DELIBRIS: Sistema Bibliotecário Utilizando *Demoiselle Framework*

Ezequiel Juliano Müller[*]

Marcelo José Both[**]

Roberson Junior Fernandes Alves[***]

Resumo

Este artigo tem por objetivo apresentar o sistema para gestão de bibliotecas Delibris, desenvolvido 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 foram necessárias ferramentas de desenvolvimento que possuem como ambiente operacional a *Web*, como a linguagem de programação Java, por meio do *framework* integrador *Demoiselle* do Governo Federal com 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: *Demoiselle*. *Framework*. Biblioteca. Java. Delibris.

[*] Graduando do Curso de Bacharelado em Sistemas de Informação
Unoesc – Campus de São Miguel do Oeste
Rua Oiapoc, 211 – São Miguel do Oeste – SC
ezequieljuliano@gmail.com

[**] Graduando do Curso de Bacharelado em Sistemas de Informação
Unoesc – Campus de São Miguel do Oeste
Rua Oiapoc, 211 – São Miguel do Oeste – SC
marcelo.both@gmail.com

[***] Especialista em Ciências da Computação (UFSC)
Professor do Curso de Bacharelado em Sistemas de Informação
Unoesc- Campus de São Miguel do Oeste
Rua Oiapoc, 211 – São Miguel do Oeste – SC
roberson.alves@unoesc.edu.br

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. Diante disso, o governo brasileiro 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 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 inicial 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.

2 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.”

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).

3 OBJECT-RELATIONAL MAPPING (ORM)

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

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).

Uma ferramenta de destaque capaz de realizar o mapeamento e acesso ao banco de dados é o *Hibernate*. O objetivo da utilização desta ferramenta é facilitar a manipulação de dados inibindo o desperdício de tempo com tarefas primorosas. Dentre os grandes benefícios do *Hibernate* encontra-se o fato de se deixar o desenvolvedor livre para focar sua atenção em problemas de lógica de negócios tornando-se, desta forma, menos complicada a interação com o banco de dados relacional. (GONÇALVES, 2007).

4 MVC – MODELO, VISÃO E CONTROLE

MVC é um padrão arquitetural que separa uma aplicação em várias camadas. Essa divisão ocorre devido ao fato das aplicações terem aumentado a sua complexidade no desenvolvimento, tornando-se indispensável a sua aplicação. 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) (ZERMEL, 2009).

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 provem da camada de modelo (GONÇALVES, 2007).

O contrário também acontece depois de realizadas 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 faz uma representação visual dos dados que provem da camada de modelo e são administradas pelo controle (GONÇALVES, 2007).

5 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.

6 DEMOISELLE FRAMEWORK

O Governo Federal por meio da SERPRO (Serviço Federal de Processamento de Dados) 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."

A plataforma foi denominada em homenagem a um modelo de avião idealizado por Santos Dumont. Segundo Brasil (2009) "[...] 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.

6.1 ARQUITETURA DO DEMOISELLE FRAMEWORK

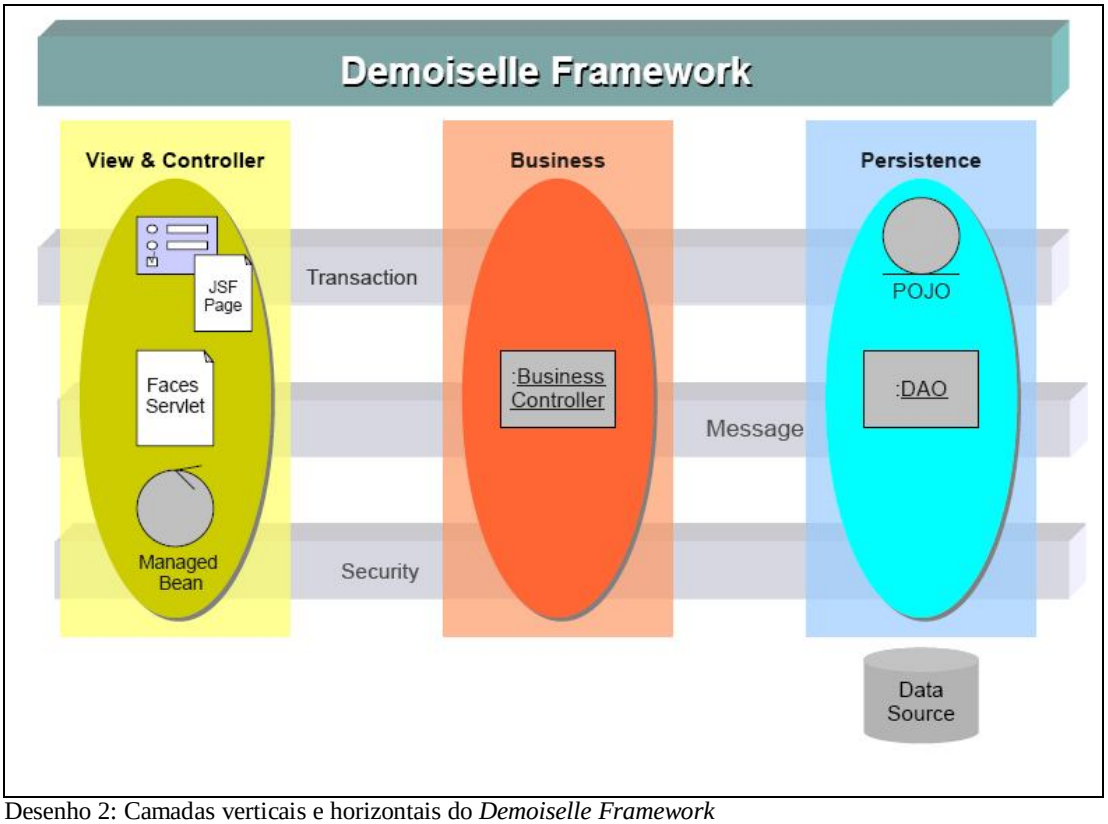
A arquitetura do *Demoiselle Framework* é divida em módulos como podem ser observados no Desenho 1. O módulo *Core* possibilita fazer padronização, extensão e integração de camadas. O módulo *Persistence* é responsável pelo armazenamento e tratamento das informações. O módulo *Util* possui componentes 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).

Desenho 1: Arquitetura do *Demoiselle Framework*
Fonte: Lisboa (2009b).

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, 2009).

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.

Segundo Lisboa (2009a), “Toda essa infraestrutura para as camadas tradicionais constitui o que chamamos de contextos”. No Desenho 2 podem ser observados os contextos em camadas horizontais e a implementação MVