos materiais ou estruturas existentes (OLIVEIRA, 2007).

Sobre a modernização e informatização das bibliotecas Côrte e outros (1999) argumenta:

> A modernização das bibliotecas está diretamente ligada à automação de rotinas e serviços, com o intuito de implantar uma infra-estrutura de comunicação para agilizar e ampliar o acesso à informação pelo usuário, tornando-se necessário haver uma ampla visão da tecnologia da informação e sua aplicação nas organizações.

As bibliotecas sofrem diariamente interferência em seus processos de trabalho, o que torna imprescindível a adequação a um processo de informatização. Os avanços tecnológicos dos últimos anos junto às exigências atuais dos usuários fazem necessária a aquisição tanto de *hardware* como de *softwares* para interligar as funções das bibliotecas, permitindo interagir homem/máquina (CÔRTE *et al* 1999).

2.2 FORMATO MARC

O formato MARC tem como objetivo oferecer um padrão para o armazenamento de informações de registros bibliográficos. Segundo Ribeiro e Júnior (2010) “Estas informações bibliográficas incluem: títulos, nomes, assuntos, notas, dados de publicação, e informação sobre a descrição física de um item, etc.”

A estrutura de um registro bibliográfico no formato MARC é composta de três componentes: Líder, Diretório e Campos Variáveis. O Líder contém as informações básicas para o processamento do registro. O Diretório apresenta as entradas que contém a identificação do campo. Cada entrada possui três elementos de dados: parágrafo, tamanho, e posição inicial. Os Campos Variáveis contém as informações sobre o registro, tais como, autor, título, ano etc. (RIBEIRO; JUNIOR, 2010).

O Delibris tem como característica a utilização do formato MARC para o lançamento e armazenamento de informações sobre o acervo bibliográfico. As entradas de informação acerca de um acervo são denominadas parágrafos. Neste sentido um acervo é composto de diversos parágrafos que armazenam dados tais como, autor, título, edição, assunto etc.

2.3 SOFTWARE LIVRE

Neste momento o trabalho tem seu foco voltado para as definições e principais características dos *softwares* livres no desenvolvimento e distribuição de aplicações.

2.3.1 Características de um Software Livre

A utilização de *software* livre vem se difundindo de forma considerável em todo o mundo (GUTIERREZ; ALEXANDRE, 2004). Hoje não é muito viável ficar refém de uma tecnologia em que os custos de implantação e a manutenção são altos. Seguindo a idéia de *softwares* livres é possível diminuir estes gastos e contribuir de uma forma mais sustentável ao desenvolvimento tecnológico, combatendo a privatização do saber (BRASIL, 2009b).

A Free Software Foundation (2009) define *software* livre como qualquer programa computacional que pode ser usado, copiado, estudado, modificado e redistribuído sem nenhuma restrição. Para tanto, este programa deve atender a quatro tipos de liberdades, descritos em seu site oficial:

- Liberdade de executar o *software* para qualquer finalidade ou propósito;
- Liberdade de estudar o *software* e adaptá-lo as suas próprias necessidades (Acesso ao código fonte é pré-requisito para esta liberdade);
- Liberdade de redistribuir cópias do *software* para que outros usuários possam se beneficiar;
- Liberdade de aprimorar o *software* e publicar estes aprimoramentos, de modo que toda comunidade possa se beneficiar com os mesmos (acesso ao código fonte é pré-requisito para esta liberdade);

Ao atender estas liberdades você está livre para redistribuir cópias, seja com ou sem modificações, seja de graça ou cobrando algum valor, para qualquer pessoa, empresa ou

instituição. Neste sentido o ser livre significa que você não tem que pedir ou pagar pela permissão de utilizar e redistribuir as soluções (FREE SOFTWARE FOUNDATION, 2009).

Mesmo partindo da idéia de compartilhamento e livre distribuição é importante lembrar que "*software* livre" não significa "não-comercial". Sobre isso Nunes (2003) estabelece:

> A aplicação comercial do Software Livre não é incomum e não contradiz nenhuma das quatro liberdades fundamentais estabelecidas como indispensáveis para que um software seja considerado livre. Embora muitos pensem que o Software Livre não pode ser vendido, ou desenvolvido para fins comerciais, esse é um dos muitos mitos que rodeiam o paradigma da Liberdade de Software proposta pela Free Software Foundation.

O que acontece é que nos acostumamos em atrelar a idéia de *free software* com "grátis", ou livre de custos financeiros. Neste sentido, às vezes, fica difícil de ver um *software* livre voltado para fins comerciais (NUNES, 2003).

2.3.2 Software Livre versus Open Source

É importante não confundir *software* livre com *software Open Source* (fonte-aberta). A iniciativa *Open Source* tem sua origem baseada na licença GPL, mas possui particularidades e segundo Gutierrez e Alexandre (2004), para um *software* se enquadrar nesta qualificação deve respeitar os seguintes requisitos:

- Deve haver uma livre redistribuição;
- O código-fonte deve ser aberto;
- As modificações e trabalhos derivados podem ser redistribuídos nos termos da licença do *software* original;
- Deve ser garantida a integridade do código-fonte do autor;
- Não pode haver discriminação quanto à pessoa, grupos ou usos;
- A licença não pode contaminar outros *softwares* distribuídos em conjunto;
- O licenciamento deve ser neutro quanto à tecnologia utilizada;

De forma geral, o uso de *software* livre fomenta a idéia de compartilhamento de conhecimento, fazendo assim, uma disseminação e uso de soluções eficientes que estimulem a economia e o setor de prestação de serviços (NUNES, 2003).

2.4 FRAMEWORKS

Agora serão discutidas definições, objetivos e utilização de *frameworks* no desenvolvimento de aplicações.

2.4.1 Trabalhando com Frameworks

Quando a reusabilidade se torna eminente no desenvolvimento de aplicações é preciso se habituar e trabalhar com a utilização de *frameworks*, segundo Braude (2005, p. 567) “Criamos *frameworks* porque queremos reutilizar grupos de classes e algoritmos entre eles”. Quando se fala em *frameworks* surge a idéia de conjunto de classes. Macias, (2008) conceitua *framework* “como sendo um projeto formado por um conjunto de classes que cooperam entre si e é reutilizável por um domínio de software determinado.” Macias, (2008) argumenta que *frameworks* ou “arcabouços” são “uma estrutura muito importante na criação de arquiteturas de software de aplicações corporativas.”

É importante diferenciar *frameworks* de *design patterns* (padrões de projetos), os *frameworks* possuem como característica serem mais concretos e de mais alto nível, geralmente são utilizados para domínios ou tecnologias específicas, um *framework* pode englobar vários *design patterns*. Já *design patterns* são dispostos em uma forma mais conceitual e podem ser aplicados em uma gama maior de problemas podendo ser usado em praticamente qualquer aplicação (JOHNSON, 2003).

Como os *frameworks* possuem características de serem reutilizáveis extensíveis e com funcionalidades abstratas que podem ser completadas, o tempo de desenvolvimento é reduzido de forma considerável e permite que problemas mais difíceis se resolvam com base nas melhores práticas já empregadas por eles (JOHNSON, 2003).

2.4.2 Objetivos dos Frameworks

Quando se utiliza *frameworks* tem se o objetivo de padronizar o projeto e o desenvolvimento. Esses parâmetros definem como é realizada a divisão entre classes e objetos, bem como suas responsabilidades, como cooperam entre si e seu fluxo de informação ditando assim a arquitetura da aplicação que os utilizam (BRASIL, 2009c).

Segundo Brasil (2009c) os objetivos que se buscam por meio da utilização de *frameworks* em suma são:

- Padronização;
- Redução da curva de aprendizagem;
- Aumento da produtividade;
- Simplificação do processo;
- Reutilização de artefatos;
- Manutenção simplificada.

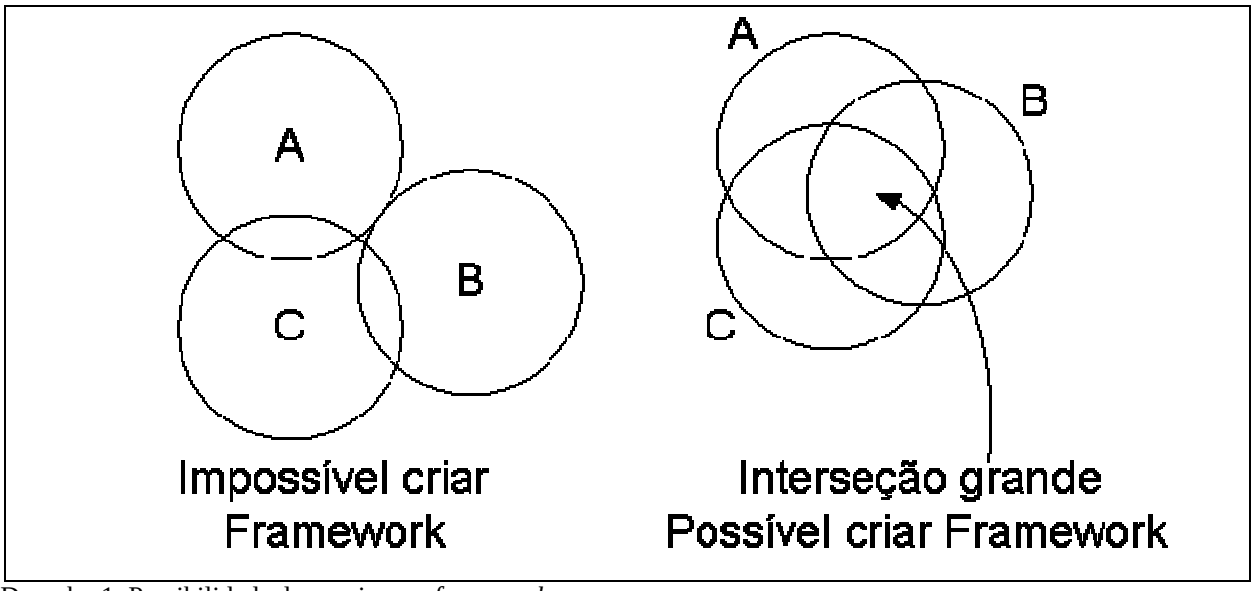
Por meio de toda esta padronização ocorre um fraco acoplamento permitindo um desenvolvimento orientado a componentes, possibilitando uma fácil manutenção, pois podem ser modificados ou acoplados novos módulos sem que o núcleo central seja alterado. Desta maneira, tem-se uma maior persistência e, segundo Braude (2005, p. 569), um "[...] dos objetivos comuns da introdução de *frameworks* é gerenciar a *persistência*".

2.4.3 Porque se utilizar Frameworks

A possibilidade de se utilizar funcionalidades comuns entre diferentes aplicativos caracterizam a criação ou a utilização de *frameworks*. Com a sua utilização é possível se beneficiar do conhecimento aplicado no desenvolvimento de *softwares* que compartilham uma família de problemas comuns entre eles (BRAUDE, 2005). Sobre isso Braude (2005, p. 567)

também afirma “Em geral, um framework começa a surgir no momento em que uma organização de desenvolvimento produz sua segunda, terceira ou quarta aplicação”.

Basicamente a utilização de *frameworks* visa buscar alternativas compatíveis em um conjunto de problemas, como é demonstrado no desenho 1. Para decompor estes problemas e possibilitar que os mesmos trabalhem em sintonia é utilizado um conjunto de classes e interfaces. Este conjunto de classes deve ser flexível e extensível, pois desta maneira é possível construir aplicações especificando apenas as particularidades de cada uma tendo um escopo central compatível entre as mesmas (SAUVÉ, 2009).



Desenho 1: Possibilidade de se criar um *framework*
Fonte: Sauvé (2009).

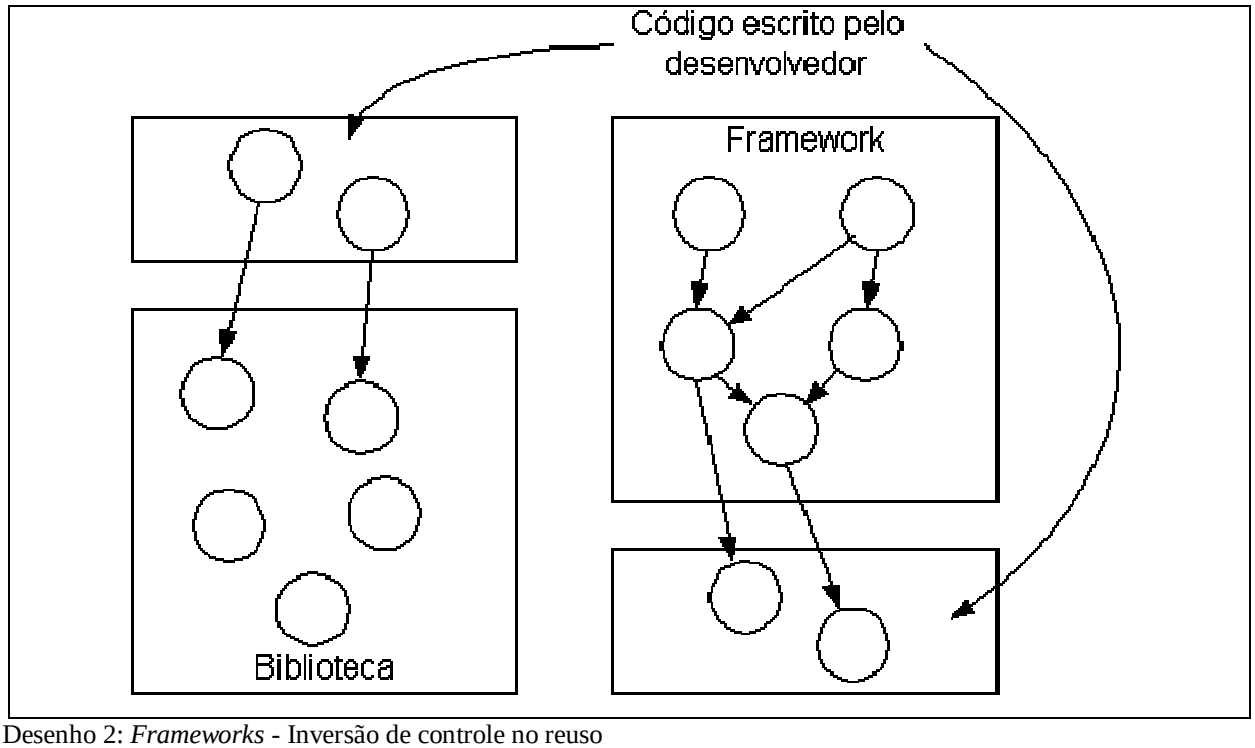
Sobre este conjunto de classes Braude (2005, p. 567) estabelece:

> As classes dentro de um *framework* podem ser relacionadas. Podem ser abstratas ou concretas. As aplicações podem utilizá-las por meio de herança, agregação ou dependência: alternativamente, um *framework* pode comportar-se como uma aplicação genérica que personalizamos inserindo nossas próprias partes. Os artefatos do *framework* estão em geral na forma pronta para usar e são códigos nativos.

Quando se pensa na utilização de *frameworks* tem-se o objetivo de ter mais agilidade e qualidade no processo de desenvolvimento, Brasil (2009c) afirma:

> A necessidade de construir *software* de forma cada vez mais ágil e a exigência da criação de produtos com mais qualidade fazem com que o processo de desenvolvimento de *software* livre seja apoiado pelo reuso de estruturas pré-existentes, por exemplo, *frameworks*. O principal propósito de um *framework* é ajudar no processo de desenvolvimento de aplicações. Ele permite que as aplicações sejam desenvolvidas mais rapidamente e mais facilmente, e deve resultar em uma aplicação de qualidade superior.

Quando se tem uma reutilização em nível de projeto através de *frameworks* ocorre uma inversão do código específico de aplicação e o *software* no qual ela se baseia. Nas aplicações que utilizam DLL (*Dynamic-link library*) o código está concentrado na aplicação e faz as chamadas dos trechos que deseja utilizar. Já no caso da utilização de *frameworks*, o trecho reutilizável está nele que chama o código que foi escrito pelo desenvolvedor (MACIAS, 2008), como pode ser observado no desenho 2.



Desenho 2: *Frameworks* - Inversão de controle no reuso
Fonte: Sauvé (2009).

Ao finalizar a abordagem sobre *frameworks* se pode observar os princípios e conceitos que fazem parte do seu desenvolvimento, bem como os reais objetivos que cercam sua utilização.

2.5 PADRÕES DE PROJETOS (DESIGN PATTERNS)

Nesta seção é exposto o conteúdo sobre *design patterns*, explicam-se algumas características, bem como os benefícios que cercam a sua utilização.

2.5.1 Características

Ao longo do desenvolvimento de *software*, desenvolvedores constantemente deparam-se com vários tipos de problemas durante a codificação. Quando se fala em desenvolvimento orientado a objetos estes problemas são bastante evidentes e comuns. Para evitar problemas constantes e semelhantes, foram criados alguns padrões, definidos como padrões de projetos ou *design patterns*. Os padrões de projetos servem para solucionar problemas que são comuns em linguagens orientadas a objetos, agilizando o desenvolvimento e criando um padrão de desenvolvimento (ZEMEL, 2009a). Braude (2009, p.158) define padrões de projetos como: "[...] uma combinação de classes e algoritmos associados que cumprem com propósitos comuns de projeto". Ainda de acordo com Braude (2009), os padrões de projetos são classificados em tipos, onde variam entre padrões criacionais, estruturais e comportamentais.

Os padrões de projetos que são caracterizados como criacionais trabalham com a instanciação dos objetos, tornando a aplicação mais flexível, pois possibilitam que o *software* seja independente da forma como são instanciados os objetos, delegando esta responsabilidade ao padrão empregado (DESTRO, 2004).

Os padrões de projetos estruturais segundo Braude (2009, p. 242), devem "[...] representar objetos complexos (o ponto de vista estático) e obter funcionalidades a partir deles de maneira a utilizar seus objetos agregados (o ponto de vista dinâmico)". Ao que diz respeito aos padrões de projetos comportamentais, estes podem ser caracterizados por serem responsáveis por captar o comportamento dos objetos durante a utilização das funcionalidades estabelecidas (BRAUDE, 2009).

2.5.2 Benefícios ao se Adotar Padrões de Projetos

As utilizações dos padrões de projetos trazem vários benefícios. Moralez (2006) cita alguns dos benefícios visíveis quando utilizado os *design patterns*:

- A utilização de *design patterns* força uma forma otimizada e clara de comunicação entre desenvolvedores, documentação e maiores possibilidades de exploração para alternativas de soluções para o projeto;
- Melhora a qualidade geral do programa, pois reduz a complexidade do código oferecendo uma abstração das classes e instâncias;
- Redução do tempo de aprendizado de novas bibliotecas ou funções;

A padronização facilita o trabalho de programadores e equipes. Com os padrões de projetos, os trabalhos repetitivos e custosos podem ser agilizados.

2.6 OBJECT-RELATIONAL MAPPING (ORM)

Neste momento será abordado o conteúdo de ORM (Mapeamento Objeto/Relacional), mostrando as suas principais definições e posteriormente a ferramenta de mapeamento *Hibernate*.

2.6.1 ORM – Definições, Utilização e Benefícios

Com o passar do tempo começaram a surgir os bancos de dados orientados a objetos, mas estes não tiveram a mesma difusão das linguagens que utilizam este paradigma, assim começou a surgir a necessidade de se utilizar ferramentas capazes de suprir a junção do mundo orientado a objetos ao mundo relacional (GALANTE; MOREIRA; BRANDÃO, 2005). Essas são a

chamadas ferramentas de mapeamento objeto/relacional (O/R) e segundo Galante, Moreira e Brandão (2005) "[...] nada mais são do que um 'tradutor' entre duas linguagens diferentes."

Houve muitas tentativas para fazer com que estas tecnologias se interligassem, ou até mesmo a possibilidade de se substituir uma pela outra. Mas o abismo que há entre ambas é um fator determinante no mundo da computação corporativa. Hoje em dia, o fornecimento de uma ponte entre paradigmas orientados a objetos e dados relacionais é o principal desafio do mapeamento objeto/relacional (ORM, *Object-Relational Mapping*) (BAUER; KING, 2005).

Sobre o que é mapeamento objeto/relacional Bauer e King (2005, p.31) afirmam:

> [...] é a persistência de objetos automatizando (e transparente) dentro de um aplicativo Java para as tabelas em um banco de dados relacional, usando metadados que descrevem o mapeamento entre objetos e o banco de dados. O ORM, essencialmente, trabalha transformando dados (de modo reversível) de uma representação em outra.

O mapeamento objeto/relacional se faz necessário devido ao fato do paradigma que os aplicativos orientados a objetos são construídos, pois, as informações são armazenadas na forma tabular e são apresentados como objetos (GONÇALVES, 2007).

Toda a interação da aplicação com a camada de dados é gerenciada pelo ORM que é incumbido de gerar todo código SQL (*Structured Query Language*) necessário. Sam-Bodden (2006, p. 86) argumenta, "Os desenvolvedores trabalham no nível de objetos e o conceito de consultas e transações são aplicados aos objetos, em vez de nos objetos do banco de dados".

Durante muito tempo a persistência de dados foi um assunto extremamente debatido entre os desenvolvedores Java. Buscam-se alternativas que automatizassem funções nos bancos de dados, mas o impacto das mesmas, por exemplo, na troca de um banco de dados, era muito grande. A portabilidade era um objetivo distante. A ORM vem para suprir estas necessidades facilitando a construção de aplicações que exigem um grande nível de portabilidade (GONÇALVES, 2007).

A solução do ORM consiste em um API (*Application Programming Interface*) que executa operações CRUD (*Create, Retrieve, Update and Delete*) em objetos de classes persistentes, na linguagem ou API que é incumbida de referenciar as classes bem como suas propriedades, em um recurso que faz o mapeamento dos metadados e uma técnica que interagem

objetos transacionais, verificando associações e otimizando funções e processos servindo como uma ponte entre os dados da aplicação (BAUER; KING, 2005).

Os benefícios quanto à utilização de ORM podem ser percebidos na: produtividade, pois não importa qual a estratégia usada no desenvolvimento da aplicação o ORM se encarregará da persistência dos dados; na manutenção, pois se diminui as linhas de código e se foca mais na regras de negócio e menos nas conexões; no desempenho, pois permite utilizar otimização o tempo todo; e na independência de fornecedor, pois ele abstrai o aplicativo do banco de dados SQL do dialeto SQL empregado (BAUER; KING, 2005).

2.6.2 Hibernate

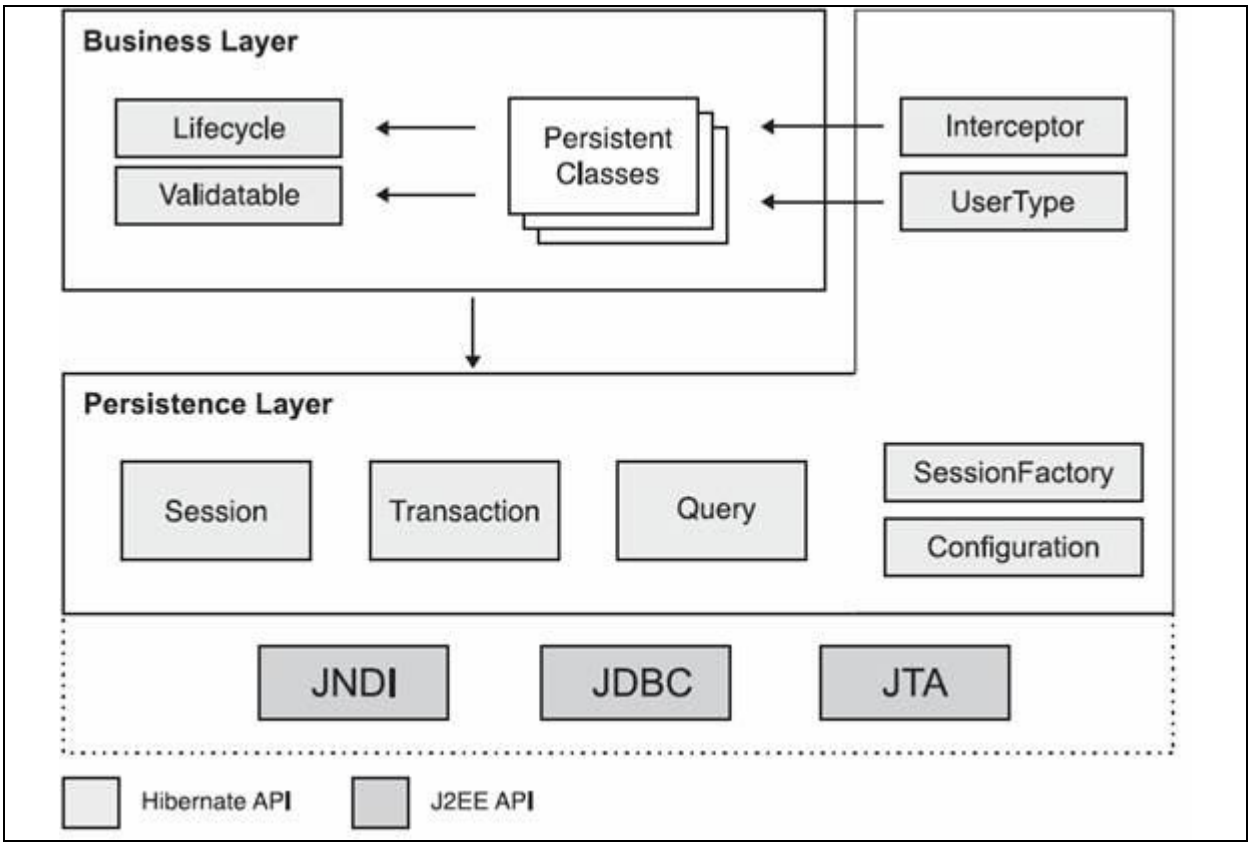
Uma das grandes vantagens de se utilizar Java no desenvolvimento de aplicações é fato de não trabalhar diretamente com o banco de dados. Atentos a este fato desenvolvedores do mundo todo começaram a desenvolver ferramentas capazes de realizar um mapeamento e acesso ao banco de dados, dentre as ferramentas existentes hoje o de mais destaque é o *Hibernate*. (GONÇALVES, 2007).

O grande sucesso desta ferramenta está na simplicidade de seus conceitos básicos. O *Hibernate* se encarrega de fornecer persistência e ser estável na consulta de objetos. A forma como é implementado baseia-se em programação declarativa orientada a objetos, não dependendo de geração de código ou modificação de *bytecode* (SAM-BODDEN, 2006).

2.6.2.1 Interfaces do Hibernate

Sobre as interfaces que são observadas no desenho 3, O *Hibernate* possui algumas que são chamadas e executam o CRUD (*Create, Retrieve, Update and Delete*) básico, elas incluem *Session, Transaction e Query,* outras Interfaces do tipo *callback* que reagem aos eventos

Interceptor, Lifecycle e Validatable e interfaces com a funcionalidade de mapeamento como o *UserType.* Além de já fazer uso de API's Java existentes como o JDBC (*Java Database Connectivity*), JTA (*Java Transaction*) e JNDI (*Java Naming and Directory Interface*) (BAUER; KING, 2005).



Desenho 3: Visão geral de alto nível da API do *Hibernate*
Fonte: Bauer e King (2005, p.51).

O objetivo da utilização desta ferramenta é facilitar a manipulação de dados, inibindo o desperdício de tempo com tarefas primorosas de banco de dados. Um dos grandes benefícios do *Hibernate* é o fato de ele deixar o desenvolvedor livre para focar sua atenção em problemas de lógica de negócios. Desta maneira, torna a interação com o banco de dados relacional menos complicada (GONÇALVES, 2007).

Sobre esta facilidade na manipulação dos dados Gonçalves (2007, p.512) ainda destaca:

> O Hibernate é um framework que se relaciona com o banco de dados, onde esse relacionamento é conhecido como mapeamento objeto/relacional para Java, deixando o desenvolvedor livre para se concentrar em problemas de lógica de negócio. Sua simplicidade de configuração, dá ao desenvolvedor algumas regras para que sejam seguidas como padrões de desenvolvimento ao escrever sua lógica de negócios e suas classes persistentes. [...] Uma mudança de banco de dados, nesse caso, não se torna traumática, alterando apenas um ou outro detalhe nas configurações do Hibernate.

O *Hibernate* se encarrega de evitar problemas de violações de restrições e chaves estrangeiras e pode ser integrado em quase todo ambiente Java, através de API's adicionais que interligam as diferenças nos ambientes e permitem que o código de persistência se mantenha portável. Apesar de abranger uma grande quantidade de cenários a sua utilização busca suprir as necessidades das tarefas com o banco de dados mantendo-se compreensível para o desenvolvedor que fica mais focado nas regras de negócio da aplicação (BAUER; KING, 2005).

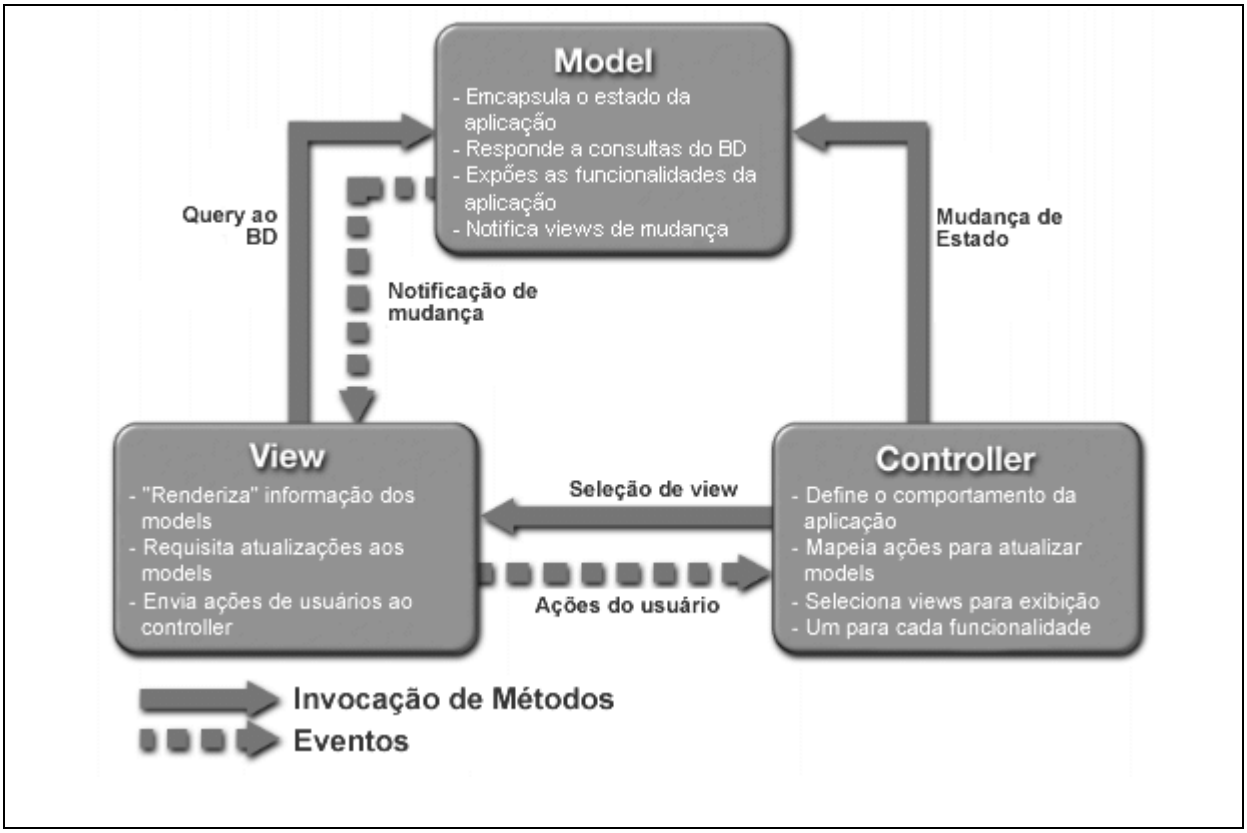
2.7 MVC – MODELO, VISÃO E CONTROLE

Agora será apresentado o padrão arquitetural MVC (*Model-View-Controller*), o foco é oferecer um embasamento teórico sobre as camadas deste modelo.

2.7.1 As Camadas do MVC

MVC é um padrão arquitetural que separa uma aplicação em várias camadas. Devido ao fato das aplicações terem aumentado a sua complexidade no desenvolvimento, é indispensável que ocorra esta divisão. Desta forma quando é feita uma alteração em uma camada, esta não afeta o funcionamento da outra. O MVC define que cada camada realiza um tipo de tarefa na aplicação, sendo dividas em *Model* (Modelo), *View* (Visão) e *Controller* (Controle) (ZEMEL, 2009b).

Como pode ser observado no esquema 1, a camada denominada modelo tem a responsabilidade de manipular os dados e fazer a aplicação das modificações que são solicitadas pela camada de controle. A camada de controle gerencia as operações que são realizadas no sistema tendo um relacionamento com a camada de visão e modelo. Neste sentido, a camada de controle tem como responsabilidade receber as solicitações que são enviadas pela camada de visão e fazer um gerenciamento das operações que provêm da camada de modelo, o contrário também acontece depois de realizado as operações na camada de modelo, o controle retorna as alterações ou as mensagens para a camada de visão. E finalmente a camada de visão que faz uma representação visual dos dados que provêm da camada de modelo e são administrados pelo controle (GONÇALVES, 2007).



Esquema 1: As camadas do MVC
Fonte: Zemel (2009).

Segundo Rinaldi (2009), sistemas baseados em camadas, possuem como características "[...] a facilidade de manutenção, redução de código e maior acessibilidade".

Este padrão tornou-se muito importante nos últimos anos no desenvolvimento de aplicações, em especial as aplicações *Web*. É fortemente empregado em linguagens como Java e C#, sendo a linguagem Java a principal difusora com os seus diversos *frameworks*.

2.8 DEMOISELLE FRAMEWORK

Agora é exposto o foco principal do trabalho: o *Demoiselle Framework*. As suas principais características, bem como a arquitetura proposta, são conteúdos abordados.

2.8.1 Características do Demoiselle Framework

O Governo Federal, por meio do Serviço Federal de Processamento de Dados (Serpro), desenvolveu um *framework* visando uma maior padronização e reuso de aplicações em sistemas utilizados em órgãos federais. Esta plataforma foi denominada *Demoiselle* e segundo Lisboa (2009a) "[...] é essencialmente uma biblioteca central de módulos que atende às necessidades de infra-estrutura básica de uma aplicação web não distribuída." Sobre as principais características do *framework* Lisboa (2009a) analisa "A adoção do Demoiselle pretende automatizar e acelerar a integração de sistemas, aumentar a produtividade e eliminar o retrabalho."

Brasil (2009c) argumenta sobre o *Demoiselle:* "[...] é uma ferramenta de código-aberto e totalmente livre, que visa garantir a interoperabilidade e facilidade de manutenção dos sistemas dos diferentes ministérios e autarquias do governo federal." É inicialmente focado para o desenvolvimento de aplicações *Web* (J2EE - *Java 2 Enterprise Edition*). Disponibilizado sob a licença LGPL (*Lesser General Public License*) 3, trata-se de um conjunto de classes que cooperam entre si tornando mais simples a padronização e reuso de aplicações. Seguindo neste sentido a construção de aplicativos é mais eficaz, o tempo gasto com a integração entre

frameworks torna-se inexistente deixando os desenvolvedores, analistas e gerentes de projetos focados exclusivamente na construção do *software* (BRASIL, 2009c).

O uso de *software* livre para auxiliar em questões técnicas, econômicas e de segurança nas esferas governamentais tem se tornado algo inevitável com o tempo. Hoje não é muito viável ficar refém de uma tecnologia, onde os custos de implantação e manutenção são altos, seguindo a idéia do *framework Demoiselle* há uma maior democratização no processo de desenvolvimento de sistemas diminuindo custos e tornando os processos mais flexíveis. (BRASIL, 2009c).

É assim denominado, em homenagem a um modelo de avião idealizado por Santos Dumont, segundo Brasil (2009c) "[...] Santos Dumont permitia a utilização, adaptação e cópia de seu trabalho." Em função deste pensamento que segue os conceitos atuais do *software* livre, *Demoiselle* foi o nome mais apropriado para batizar este *framework*. É importante destacar, que mesmo sendo concebido inicialmente para aplicações do governo, esta plataforma não se limita apenas a isso podendo ser usada livremente por qualquer entidade ou pessoa.

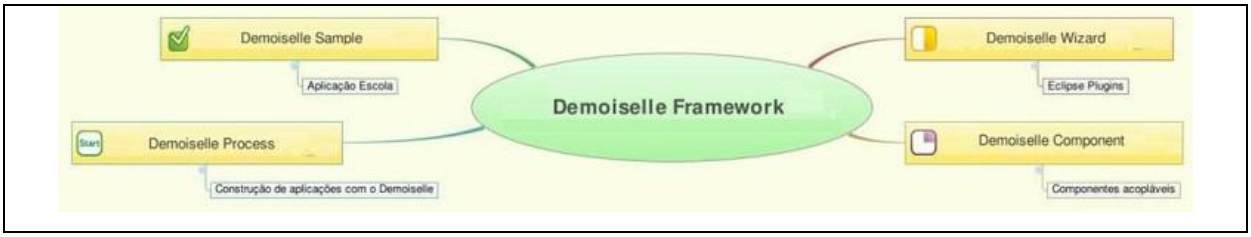
Devido ao fato de o *framework* estar registrado como *software* livre ele será mantido em comunidade, totalmente compartilhado, permitindo que pessoas e entidades possam contribuir e ser beneficiadas pelo reuso de códigos. O principal objetivo do *Demoiselle* é estabelecer uma plataforma extensível e ter sua documentação publicada (LISBOA, 2009a).

Um grande benefício que se tem utilizando esta plataforma de desenvolvimento é a economia financeira, pois não há necessidade de se gastar com licenças de *softwares*. Além disso, ele possui um acoplamento fraco, orientado a componentes, possibilitando assim que qualquer desenvolvedor o customize de maneira a atender seus propósitos (LISBOA, 2009a).

2.8.2 O Projeto Demoiselle Framework

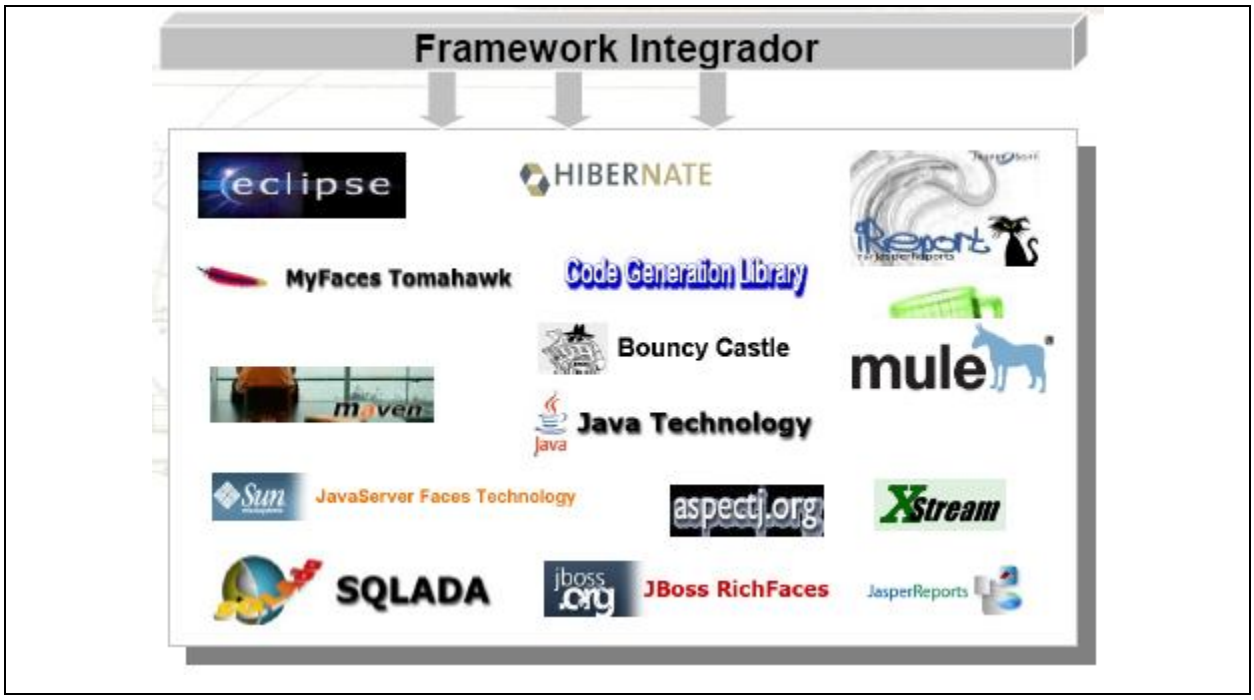
Como pode ser observado no esquema 2, o *Demoiselle* é divido em subprojetos, sendo que cada um possui um ciclo de vida independente. O *Demoiselle Wizard* se encarrega de gerar códigos automáticos baseados em modelos Java, o *Demoiselle Sample* trata-se de uma aplicação exemplo que pode ser estudada, o *Demoiselle Component* é um pacote que serve para

desenvolver componentes e o *Demoiselle Process* tem seu foco voltado para a sugestão de processos para construção de aplicações (BRASIL, 2009c).



Esquema 2: Subprojetos do *Demoiselle Framework*
Fonte: Brasil (2009c).

O *Demoiselle* é engajado na idéia de *Framework* Integrador, segundo Macias (2008) um *Framework* Integrador "[...] realiza a integração entre vários frameworks especialistas e garante a evolução, manutenibilidade e a compatibilidade entre cada um deles." Como pode ser observado no desenho 4 o *Demoiselle* engloba os seguintes *frameworks*: Eclipse – IDE desenvolvimento; *Jasper* e *iReport* – relatórios; *Hibernate* – serviço de persistência OO/Relacional; *Sun JSF* – *Tomahawk*, *RichFaces*; *Bouncy Castle* – criptografia; *AspectJ* – *plugin* Eclipse para Aspectos; *JBoss* – servidor de aplicação JEE; *Mule* – *Enterprise Service Bus; Xstream* – manuseio de XML; CGLIB – segurança; *Maven* – documentação automatizada; *Log4J* – geração de *logs*; SQLADA – conector *micromainframe*.



Desenho 4: *Demoiselle*: *Framework* Integrador
Fonte: Brasil (2009d).

Desta maneira, o desenvolvimento de aplicações se torna mais eficaz, pois o *Demoiselle* já agrega tecnologias que suprem as necessidades de implementação de uma aplicação.

2.8.3 Arquitetura do Demoiselle Framework

A arquitetura do *Demoiselle Framework* é divida em módulos como pode ser observado no esquema 3. O módulo *Core* possibilita fazer padronização, extensão e integração de camadas. O módulo *Persistence* é responsável pelo armazenamento e tratamento das informações. O módulo *Util* possui componentes que facilitam o trabalho do *framework*. O módulo *View* é responsável pela interface com o usuário. E finalmente o módulo *Web* que tem como responsabilidade prover o tratamento de sessões e requisições do usuário (LISBOA, 2009a).

Esquema 3: Arquitetura do *Demoiselle Framework*
Fonte: Brasil (2009d).

O *Demoiselle* foca a idéia de reaproveitamento e construção de *software* modular e escalável. O *framework* separa as suas principais características em camadas onde é possível separar responsabilidades e dar maior manutenibilidade ao código, sendo assim, há uma grande diminuição no tempo de aprendizagem, os processos se tornam mais simples, tornando fácil a reutilização de código, além de facilitar a manutenção dos sistemas (BRASIL, 2009d).

Sobre a utilização e criação de camadas Lisboa (2009a) estabelece:

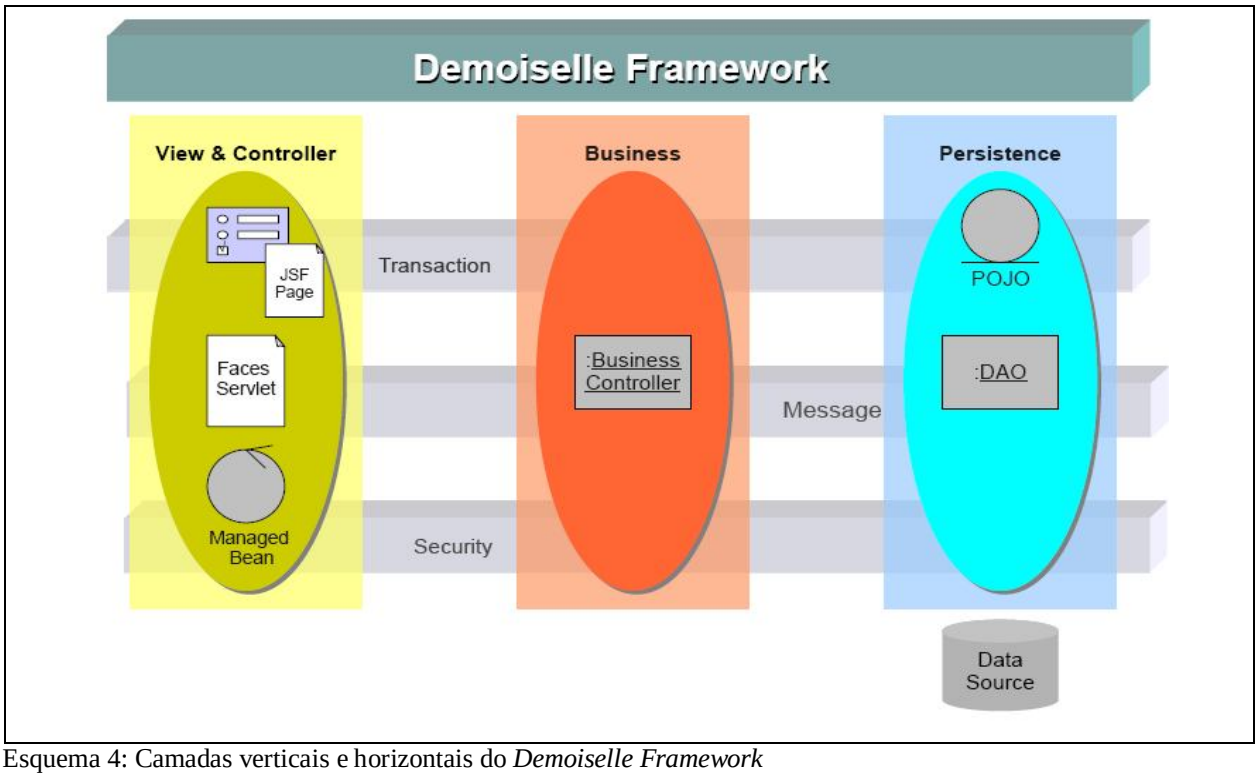
> A criação de camadas visa diminuir a complexidade da aplicação, separando-a em partes mais compreensíveis, substituíveis e fáceis de manter. Geralmente usamos metáforas como bolos e tortas, onde uma camada repousa sobre a outra. Nessa metáfora, as camadas de software só podem se comunicar com as camadas adjacentes, imediatamente "acima" ou "abaixo". O Demoiselle separa as aplicações em camadas, representadas na implementação por pacotes Java, seguindo este modelo.

Além de possuir as camadas clássicas do modelo MVC (Modelo, Visão e Controlador), o *framework* ainda possui camadas de persistência, transação, segurança, injeção de dependência e mensagem.

As camadas do *Demoiselle* são: *View* & *Controller*, corresponde às letras V e C do MVC e são responsáveis pela interface gráfica e controle de eventos; *Business*, camada que se encarrega das regras de negócio da aplicação; *Persistence*, corresponde a letra M do MVC, portanto aos modelos, é onde se encontram os mapeamentos, as classes POJOS (*Plain Old Java Objects*), filtros de pesquisas e classes DAO (*Data Access Object*) (LISBOA, 2009a).

Desta maneira cada camada possui sua função específica, possuindo determinados conjuntos de tarefas. No entanto, é necessário que algumas operações consigam passar livremente por todas as camadas, como por exemplo, as transações, as mensagens da aplicação e o controle de segurança. Isso é possível através da orientação a aspectos. A orientação a aspectos no *Demoiselle Framework* é feita através do *AspectJ*, que se trata e uma extensão para a linguagem Java, incluindo uma linguagem orientada a aspectos e um compilador (LISBOA, 2009a).

Segundo Lisboa (2009a) "Toda essa infraestrutura para as camadas tradicionais constitui o que chamamos de contextos". No esquema 4 podem ser observados os contextos em camadas horizontais e a implementação MVC do *Demoiselle* em camadas transversais.



Esquema 4: Camadas verticais e horizontais do *Demoiselle Framework*
Fonte: Adaptado de Lisboa (2009b).

Portanto, o *Demoiselle* se torna uma ótima opção para o desenvolvimento de aplicações voltadas para a *Web*, pois integra um conjunto de tecnologias que cooperam entre si buscando uma maior padronização, aumento da produtividade e diminuição do retrabalho.

2.9 UNIFIED MODELING LANGUAGE - UML

A cada dia que passa cresce o volume de informações para auxiliar nas tomadas de decisões de pequenas, médias e grandes empresas. Estas informações devem estar corretas e disponíveis de forma imediata, pois a maioria dos problemas encontrados nas organizações hoje em dia reside na adoção de métodos inapropriados para análise e levantamento de requisitos, resultando em projetos com objetivos incorretos ou não muito claros (LIMA, 2007).

A modelagem surge, pois não pode-se compreender os sistemas completamente, e sobre isso Lima (2006, p.29), argumenta:

> Os melhores modelos expressam a realidade, e quanto maior a complexidade do produto a ser entregue, maior a necessidade de boas técnicas de modelagem, com um conjunto de visões (pontos de vista ou perspectivas), suficientes para garantir a sua compreensão. [...] o exercício de concentrar-se no que é relevante e ignorar os demais detalhes, é essencial para apreender e comunicar o domínio da aplicação e, com bons modelos, podemos assegurar a comunicação entre a equipe de projeto e uma arquitetura sólida.

O *software* se tornou uma ferramenta indispensável para gerenciamento dos processos organizacionais. Neste sentido as empresas perceberam a quantidade de benefícios oferecidos pela orientação a objetos no desenvolvimento de aplicações corporativas e, a partir daí, começaram a adotar técnicas que possibilitassem implementar o modelo orientado a objeto, definindo padrões claros e objetivos, principalmente com o crescimento da *Web* (LIMA, 2007).

Com o objetivo de especificar, documentar e estruturar os projetos de *softwares*, permitindo padronizar as formas de modelagem, surge a UML e segundo Lima (2007, p.40), “A UML é uma linguagem visual para especificação, construção, visualização e documentação de artefatos de um software intensivo, para representar projetos orientados a objetos utilizando uma notação comum.” Sobre a *Unified Modeling Language* (UML), Pender (2004, p.3) argumenta que

“[...] é uma destilação de três notações principais e uma série de técnicas de modelagem retiradas de metodologias bastante diversificadas, que já existem na prática desde as duas últimas décadas.”

Suas características permitem criar modelos que admitam escalabilidade e segurança, e sobre isso Pender (2004, p.3) cita:

> Como a modelagem UML eleva o nível de abstração por todo o processo de análise e projeto, é mais fácil identificar padrões de comportamento e, portanto, definir oportunidades para recriação e reuso. Conseqüentemente, a modelagem UML facilita a criação de projetos modulares, resultando em componentes e bibliotecas de componentes que agilizam o desenvolvimento e ajudam a garantir a coerência através de sistemas e implementações.

A UML se destaca por oferecer um conjunto de recursos extremamente valiosos para o mundo da modelagem, existem os mecanismos de extensibilidade, onde em um primeiro momento a UML foca em conceitos básicos de funcionalidades, sendo que por meio desta extensibilidade estes recursos iniciais podem ser ajustados ou aumentados sem alterar a sua integridade inicial. Para solucionar problemas comuns, existem os padrões e colaborações, para modelar processos lógicos existem os diagramas de atividades, para tratar relacionamentos entre níveis de abstração existe o refinamento, para modelar focalizando conectividade e comunicação existe as interfaces e componentes e para garantir a integridade dos modelos existe a linguagem de restrição (PENDER, 2004).

A UML possui uma área de abrangência muito grande, oferecendo diversas formas para a realização de diagramação. Os diagramas estruturais e comportamentais se destacam por estar entre os mais utilizados para modelagem de sistemas nos dias atuais.

Descrever os elementos que compõe o sistema por meio de diagramas de classes, objetos, estrutura de composição, componentes e implantação é característica dos diagramas estruturais. Já a característica de descrever o comportamento dos elementos e suas interações por meio de diagramas de caso de uso, atividades, interação e máquina de estados referenciam os diagramas comportamentais (PENDER, 2004).

2.9.1 Diagrama de Classes

Tem como característica definir os recursos essenciais para a correta operação do sistema, modelando cada recurso em termos de sua estrutura, relacionamento e comportamento, definindo a geração de código, oferecendo um foco para os demais diagramas e possibilitando a engenharia reversa (PENDER, 2004).

Sobre este diagrama Lima (2004, p.174) argumenta:

> Um diagrama de classe mostra a estrutura estática do modelo, em que os elementos são representados por classes, com sua estrutura interna e seus relacionamentos. Os diagramas de classes também podem ser organizados em pacotes, mostrando somente o que é relevante em um pacote específico, considerando, por exemplo, o contexto que um grupo de classes têm em comum [...], ou os atores que utilizam seus serviços [...].

Portanto, o diagrama de classe representa as classes, atributos e métodos que serão implementados ao longo do desenvolvimento de uma aplicação.

2.9.2 Diagrama de Caso de Uso

Tem como característica identificar os recursos propostos para o sistema de uma forma mais simples, facilitando a visualização dos requisitos por parte do cliente. É por meio dele que o cliente saberá se o sistema está atingindo as suas expectativas, servindo também, como base para o acompanhamento do progresso dos trabalhos (PENDER, 2004).

Sobre este diagrama Lima (2004, p.58) argumenta:

> Um caso de uso bem escrito é fácil de ler, pois consiste em sentenças escritas em passos de ação simples, em que um ator alcança um resultado ou transmite informação ao sistema. De modo geral, a leitura de um caso de uso toma poucos minutos e pode ser compreendida por qualquer leitor envolvido com o negócio, não sendo exigido nenhum conhecimento de informática – o foco é o problema e não a solução informatizada.

Desta forma, quanto mais claro e objetivo se definir os casos de usos, maiores são as chances de sucesso da aplicação no momento de obter um *feedback* do cliente.

2.10 SCRUM

Agora é apresentado um embasamento teórico sobre a metodologia ágil *Scrum*. Suas principais características bem como seu ciclo de desenvolvimento serão abordados.

2.10.1 Forma de Trabalho

O *Scrum* pode ser definido como uma metodologia de desenvolvimento ágil, que possui como característica o gerenciamento de projetos complexos através de métodos simples. O *Scrum* visa os pontos mais importantes do negócio, assim as principais funcionalidades são desenvolvidas e entregues primeiro, atendendo mais rapidamente as necessidades dos clientes. Processos das metodologias tradicionais possuem o foco voltado para o desenvolvimento em regras, diferentemente do *Scrum* (FERREIRA, 2009).

As principais características do *Scrum* estão na forma do trabalho em equipe e na elaboração dos processos, de acordo com Pressman (2006), os princípios do *Scrum* são definidos como:

- Formação de pequenas equipes de trabalho, comprometidas com a comunicação e compartilhamento de informações;
- Possibilidade de adaptação dos processos, ao longo da evolução do projeto;
- Freqüentemente existe algum incremento de *software*, onde tanto a equipe ou o cliente podem avaliar;
- A realização de testes e a construção da documentação acontecem ao longo do desenvolvimento do produto.

2.10.2 Desenvolvendo um Projeto com Scrum

Ao iniciar-se algum projeto com *Scrum* faz-se necessário que sejam estabelecidos alguns papéis (responsabilidades), para o correto funcionamento dos processos. Existem três papéis elementares e que são definidos como *Product Owner, Scrum Master* e equipe. O *Product Owner* é responsável pela definição dos requisitos, geralmente é caracterizado pelo cliente. O *Scrum Master* tem o objetivo de facilitar o trabalho durante o desenvolvimento, ele é responsável por retirar obstáculos do caminho e gerenciar o projeto. A equipe caracteriza-se por programar o produto, sendo composta por 5 a 9 pessoas, onde trabalham de forma auto gerenciada (ARAUJO, 2009).

Durante a utilização do *framework Scrum* é necessário observar alguns processos de desenvolvimentos ou atividades. Pressman (2006) cita que estes processos são caracterizados como pendência, *sprints*, reuniões de *Scrum* e *demos*.

A pendência também conhecida por *product backlog*, caracteriza-se por conter os requisitos, características ou problemas a serem trabalhados ao longo do projeto. Uma pendência sempre pode ser incrementada de algum destes fatores, sendo responsabilidade do gerente verificar a viabilidade da inclusão (PRESSMAN, 2006).

Os *sprints* são definidos como unidades de trabalho responsáveis para desenvolver um item de alguma pendência, onde é definido um tempo (duas a quatro semanas) para o desenvolvimento e este tempo deve ser rigorosamente cumprido. Quando iniciado um *sprint* não deve mais ocorrer inclusão de itens a pendência em trabalho, garantindo estabilidade e velocidade ao trabalho em realização (PRESSMAN, 2006).

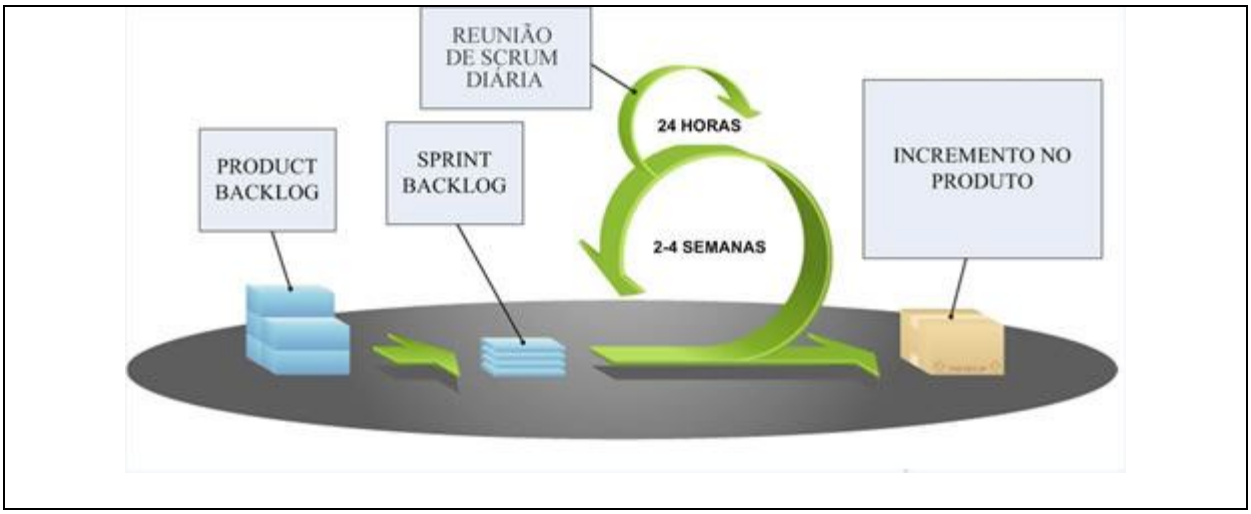
As reuniões de *Scrum* ocorrem diariamente, sendo de duração curtas (15 minutos), consistindo em algumas perguntas básicas sobre o desenvolvimento individual dos colaboradores. As perguntas geralmente lançadas pelo líder da equipe são:

- O que foi realizado ontem?
- O que será feito hoje?
- Houve algo que atrapalhou o trabalho ontem?

As reuniões servem para identificar os problemas ou até futuros problemas. Também tem caráter de socialização dos conhecimentos e experiências (PRESSMAN, 2006).

Os *demos* caracterizam-se por ser um incremento de *software*. O cliente através deste incremento é capaz de avaliar se a funcionalidade está dentro dos requisitos desejados. Um *demo* não é necessariamente o módulo ou funcionalidade completa, mas sim a pendência que estava estabelecida e foi realizada nos *sprints* no prazo estabelecido (PRESSMAN, 2006).

O esquema 5 representa as fases do desenvolvimento com *Scrum*.



Esquema 5: Ciclo do *Scrum*
Fonte: Adaptado de Araujo (2009).

Devido ao fato de ser uma metodologia de fácil entendimento, os projetos coordenados com essa metodologia tendem a ter um grau de complexidade gerencial baixo, acarretando em uma elevada velocidade de desenvolvimento e um produto de qualidade, sendo estas características fundamentais no desenvolvimento de *software*.

3 DESENVOLVIMENTO

O projeto teve como objetivo o estudo do *framework Demoiselle* e a sua posterior utilização no desenvolvimento de um *software* para gerenciamento do acervo bibliográfico. Agora serão descritas as diretrizes envolvidas no desenvolvimento do projeto.

Primeiramente foi realizado um estudo da linguagem Java, revendo os conceitos de orientação a objetos e fixando os conceitos da programação orientada a aspecto que são empregados no *framework Demoiselle*. Também foi contemplado o estudo de alguns *frameworks* que estão acoplados ao *Demoiselle* como o JSF (*Java Server Faces*) e o *Hibernate*. Finalizando essa primeira parte, foi estudado o funcionamento dos processos bibliotecários, com o objetivo de facilitar o processo de análise e desenvolvimento do *software*.

Após a realização dos estudos das tecnologias empregadas e dos processos realizados no universo das bibliotecas foi realizado o levantamento dos requisitos prevendo as funcionalidades que o *software* deveria atender. Posteriormente foi realizada a modelagem dos diagramas UML necessários para o desenvolvimento e entendimento da aplicação. O que se buscou foi seguir o desenvolvimento de acordo com as etapas definidas no planejamento realizado e descritas no cronograma entregue no projeto.

Para possibilitar a separação das funcionalidades do sistema, o aplicativo foi divido em módulos que compreendem: cadastros, aquisição, formato MARC, procedimentos (catalogação, consulta e exemplares), circulação de materiais, gerenciamento de usuários e configurações.

O *software* foi desenvolvido com o objetivo de ser ambientado na *Web*. Este ambiente foi escolhido pelo fato de se oferecer serviços ao usuário final, como por exemplo, consulta do acervo, reserva, empréstimo, devolução e renovação de materiais. Ao se utilizar a *Web* é possível ter um *software* multiusuário e multiplataforma, neste sentido, agrega-se valor ao sistema oferecendo uma independência de sistema operacional.

Para o desenvolvimento da aplicação fez-se necessário a utilização de uma ferramenta de desenvolvimento. A ferramenta utilizada foi a IDE (*Integrated Development Environment*) *open source* Eclipse em sua versão 3.5 *Galileo*. Juntamente ao Eclipse, foi realizada a instalação dos *plug-ins* do *framework Demoiselle*, do *AspectJ* (para a programação orientada a aspectos), do

JSF, do *Hibernate* e do SVN *Subversion* (necessário para o controle de versão e gerenciamento de código fonte em trabalhos em equipes).

A base de dados por definições do *framework* foi desenvolvida com o *Postgres* 8.4, que é um SGDB (Sistema Gerenciador de Banco de Dados) gratuito e muito robusto para aplicações em geral. A modelagem da base de dados foi realizada com a ferramenta DB *Designer* 4 *Fork*, que permite a criação do diagrama e em seguida exportar o *script* para o SGDB em questão.

Para disponibilizar a aplicação desenvolvida ao usuário final e também poder visualizar a aplicação para testes manuais foi necessário a utilização do servidor de aplicação *Apache Tomcat* 6, um dos mais utilizados no desenvolvimento de aplicações corporativas em Java.

Com o objetivo de oferecer mais segurança e qualidade no *software* desenvolvido, ao término da codificação de cada módulo do sistema foi realizada uma bateria de testes para verificar o seu correto funcionamento. Neste sentido, foi organizado um roteiro percorrendo cada módulo e verificando o seu funcionamento.

O *software* em questão teve como base para levantamentos de requisitos a Biblioteca Pública de São Miguel do Oeste, contando com o apoio da equipe de funcionários, bem como da bibliotecária da UNOESC (Universidade do Oeste de Santa Catarina) Sra. Marilene Chaves Furtado, inscrita no CRB (Conselho Regional de Biblioteconomia) sob número 14/545.

3.1 METODOLOGIA DE DESENVOLVIMENTO

Pelo fato do *Scrum* se tratar de uma metodologia ágil para gestão e planejamento de projetos de *softwares*, se optou por adotar uma combinação de técnicas que o compõe visando os pontos mais importantes do projeto. Desta forma, o objetivo foi priorizar as principais funcionalidades do sistema, possibilitando a entrega de cada funcionalidade em um espaço menor de tempo. Para elaboração e desenvolvimento do projeto foram adotadas as seguintes técnicas:

- Elaboração do *product backlog:* tem como característica conter os requisitos a serem trabalhados ao longo de um projeto. Em complementação ao trabalho foram feitas visitas à Biblioteca Pública de São Miguel do Oeste para coletar as necessidades e

informações de como funcionam os processos da biblioteca. Após o levantamento de dados foi elaborada uma visão geral do sistema e um detalhamento dos requisitos, conforme demonstra o quadro 1.

R = Requisito Funcional R 1: Efetuar Catalogação		RF = Requisito Não Funcional Oculto ( )		
Descrição: O sistema deve permitir a catalogação do acervo bibliográfico, por meio do formato MARC, com seus campos, sub-campos e indicadores. Deve ser possível controlar múltiplos exemplares, amarrados ao título (acervo) da obra. As principais informações mantidas sobre uma obra são: autor, título, ISBN, editora, cidade e, quando for o caso, a série ou coleção à qual o livro pertence.				
Requisitos Não Funcionais				
Nome	Restrição	Categoria	Desejável	Permanente
RF 1.1 Cadastros Auxiliares	Os cadastros auxiliares já devem estar cadastrados no sistema para que seja possível efetuar a catalogação.	Segurança	( )	(X)
RF 1.2 Identificação dos Exemplares	Os números gerados automaticamente deverão ser únicos, para assim distinguir todos os exemplares.	Segurança	( )	(X)
RF 1.3 Janelas de Cadastro	A catalogação deve ser feita em apenas uma tela (janela), separando os dados por abas.	Interface	(X)	( )

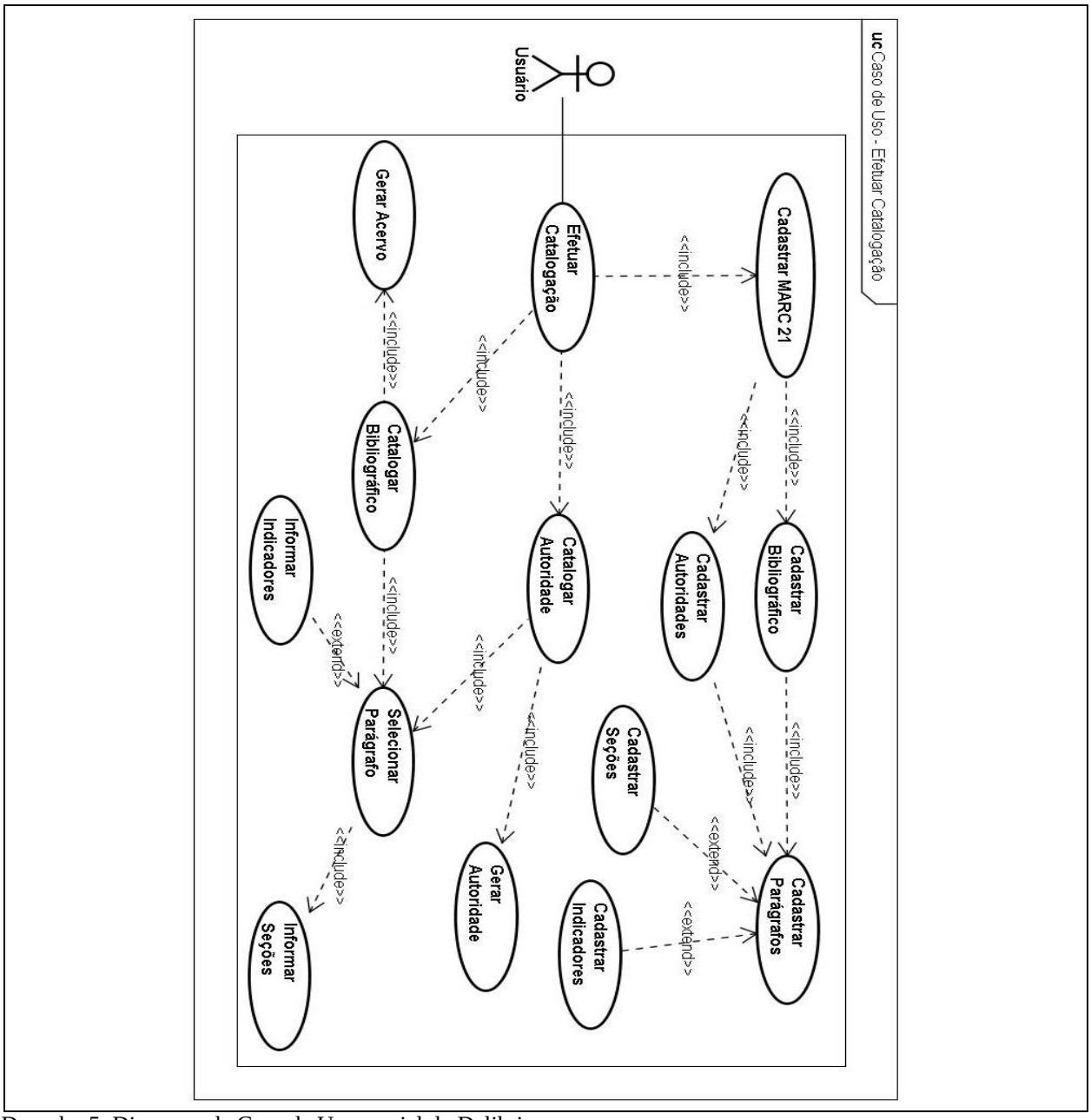
Quadro 1: Detalhamento parcial do requisito “Efetuar Catalogação”
Fonte: Os autores (2010).

- Divisão do sistema em módulos: visando reduzir a complexidade do sistema, pois tratar cada módulo é mais simples do que o sistema em sua globalidade, o aplicativo foi divido em módulos. Estes compreendem: cadastros, aquisição, formato MARC, procedimentos (catalogação, consulta e exemplares), circulação de materiais, gerenciamento de usuários e configurações.
- Acompanhamento visual através do quadro *Kanban* (apêndice D): com o objetivo de se obter uma maior comunicação e visibilidade entre as pessoas envolvidas diretamente no desenvolvimento do projeto, todos os requisitos foram colocados em um quadro de acompanhamento de execução de tarefas. Cada item do *product backlog* foi colocado no quadro por meio de *post-its* da mesma cor, divididos em: “planejado, iniciado, testes e pronto”. Para tratar tarefas em atraso, não previstas ou *bugs* foram utilizados *post-its* de cores diferentes facilitando a visualização do andamento do projeto. Cada requisito mudava de coluna conforme o seu processo amadurecia.

Por meio da utilização destas técnicas do *framework Scrum*, foi possível reduzir o grau de complexidade de entendimento e desenvolvimento da aplicação voltando esforços para as principais funcionalidades do sistema, oferecendo assim, um produto de maior qualidade em um espaço de tempo menor.

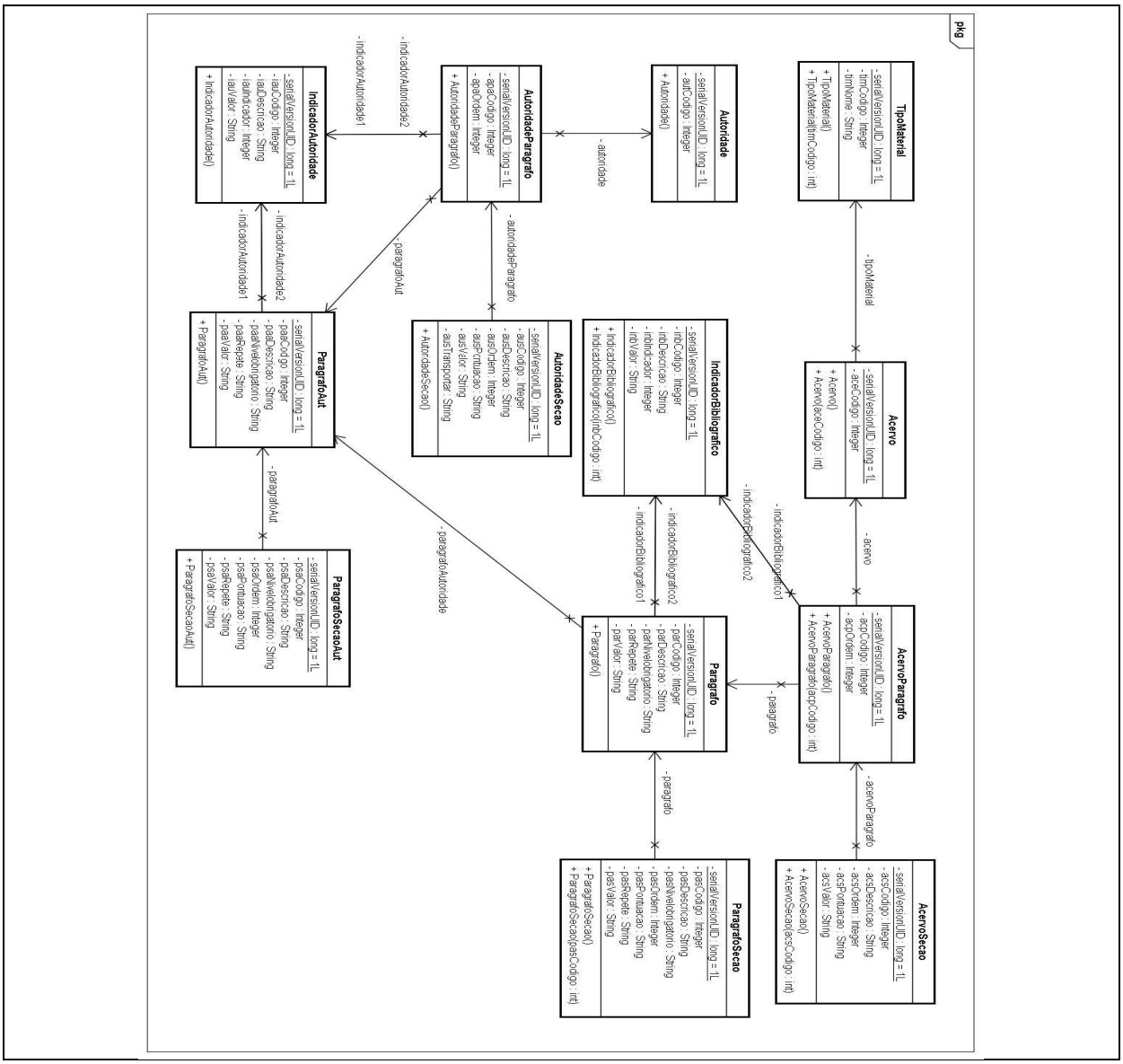
3.1.1 Modelagem e Diagramas

Com o objetivo de efetuar uma avaliação mais detalhada dos requisitos do sistema, foram desenvolvidos dois tipos de diagramas: diagrama de caso de uso e diagrama de classes.



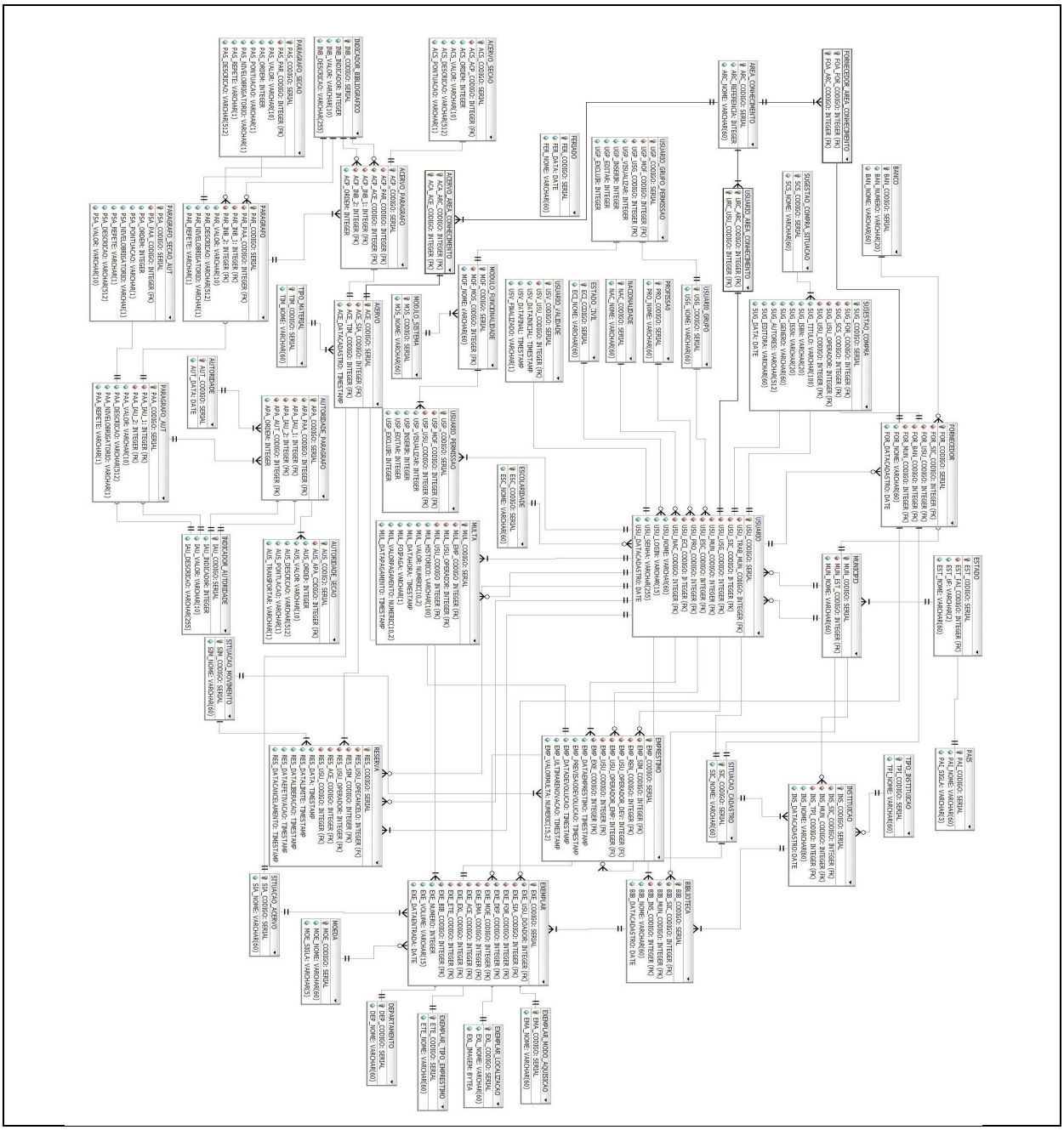
Desenho 5: Diagrama de Caso de Uso parcial do Delibris
Fonte: Os autores (2010).

O diagrama de caso de uso, conforme desenho 5, é uma técnica que descreve os requisitos funcionais do sistema e tem como objetivo permitir uma compreensão mais simples dos processos desenvolvidos no sistema Delibris.



Desenho 6: Diagrama de classes parcial do Delibris
Fonte: Os autores (2010).

O diagrama parcial de classes (desenho 6) tem por objetivo demonstrar a estrutura e o relacionamento entre as classes do sistema. É ferramenta indispensável na análise do programador para entender o funcionamento de cada classe que compõe o sistema.



Desenho 7: Diagrama entidade-relacionamento do Delibris
Fonte: Os autores (2010).

Desenvolveu-se um diagrama entidade-relacionamento conforme os requisitos levantados por meio da ferramenta *DBDesignerFork*. Esta ferramenta possibilita modelar o banco de dados de forma visual, conforme mostra o desenho 7.

3.2 DELIBRIS

O *Demoiselle Framework* foi concebido com o intuito de oferecer uma maior padronização, agilidade e qualidade no desenvolvimento de soluções. O Delibris está sendo desenvolvido sob a perspectiva de usufruir dos recursos oferecidos por esta tecnologia objetivando automatizar com qualidade a gestão de bibliotecas.

O desenvolvimento por meio de tecnologias livres é a característica base do projeto Delibris. A arquitetura de referência proposta pelo *Demoiselle* possibilita que a aplicação seja construída de forma distribuída, baseada em componentes, com processos orientados a eventos resultando em um baixo acoplamento de funções de negócios e um fácil acesso por meio da internet. Neste sentido, o Delibris pode ser considerado um *software* escalável, pois permite um crescimento de dados e uma expansão do escopo de funcionalidades do sistema, possuindo toda a sua interface baseada em navegador.

3.2.1 Delibris - Arquitetura

O *Demoiselle* tem como uma de suas premissas o baixo acoplamento, desta forma, ao utilizá-lo no desenvolvimento de uma aplicação, como ponto de partida acontece uma separação do projeto em três camadas distintas: camada de visão, camada de negócio e camada de persistência. A estrutura de pacotes tem o objetivo de garantir a injeção de dependência para manter a integração entre as camadas do sistema com um baixo acoplamento cujo resultado é maior facilidade no momento de efetuar a manutenção nas classes, pois elas estão organizadas tanto em sua estrutura como em seu código fonte.

Estrutura do Delibris – Função de Cada Pacote
br.com.sistema.delibris.bean
Pacote onde serão armazenadas as classes dos POJOs da aplicação.
br.com.sistema.delibris.business
Pacote onde serão armazenadas as interfaces dos Business Controllers da aplicação.
br.com.sistema.delibris.business.implementation
Pacote onde serão armazenadas as implementações dos Business Controllers da aplicação.
br.com.sistema.delibris.config
Pacote onde serão armazenadas as classes de configuração.
br.com.sistema.delibris.constant
Pacote onde serão armazenadas as classes de constantes da aplicação.
br.com.sistema.delibris.message
Pacote onde serão armazenadas as classes de mensagens da aplicação.
br.com.sistema.delibris.persistence.dao
Pacote onde serão armazenadas as interfaces dos DAOs da aplicação.
br.com.sistema.delibris.persistence.dao.filter
Pacote onde serão armazenadas as classes dos filtros de dados da aplicação.
br.com.sistema.delibris.persistence.dao.implementation
Pacote onde serão armazenadas as implementações dos DAOs da aplicação.
br.com.sistema.delibris.view.managedbean
Pacote onde serão armazenadas as implementações dos *Managed Beans* da aplicação.
br.com.sistema.delibris.view.report
Pacote onde serão armazenadas as implementações dos relatórios da aplicação.

Quadro 2: Função de cada pacote da estrutura do Delibris
Fonte: Os autores (2010).

Como pode ser observado no quadro 2, cada pacote possui sua função específica dentro do sistema e são separados por responsabilidades em comum. Toda essa estruturação fornece inúmeros benefícios, pois a partir do momento que o projeto começa a ganhar corpo, é extremamente importante manter uma organização entre as classes, para não haver trechos de

códigos ilegíveis e de difícil entendimento. Os pacotes responsáveis por implementar as interfaces possuem um papel muito importante, pois abstraem o objeto para os pacotes de implementação.

3.2.2 Camada de Persistência

Esta camada tem por objetivo garantir transparência às demais camadas da aplicação na manipulação e tratamento das informações providas do Sistema Gerenciador de Banco de Dados. Nela acontecem as principais operações de um DAO (*Data Access Object*), as implementações JDBC (*Java Database Connectivity*) e a implementação da interface para tratamento de Mapeamento Objeto Relacional (ORM).

A camada de persistência engloba os seguintes pacotes:

- br.com.sistema.delibris.*bean*;
- br.com.sistema.delibris.*persistence*.dao;
- br.com.sistema.delibris.*persistence*.dao.*filter*;
- br.com.sistema.delibris.*persistence*.dao.*implementation*;

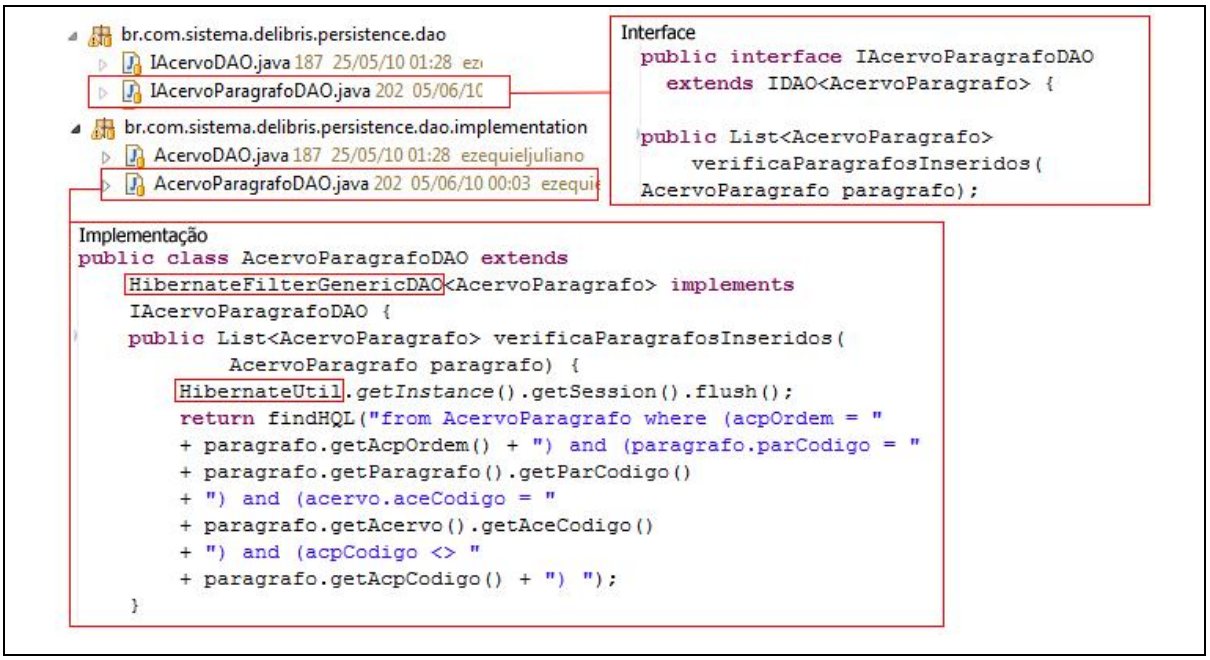
O pacote *Bean* representa as classes do tipo Pojo, que são as entidades da aplicação. Utilizando o *Hibernate* foi realizado o Mapeamento Objeto Relacional com o uso de *Annotation* conforme mostra o desenho 8.

Acervo.java

```
@Entity
@Table(name = "acervo") //identifica a tabela
public class Acervo implements Serializable,
        br.gov.framework.demoiselle.core.bean.IPojo { //classe IPojo do Demoiselle
    private static final long serialVersionUID = 1L;

    @Id //Identifica chave primária e utiliza o generator da base
    @SequenceGenerator(name = "ACERVO_ACECODIGO_GENERATOR", sequenceName = "ACERVO_ACE_CODIGO")
    @GeneratedValue(strategy = GenerationType.IDENTITY, generator = "ACERVO_ACECODIGO_GENERATOR")
    @Column(name = "ace_codigo")
    private Integer aceCodigo;

    @Column(name = "ace_datacadastro") //mapeamento da coluna da tabela com o atributo
    private Timestamp aceDatacadastro;

    // Bi-Direcional muitos-pra-um associacao com Situacao do Acervo
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "ace_sia_codigo")
    private SituacaoAcervo situacaoAcervo;

    // Bi-Direcional muitos-pra-um associacao com Tipo de Material
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "ace_tim_codigo")
    private TipoMaterial tipoMaterial;

    // Bi-Direcional muitos-pra-muitos associacao com Area de Conhecimento
    @ManyToMany
    @JoinTable(name = "acervo_area_conhecimento",
            joinColumns = { @JoinColumn(name = "aca_ace_codigo") },
            inverseJoinColumns = { @JoinColumn(name = "aca_arc_codigo") })
    private List<AreaConhecimento> areaConhecimentos;
```

Desenho 8: Mapeamento com *Annotation*
Fonte: Os autores (2010).

Conforme pode ser acompanhado no desenho 9, o pacote *DAO.Implementation* contém a implementação dos métodos que realizam o tratamento das informações providas do banco de dados e o pacote DAO contém as interfaces destas implementações. O controle de transação é feito por meio do *framework Hibernate* que oferece um conjunto de funcionalidades integradas ao contexto de transação provendo um mecanismo completo para a camada de persistência. O *HibernateFilterGenericDAO* implementa funcionalidades (consulta, paginação, inserção, alteração e exclusão) que simplificam a criação das classes IDAO (interfaces) que são criadas na aplicação. O *HibernateUtil* se responsabiliza pela integração da persistência *Hibernate* e o controle transacional tendo como principais funções criar/fornecer transações e seções e aplicar *Commit* e *Rollback*.



Desenho 9: DAO e DAO.*Implementation* utilizadas no Delibris
Fonte: Os autores (2010).

Ainda nesta camada foi utilizado o componente *Hibernate Filter* adaptado ao *Demoiselle* para a geração de filtros de pesquisas conforme pode ser observado no desenho 10. O pacote DAO.*Filter* é responsável por implementar estes filtros, sendo possível definir um conjunto de campos a serem filtrados, ordenando os mesmo em ordem crescente ou decrescente e aplicando conjunto de critérios como: *addEquals, addNotEquals, addLike, addLikeIgnoreCase, addBetween, addIsNull*.

FiltroParagrafo.java

```
package br.com.sistema.delibris.persistence.dao.filter;

import br.com.sistema.delibris.bean.Paragrafo;

public class FiltroParagrafo extends Filter {

    private static final long serialVersionUID = 1L;

    public static final String ACPCODIGO = "acpCodigo";
    public static final String PARCODIGO = "parCodigo";
    public static final String PARVALOR = "parValor";
    public static final String PARDESCRICAO = "parDescricao";
    public static final String PARNIVELOBRIGATORIO = "parNivelobrigatorio";
    public static final String PARREPETE = "parRepete";

    public FiltroParagrafo() {
        setClazz(Paragrafo.class);
        addOrder(PARDESCRICAO, ASC);
    }

}
```

Utilizando o Filtro em Outra Classe

```
public Paragrafo buscar(Paragrafo paragrafo) {
    HibernateUtil.getInstance().getSession().flush();
    FiltroParagrafo filtro = new FiltroParagrafo();
    filtro.addEquals(FiltroParagrafo.PARCODIGO,
            paragrafo.getParCodigo());
    List<Paragrafo> retorno = find(filtro);
    if (retorno != null && retorno.size() > 0)
        return retorno.get(0);
    return null;
}
```

Desenho 10: DAO.*Filter* utilizado no Delibris
Fonte: Os autores (2010).

A paginação de dados também está disponível nesta camada, tendo como objetivo manter um controle dos dados providos do banco de dados, recuperando apenas as informações que serão exibidas na visão do usuário. Um grande benefício trazido pela sua utilização é a redução de dados trafegados na rede, bem como uma diminuição do processamento por parte do servidor de aplicação.

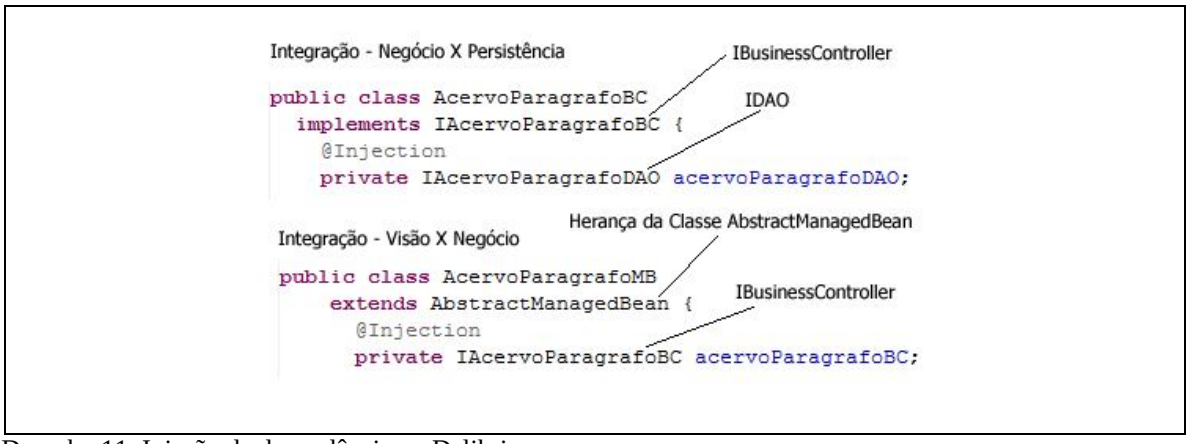
3.2.3 Camada de Negócio

Esta camada é responsável por implementar as regras de negócio definidas para o sistema especificando os processos de integração entre os módulos. Esta integração com as demais camadas ocorre por meio do mecanismo de injeção de dependência. Na arquitetura do *Demoiselle*

o módulo *Core* que especifica quem trata a injeção de dependência, os módulos que implementam o *Core* definem como esta injeção será realizada e o módulo *Web* faz a implementação da mesma.

Os pacotes *Business* (interface) e *Business.Implementation* (implementação) representam a camada de negócio da aplicação, sendo responsáveis por implementar as regras pertinentes a esta camada e integrando os processos entre os módulos. O tratamento das informações na camada de negócio possibilita que os dados sejam enviados para a camada de persistência de forma mais consistente, diminuindo a possibilidade de armazenamento de informações incoerentes.

Quando a camada de negócio faz uma integração com a camada de visão usa-se a interface *IViewController* (abstração para o objeto da camada de visão) ou a herança da classe *AbstractManagedBean* e a interface *IBusinessController* (abstração para o objeto da camada de negócio). Quando ocorre com a camada de persistência utilizam-se as interfaces *IBusinessController* e *IDAO* (abstração para o objeto da camada de persistência).



Desenho 11: Injeção de dependência no Delibris
Fonte: Os autores (2010).

O controle de acesso da aplicação é feito por meio do componente *Demoiselle Authorization*, que fornece mecanismos de autorização baseado em especificações JAAS (*Java Authentication and Authorization Service*). O JAAS é uma API que permite que aplicações escritas em J2EE usem serviços de autenticação e autorização sem a necessidade de estarem diretamente dependentes desses serviços. Para ter acesso às demais funcionalidades do sistema

este componente utiliza a Orientação a Aspectos (AOP) e se destaca neste sentido por poder ser utilizado em qualquer camada do sistema. O *Demoiselle Authorization* define as regras de autorização para os métodos das classes *IViewContoller*, *IBusinessController* e IDAO através da meta-informação *Annotation*: *RequiredRole*, por meio dela podem ser inseridos novos papéis conforme mostra o desenho 12.

```
@RequiredRole(roles={AliasRole.ROLE_ADMINISTRADOR,
        AliasRole.ROLE_BIBLIOTECARIO})
public void alterar(Profissao arg0) {
    profissaoDAO.update(arg0);
    ContextLocator.getInstance().
    getMessageContext().addMessage(InfoMessage.ALTERAR_OK);
}
```

Desenho 12: Inclusão de um papel de usuário
Fonte: Os autores (2010).

Quando ocorre uma violação de uma regra de autorização o *Demoiselle Authorization* faz uso *AutorizationException* que efetua o tratamento destas exceções. Essas exceções podem ser do tipo "*Checked*" (necessário o tratamento) e "*Un-Checked*" (não força o tratamento). No *Demoiselle* cada componente pode gerar suas exceções, a mais comum delas é a *ApplicationRuntimeException*, como pode ser observado no desenho 13.

```
/**
 * Valida os campos que são not null;
 * */
public void validarCamposParagrafo(Paragrafo paragrafo) {
    if (paragrafo.getParValor() == null) {
        throw new ApplicationRuntimeException(
                ErrorMessage.PARAGRAFO_PARVALOR_NULL);
    } else {
        if (paragrafo.getParNivelobrigatorio() == null) {
            throw new ApplicationRuntimeException(
                    ErrorMessage.PARAGRAFO_PARNIVELOBRIGATORIO_NULL);
        } else {
            if (paragrafo.getParRepete() == null) {
                throw new ApplicationRuntimeException(
                        ErrorMessage.PARAGRAFO_PARREPETE_NULL);
            }
        }
    }
}
```

Desenho 13: Efetuando tratamento de exceção
Fonte: Os autores (2010).

Ao realizar tratamento das exceções é interessante mostrar mensagens mais amigáveis aos usuários, isto é possível através dos contextos de mensagens que possuem como característica efetuar a troca das mesmas entre as camadas da aplicação utilizando a interface *IMessage*.

3.2.4 Camada de Visão

A camada de visão é representada pelo pacote *ManagedBeans* e tem como responsabilidade apresentar as informações aos usuários e controlar suas interações com o sistema. Ao utilizar o *Demoiselle Framework*, o módulo de visão e controle é gerenciado pelo *Java Server Faces* (JSF).

Para que o JSF funcione de forma correta, é necessário configurar os arquivos *web.xml* e *faces-config.xml* (desenho 14) da aplicação. Além disso, é necessário criar as páginas JSP com os componentes JSF mapeados para os *ManagedBeans*.

faces-config.xml

```
    <managed-bean-class>br.com.sistema.delibris.view.managedbean.BancoMB</managed-bean-class>
    <managed-bean-scope>session</managed-bean-scope>
  </managed-bean>
  <managed-bean>
    <managed-bean-name>moduloSistemaMB</managed-bean-name>
    <managed-bean-class>br.com.sistema.delibris.view.managedbean.ModuloSistemaMB</managed-bean-class>
    <managed-bean-scope>session</managed-bean-scope>
  </managed-bean>
  <managed-bean>
    <managed-bean-name>usuarioMB</managed-bean-name>
    <managed-bean-class>br.com.sistema.delibris.view.managedbean.UsuarioMB</managed-bean-class>
    <managed-bean-scope>session</managed-bean-scope>
  </managed-bean>
  <managed-bean>
    <managed-bean-name>usuarioGrupoMB</managed-bean-name>
    <managed-bean-class>br.com.sistema.delibris.view.managedbean.UsuarioGrupoMB</managed-bean-class>
    <managed-bean-scope>session</managed-bean-scope>
  </managed-bean>
```

Desenho 14: Arquivos faces-config do Delibris
Fonte: Os autores (2010).

Os *ManagedBeans* tem como característica implementar a interface *IViewControler* ou herdar a classe *AbstractManagedBean*. O JSF permite ao usuário interagir com a página no navegador da aplicação executando alguma ação, conseqüentemente esta ação gera uma requisição HTTP, esta requisição é tratada pelo *ManagedBean* que se encontra mapeado no componente que gerou a requisição, conforme mostra o desenho 15.

```
<h:column>
    <fieldset>
       <legend>
          <h:outputLabel
               for="acervoParagrafoMB_acervo" styleClass="outputLabel"
               value="Código do Acervo" />
       </legend>
          <t:inputText disabled="true"
               id="acervoParagrafoMB_acervo" tabindex="1"
               style="width: 100px; background-color:#FFD8AF;"
               styleClass="inputText"
               value="#{acervoParagrafoMB.acervoParagrafo.acervo.aceCodigo}" />
    </fieldset>
</h:column>
```

Desenho 15: Mapeamento do *ManagedBean* com JSF
Fonte: Os autores (2010).

Desta forma, o *ManagedBean* manda o Pojo mapeado para a camada de negócio, que realiza o tratamento das informações e retorna os dados para o *ManagedBean*. Posteriormente os dados são formatados e a página *web* é construída exibindo estes dados ao usuário.

3.2.5 Interface

A preocupação com a automatização dos processos bibliotecários cresce na mesma proporção em que surgem novas bibliotecas. Oferecer uma solução de qualidade que diminua o impacto desta automatização sempre foi o objetivo vislumbrado. Mas é extremamente importante estender este conceito de qualidade para a forma de como os dados são apresentados ao usuário final, por meio de uma interface simples e intuitiva.

O sistema oferece toda a estrutura de cadastros auxiliares e cadastros bases para o gerenciamento do acervo bibliográfico, disponibiliza toda a estrutura de cadastramento do formato MARC para lançamento do acervo e das autoridades, além de oferecer mecanismos para o lançamento de informações referente à circulação de materiais.

Para acessar o sistema Delibris, obrigatoriamente o usuário necessita estar logado, isto é possível por meio da tela de acesso ao sistema, conforme mostra o desenho 16. O projeto conta com uma área de gerenciamento e controle de usuários, onde podem ser definidos grupos especificando suas permissões de acesso.

Desenho 16: Tela de acesso ao sistema
Fonte: Os autores (2010).

No desenho 17 é apresentada a visualização do usuário após o *Login*, onde o "Menu" central é disponibilizado com ícones grandes destacando as principais funcionalidades do sistema, visando uma maior facilidade na interação por parte do usuário. O "Menu" superior fica sempre disponível, pois as páginas são carregados de forma dinâmica na parte central, desta forma o usuário tem mais facilidade para abrir as demais telas de lançamento de informações do Delibris.

Desenho 17: Tela principal do sistema Delibris
Fonte: Os autores (2010).

Conforme pode ser observado no quadro 3, o Delibris disponibiliza toda a infra-estrutura de cadastros bases para o funcionamento das principais funcionalidades do sistema, além de oferecer, em relação ao formato MARC, os cadastros de indicadores e parágrafos tanto no modo bibliográfico como no modo de autoridades.

Estrutura do Menu Principal		
Cadastros	Auxiliares	Áreas de Conhecimentos
		Departamentos
		Localização de Exemplares
		Modos de Aquisição
		Tipos de Empréstimos
		Tipos de Instituições
		Tipos de Materiais
	Básicos	Bancos
		Escolaridades

		Estados Civis
		Nacionalidades
		Moedas
		Profissões
	Localização	Estados
		Municípios
		Países
	Situações	Acervo
		Cadastro
		Movimento
Aquisição	Fornecedores	
Formato MARC21	Bibliográfico	Indicadores
		Parágrafos
	Autoridade	Indicadores
		Parágrafos
Circulação de Materiais	Reservas	
	Empréstimos	
	Multas	
Procedimentos	Acervo	
	Exemplares	
	Autoridades	
	Catalogação	
Gerenciamento de Usuários	Módulos do Usuário	
	Grupos de Usuários	
	Usuários	
Configurações	Instituições	
	Bibliotecas	
	Parâmetros	

Quadro 3: Estrutura do Menu Principal
Fonte: Os autores (2010).

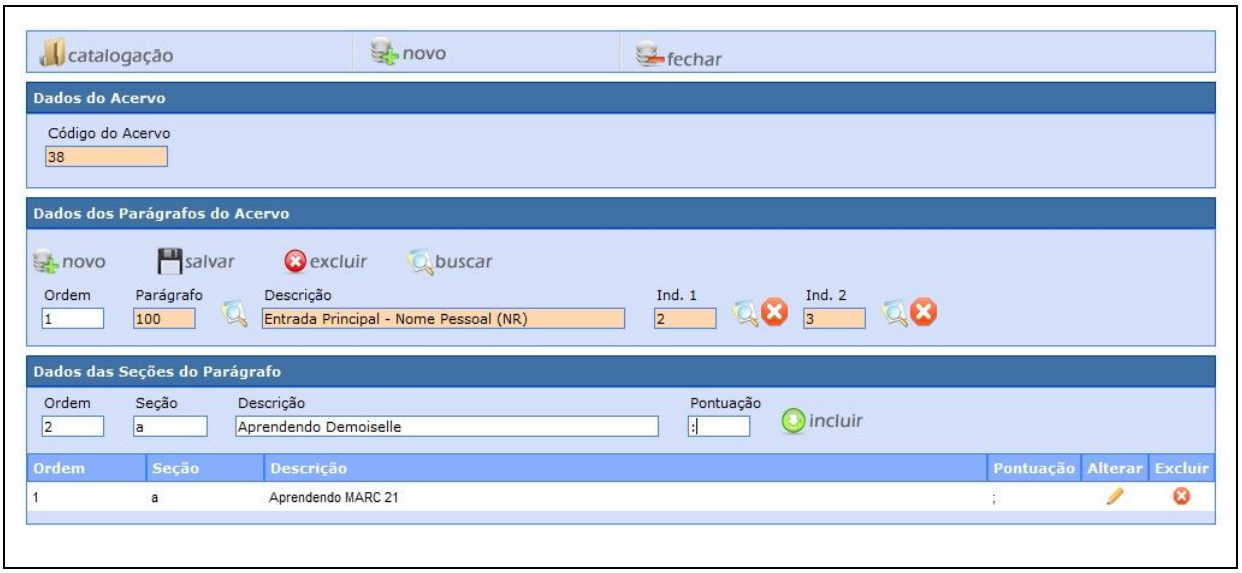
Exceto as telas de catalogação, autoridades e as que compõem a circulação de materiais, as demais telas do sistema seguem um mesmo padrão de lançamento de informações conforme mostra o desenho 18. Na parte superior foi colocado o nome da tela, servindo para situar da melhor forma possível o usuário, e os botões com as principais interações, em que se pode ser inserido um novo registro, listar os que já existem ou fechar a tela. Mais abaixo existe a opção de realizar pesquisas, e seu funcionamento é simples, basta informar o que se deseja buscar, normalmente código e descrição, e posteriormente informar o parâmetro de pesquisa.

Os registros são mostrados ao centro da tela e pelo fato do sistema utilizar paginação, são disponibilizados em grupos de dez tendo a possibilidade de navegar entre eles por meio da barra de navegação. Para realizar uma exclusão ou uma alteração, basta utilizar a opção no lado direito de cada um deles.

Desenho 18: Tela padrão para cadastros do sistema Delibris
Fonte: Os autores (2010).

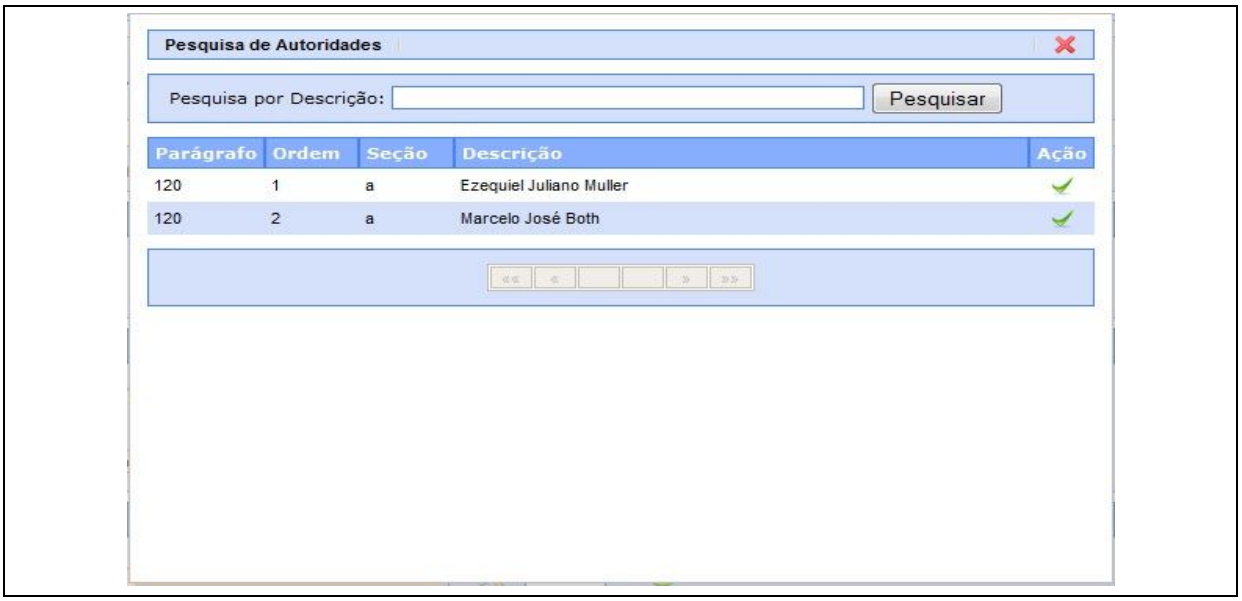
Os processos de catalogação e lançamento de informações de autoridades possuem em comum diversos aspectos. Para efetuar a catalogação o usuário inicialmente deve buscar um número do acervo. Caso não estiver disponível pode-se cadastrar o mesmo através do botão

“Cadastrar” que abre uma tela para uma inserção rápida. Com o acervo selecionado abre a segunda parte da catalogação, o lançamento dos parágrafos e seus sub-campos. Com o botão “Buscar” o usuário pode pesquisar os parágrafos já cadastrados para este acervo e efetuar uma manipulação dos seus dados. Caso deseje inserir um novo, basta usar o botão “Novo”, informar o parágrafo e seus indicadores e, após salvar, efetuar o lançamento das informações dos sub-campos (seções dos parágrafos), conforme mostra o desenho 19.



Desenho 19: Tela de catalogação do sistema Delibris
Fonte: Os autores (2010).

Quando se está realizando o lançamento de informações de sub-campos, caso o parágrafo em questão possui ligação com um parágrafo de autoridade, é disponibilizada uma opção para buscar as informações da autoridade e persistir esta informação no sub-campo. Desta forma, abre-se a tela de pesquisa padrão do sistema, conforme desenho 20, para realizar a seleção do dado.



Desenho 20: Tela de pesquisa padrão do sistema Delibris
Fonte: Os autores (2010).

O processo de circulação de materiais (empréstimo, devolução e reserva) é feito de forma simples. Para realizar uma reserva, é necessário informar o leitor e a obra, a informação do tempo de duração da reserva é definida por meio de parâmetros do sistema. Um empréstimo pode ser feito por meio de uma reserva já existente ou fazendo a sua inserção de forma direta. Para a sua realização basta o usuário/operador informar o leitor e o exemplar que esta sendo retirado, além de informar alguns dados essenciais como a data de retirada e a data prevista para devolução. A devolução é realizada apenas informando o código do exemplar que se encontra emprestado, feito isso automaticamente o seu status é atualizado para disponível.

O *Demoiselle* por meio do JSF disponibiliza diversos componentes tipados, que tornam a inserção de dados mais agradável minimizando erros. Desta forma, buscou-se manter um padrão quanto a imagens e a forma de como é feita a interação do usuário com o sistema Delibris, visando minimizar o impacto da inserção de uma nova forma de trabalho tornando o aprendizado ao sistema e sua aceitação mais fácil.

3.3 CONSIDERAÇÕES FINAIS

Após a análise da referência bibliográfica e a realização da execução do projeto proposto, foi possível apreender e explorar as potencialidades do *Demoiselle Framework*, tais como padronização, reuso de código e métodos, baixo acoplamento, divisão em camadas e padrões de projetos e aplicá-las no desenvolvimento de um sistema que explore a modalidade de *software* livre e que seja capaz de automatizar o processo de catalogação e circulação de materiais realizados no universo das bibliotecas.

O Delibris é uma aplicação que utiliza as camadas propostas pelo *framework*, oferecendo assim, uma dimensão do que o *Demoiselle* disponibiliza para o desenvolvimento de novas soluções. Esta estrutura em camadas facilita a manutenção e a reutilização de códigos, diminuindo o impacto de um aumento do escopo de funcionalidades. Desta forma, a utilização do *Demoiselle* e de seus recursos na implementação do projeto foi vista como um resultado positivo que agregou valor ao sistema desenvolvido.

O desenvolvimento por meio destas camadas proporciona uma maior flexibilidade no tratamento da informação pelo sistema. A camada de persistência resulta em um baixo acoplamento nas camadas superiores, garantindo a integridade dos dados providos do banco de dados, facilitando a integração com tecnologias de persistência como o *Hibernate*. A camada de negócio, por meio de sua estrutura, possibilita a definição da lógica de negócios da aplicação, gerenciando regras pertinentes, como por exemplo, ao formato MARC, realizando a integração com as demais camadas por meio do mecanismo de injeção de dependência. A camada de visão, por meio do JSF, controla a navegação de páginas, sendo responsável pela interação do usuário com o sistema, definindo a entrada e saída dos dados, sendo o seu ator principal o *ManagedBean* com os componentes JSF devidamente mapeados.

No início do desenvolvimento do projeto, sentiu-se dificuldade quanto à utilização do *Demoiselle Framework* devido à falta de uma documentação mais sólida sobre o seu funcionamento e a sua forma de interagir com os *frameworks* que o compõe. Apesar do crescimento constante de sua comunidade ainda existem poucos fóruns em discussão tratando sobre seu funcionamento, o que dificultou a procura por informações. Mas, ao mesmo tempo em

que o projeto foi evoluindo o *framework* também se manteve em constante evolução, desta forma começaram a ser disponibilizados tutoriais e laboratórios com uma abordagem mais prática facilitando a compreensão da forma de se desenvolver com o *Demoiselle*.

Devido ao fato de incorporar, em sua concepção, diversos padrões e tecnologias especialistas, a utilização do *Demoiselle* proporcionou ganhos de produtividade e agilidade na execução do projeto. Em um período de seis meses de estudo e desenvolvimento, tendo duas pessoas envolvidas diretamente com o projeto, foi possível implementar um valor aproximado de 31.000 linhas de código, e conforme demonstra o gráfico 1, 266 classes, 86 interfaces, 135 páginas, 51 tabelas no banco de dados e 28 *packages*.

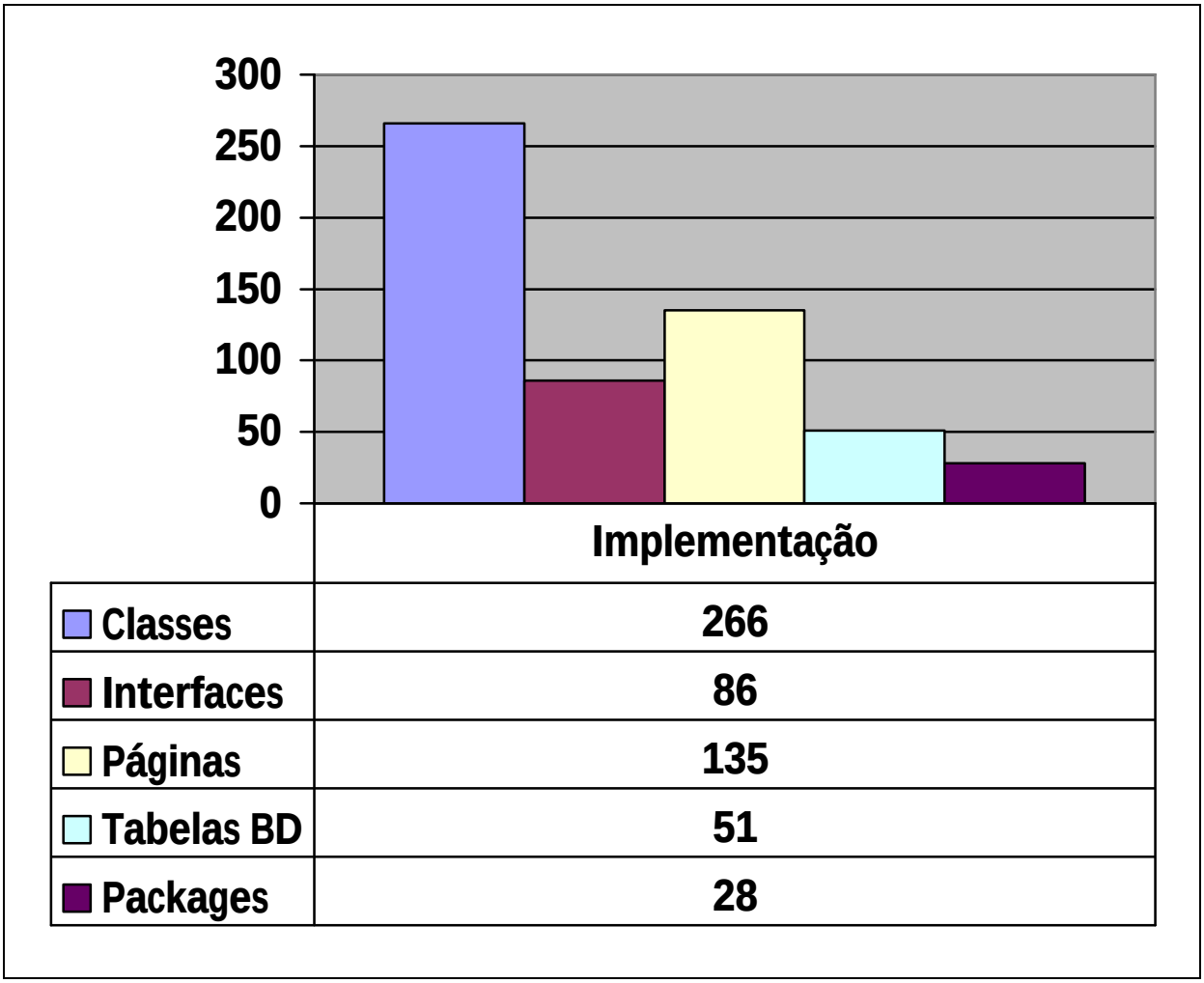


Gráfico 1: Implementação do Delibris
Fonte: Os autores (2010).

Um resultado importante alcançado foi o convite para apresentar o projeto Delibris no I Encontro *Demoiselle* que ocorreu durante o III Congresso Internacional *Software* Livre e Governo Eletrônico (CONSEGI) na cidade de Brasília – DF, durante o período de 18 a 20 de agosto de 2010 (apêndice A e apêndice B). A oportunidade surgiu por meio da troca de informações com a equipe de desenvolvimento da Serpro para sanar dúvidas sobre o *Demoiselle*.

Ao tomar conhecimento do projeto Delibris, foi realizado o convite para apresentar o mesmo na modalidade de caso de sucesso de utilização do *framework*, dividindo espaço com mais três projetos apresentados respectivamente pela Agência Estadual de Tecnologia da Informação do Governo de Pernambuco (ATI), Secretaria do Tesouro Nacional (STN) e o próprio Serviço Federal de Processamento de Dados (Serpro).

Este evento proporcionou uma troca de experiências e informações entre a comunidade *Demoiselle*. Para o projeto Delibris isto foi extremamente importante, pois proporcionou a sua divulgação ganhando espaço no site oficial do *framework* (www.frameworkdemoiselle.gov.br) (apêndice C) na modalidade de caso de sucesso, ficando disponível para toda a comunidade a apresentação, projeto, artigo e um vídeo sobre o funcionamento do sistema.

É importante destacar que várias pessoas ligadas ao desenvolvimento do *Demoiselle* demonstraram interesse em contribuir de uma forma direta ou indireta para a continuidade do projeto para que, desta forma, ele possa automatizar diversas bibliotecas em todo o Brasil.

4 CONCLUSÃO

O desenvolvimento de uma solução informatizada para gerenciamento do acervo bibliotecário proporciona uma melhoria na qualidade dos serviços prestados e diminui a utilização de material impresso no armazenamento dos dados. O Delibris minimiza a possibilidade de erros, elimina o retrabalho e disponibiliza um acesso rápido e eficaz às informações.

No que tange a compreensão das práticas de biblioteconomia, por meio de visitas realizadas a Biblioteca Pública de São Miguel do Oeste para coleta de requisitos e troca de informação sobre o funcionamento dos processos bibliotecários que ocorrem nesta instituição, bem como conversas com a bibliotecária da Unoesc, Sra. Marilene Chaves Furtado, foi possível obter uma visão geral e compreender de forma significativa as regras de negócio para uma biblioteca e as formas de catalogação do acervo.

Neste sentido, é possível destacar a busca por informações e o aprendizado a cerca do formato de catalogação MARC (*Machine Readable Cataloging*), que agregou valor ao sistema Delibris, possibilitando que os registros bibliográficos possam ser lançados em um formato utilizado em vários países.

Possibilitar o lançamento do material bibliográfico em um formato internacional, oferecendo uma interface amigável ao usuário para consulta destas informações, faz do Delibris um *software* em potencial para o gerenciamento de pequenas, médias e grandes instituições bibliotecárias.

Ao adotar uma metodologia de desenvolvimento por meio de tecnologias livres, foi possível reduzir custos, pois não houve a necessidade de se adquirir licenças de *softwares*, além de possibilitar uma independência de fornecedores. Ao desenvolver o Delibris seguindo esta modalidade, é permitida uma disseminação de conhecimento, possibilitando que demais pessoas possam beneficiar e ser beneficiadas com o seu uso.

Referente ao auxílio da metodologia ágil *Scrum* no desenvolvimento do projeto, a adoção de algumas de suas técnicas, como por exemplo, o *product backlog* e o quadro *Kanban* (apêndice D), proporcionaram ganhos de produtividade, facilitando a visualização e acompanhamento do

desenvolvimento do sistema. Em um primeiro momento a intenção era utilizar o *Scrum* de forma completa para gerenciamento dos ciclos de desenvolvimento, mas isto não foi possível, pois se percebeu que para isso acontecer seria necessário um fluxo contínuo de desenvolvimento respeitando algumas regras da metodologia, quanto a reuniões e *sprints*. Como isso não seria possível pelo fato do desenvolvimento ocorrer em horários não convencionais, a adoção completa prejudicaria o desenvolvimento do projeto.

Tendo como ponto de partida um nível de conhecimento acadêmico do paradigma orientado a objeto e da linguagem de programação Java, ao realizar pesquisas e desenvolver a aplicação, foi possível obter um conhecimento maior sobre estas tecnologias e perceber que é possível trabalhar em um alto nível de abstração e, além do mais, a reutilização de código oferece ganhos de produtividade sendo possível desenvolver sistemas complexos em um espaço de tempo menor.

A utilização do *Demoiselle Framework* no desenvolvimento do Delibris foi vista de forma muita positiva. O fato de o *framework* ser uma ferramenta de código aberto, totalmente livre, visando ser a base para a manutenção das mais diversas instituições do Governo Federal, possibilita a qualquer pessoa que detenha o conhecimento sobre o seu funcionamento possa se beneficiar com a sua utilização. A sua arquitetura faz com que seus componentes tenham um ciclo de vida independente facilitando um desenvolvimento colaborativo e seus *frameworks* base garantem transparência no desenvolvimento de novas soluções.

O *Demoiselle* possui como premissas, ser fácil de usar e garantir uma maior produtividade e, neste sentido, foi possível perceber que a curva de aprendizado sobre o seu funcionamento foi rápida e a simplificação de processos resultou no desenvolvimento mais objetivo, oferecendo assim, ganhos de produtividade.

Ao fazer uso do padrão de arquitetura MVC (*Model-View-Controller*) foi possível perceber uma maior facilidade quanto à manutenção da aplicação, possibilitando adicionar, alterar ou retirar funcionalidades do sistema de forma mais simples e eficaz, sem afetar o sistema como um todo. O uso do *Hibernate* na camada de persistência possibilitou que a aplicação, baseada no modelo orientado a objetos, manipulasse as informações e, de uma forma menos complexa, armazenasse as mesmas em um banco de dados relacional.

4.1 RECOMENDAÇÕES PARA TRABALHOS FUTUROS

No início do desenvolvimento do Delibris, um dos objetivos visados era possibilitar a sua continuidade mesmo após a conclusão do curso. Desta forma, ao longo da evolução do projeto, houve uma busca por ferramentas e métodos para auxiliar neste plano de continuidade, bem como um acompanhamento das novas tecnologias e funcionalidades disponibilizadas pela Serpro a cerca do *Demoiselle Framework*. Neste sentido, seguem algumas diretrizes sobre o futuro do Delibris:

- Gerenciar a aplicação nos ambientes instalados fazendo uso do componente *Demoiselle Monitoring*. Este componente tem por objetivo, fornecer mecanismos que possibilitem resposta a requisições (*polling*) e envio de notificações (*trapping*) para servidores de monitoramento, como o SNMP e o *Zabbix*;
- Gerenciar tarefas agendadas por meio do *Demoiselle Scheduler*;
- Por meio do *Demoiselle Common*, facilitar o aproveitamento de tarefas rotineiras como imprimir dados de um objeto, validar informações, formatar entrada de dados em diversos locais da aplicação;
- Disponibilizar o projeto no *SourceForge.net* para toda a comunidade *Demoiselle* interessada em contribuir no seu desenvolvimento, tornando o Delibris um *software* de qualidade para gerenciamento não apenas de uma, mas de várias bibliotecas no Brasil;

Como recomendação para trabalhos futuros abordando o *Demoiselle* é possível citar a possibilidade de inserção do *framework* em um contexto de computação em nuvem (*cloud computing*). Neste sentido, um primeiro passo já foi dado com a possibilidade de hospedar aplicações *Demoiselle* na nuvem consumindo os recursos deste ambiente. O próximo passo é disponibilizar ambientes de desenvolvimento e produção, bem como *webservices* baseados no *framework* poderão ser consumidos por meio de uma nuvem.

REFERÊNCIAS

ARAUJO, Daniel. **SCRUM: Gerenciamento ágil de projetos de software.** 2009. Disponível em: <http://pmpmobile.com/?tag=gerenciamento-de-projetos>. Acesso em: 20 out. 2009.

BAUER, Christian; KING, Gavin. **Hibernate em Ação.** 2. ed. Rio de Janeiro: Ciência Moderna, 2005. 530 p.

BRASIL. Instituto Brasileiro de Geografia e Estatística. **Pesquisa de Informações Básicas Municipais - Gestão Publica 2005:** Ponto a ponto, o IBGE mostra um país de bordadeiras. 2005a. Disponível em: <http://www.ibge.gov.br/home/presidencia/noticias/noticia_impressao.php?id_noticia=744>. Acesso em: 01 set. 2009.

BRASIL. Ministério da Fazenda. Serviço Federal de Processamento de Dados. **Software Livre.** 2009b. Disponível em: <http://www.serpro.gov.br/tecnologia/software-livre>. Acesso em: 14 out. 2009.

BRASIL. Ministério da Fazenda. Serviço Federal de Processamento de Dados. **Demoiselle Framework.** 2009c. Disponível em: <http://www.frameworkdemoiselle.gov.br/>. Acesso em: 29 ago. 2009.

BRASIL. Vanderson Botelho. Ministério da Fazenda. Serviço Federal de Processamento de Dados. **Desenvolvimento Web com Framework Demoiselle versão 1.0.** 2009d. Disponível em: <http://demoiselle.svn.sourceforge.net/viewvc/demoiselle/framework/trunk/docs/others/tutorial/Demoiselle-Tutorial-Modulo01-Arquitetura-Apresentacao.pdf>. Acesso em: 29 ago. 2009.

BRAUDE, Eric. **Projeto de Software:** Da programação à arquitetura: uma abordagem baseada em Java. Porto Alegre: Bookman, 2005. 619 p.

CÔRTE, Adelaide Ramos e et al. **Automação de bibliotecas e centros de documentação:** o processo de avaliação e seleção de softwares. 1999. Disponível em: <http://www.scielo.br/scielo.php?pid=S0100-19651999000300002&script=sci_arttext&tlng=es>. Acesso em: 24 out. 2009.

CRUZ, Anamaria da Costa; MENDES, Maria Terezinha Reis; WEITZEL, Simone da Rocha. **A biblioteca: o técnico e suas tarefas.** 2. ed. Niterói: Intertexto, 2004.

DESTRO, Daniel. **Implementando Design Patterns com Java.** 2004. Disponível em: <http://www.guj.com.br/article.show.logic?id=137>. Acesso em: 12 out. 2009.

FERREIRA, Rodrigo Lopes. **Introdução ao desenvolvimento ágil com Scrum.** São Paulo, 2009.

FREE SOFTWARE FOUNDATION. Free Software Foundation. **The Free Software Definition.** 2009. Disponível em: <http://www.fsf.org/licensing/essays/free-sw.html>. Acesso em: 17 out. 2009.

GALANTE, Alan Carvalho; MOREIRA, Elvis Leonardo Rangel; BRANDÃO, Flávio Camilo. **Banco de Dados Orientado a Objetos:** Uma Realidade. 2005. Disponível em: <http://www.fsma.edu.br/si/edicao3/banco_de_dados_orientado_a_objetos.pdf>. Acesso em: 16 out. 2009.

GONÇALVES, Edson. **Desenvolvendo Aplicações Web com JSP, Servlets, Javaserver Faces, Hibernate, EJB 3 Persistence e Ajax.** Rio de Janeiro: Editora Ciência Moderna, 2007. 736 p.

GUTIERREZ, Regina Maria Vinhais; ALEXANDRE, Patrícia Vieira Machado. **COMPLEXO ELETRÔNICO:** INTRODUÇÃO AO SOFTWARE. 2004. Disponível em: <http://ce.desenvolvimento.gov.br/SOFTWARE/BNDES%20-%20SW.pdf>. Acesso em: 18 out. 2009.

JOHNSON, Rod. **Expert one-on-one:** J2EE desing and development. Indianápolis - USA: Wrox, 2003. 742p.

LAURINDO, Fernando José Barbin et al. **O Papel da Tecnologia da Informação (TI) na Estratégia das Organizações.** 2001**.** Disponível em: <http://www.scielo.br/pdf/gp/v8n2/v8n2a04.pdf>. Acesso em: 01 set. 2009.

LIMA, Adilson da Silva. **UML 2.0:** Do Requisito a Solução. 2. ed. São Paulo: Editora Érica, 2007. 326 p.

LISBOA, Flávio Gomes da Silva. Made in Brazil: Padronização e Reuso de Aplicações Web com Demoiselle Framework. **Revista Mundo Java**: jruby da Web ao Desktop, Curitiba, n. 36, p.8-17, 01 jul. 2009a. Bimestral.

LISBOA, Flávio Gomes da Silva. **Demoiselle Framework.** 2009b. Disponível em: <http://www.frameworkdemoiselle.gov.br/menu/framework/apresentacoes/apresentacao-conip-2009>. Acesso em: 08 out. 2009.

MACIAS, Ananda De Medeiros. **Framework de desenvolvimento:** visão geral. 2008. Disponível em: <http://www.serpro.gov.br/clientes/serpro/serpro/imprensa/publicacoes/tematec/2008/ttec92_a>. Acesso em: 29 set. 2009.

MORALEZ, Guilherme Bacellar. **Design Patterns: As long as it be.** 2006. Disponível em: <http://www.linhadecodigo.com.br/Artigo.aspx?id=1176>. Acesso em: 14 out. 2009.

NUNES, Krishnamurti Lelis Lima Vieira. **Aspectos Sociais do Uso do Software Livre.** 2003. Disponível em: <http://im.ufba.br/pub/MATA68/Tema5/SoftwareLivre.pdf>. Acesso em: 14 out. 2009.

OLIVEIRA, Bernadette Trzeciak de; SIENNA, Maria Marta. **Leitura pública: Um estudo de caso na biblioteca pública do Paraná.** 2007. Disponível em: <http://www.cerlalc.org/picbip/Boletin_Brasil/port/picbip25_lectura_2.html>. Acesso em: 23 out. 2009.

PENDER, Tom. **UML a Bíblia.** Rio de Janeiro: Editora Campus, 2004. 711 p.

PINHO, Antônio Carlos; MACHADO, Ana Lúcia. **História das Bibliotecas:** Origens. 2003. Disponível em: <http://www.mundocultural.com.br/index.asp?url=http://www.mundocultural.com.br/artigos/Colunista.asp?artigo=635>. Acesso em: 01 fev. 2009.

PRESSMAN, Roger S.. **Engenharia de software.** 6. ed. São Paulo: Mcgraw-hill, 2006. 719 p.

RIBEIRO, Rejane Maria Rosa; PASSOS JUNIOR, Jorge Fernando Guimarães. CATALOGAÇÃO AUTOMATIZADA COMERCIAL: PADRÃO MARC 21. Disponível em: <http://www.sibi.ufrj.br/snbu/snbu2002/oralpdf/122.a.pdf>. Acesso em: 04 jul. 2010.

RINALDI. **Porque usar MVC.** 2009. Disponível em: <http://grifemidia.com.br/noticias-grife/2009/02/17/porque-usar-mvc/>. Acesso em: 14 out. 2009.

SAM-BODDEN, Brian. **Desenvolvendo em Pojos:** do Iniciante ao Profissional. Rio de Janeiro: Alta Books, 2006. 356 p.

SAUVÉ, Jacques Philippe. **Frameworks:** O que é um framework. 2009. Disponível em: <http://www.dsc.ufcg.edu.br/~jacques/cursos/map/html/frame/oque.htm>. Acesso em: 29 set. 2009.

ZEMEL, Tárcio. **Padrões de projetos (ou design patterns): o que são e para que servem e qual a sua implicação de uso.** 2009a. Disponível em: <http://codeigniterbrasil.com/passos-iniciais/padroes-de-projeto-ou-design-patterns-o-que-sao-para-que-servem-e-qual-sua-implicacao-de-uso/>. Acesso em: 12 out. 2009.

ZEMEL, Tárcio. **MVC(Model-View-Control).** 2009b. Disponível em: <http://codeigniterbrasil.com/passos-iniciais/mvc-model-view-controller/>. Acesso em: 14 out. 2009.

APÊNDICES

APÊNDICE A – Apresentação do Delibris no CONSEGI 2010

Marcelo J. Both e Ezequiel J. Müller no momento da apresentação.

APÊNDICE B – Orientador, Acadêmicos e Funcionários da Serpro no CONSEGI 2010

Professor Roberson J. F. Alves (Orientador), Serge Rehem (Serpro), Marcelo J. Both, Antônio Tiboni (Serpro), Ezequiel J. Müller e Flávio Gomes da Silva Lisboa (Serpro).

APÊNDICE C – Espaço no Site Oficial do *Demoiselle Framework*

Espaço no site oficial do *framework* na categoria de Caso de Sucesso.

APÊNDICE D – Quadro Kanban para Acompanhamento de Tarefas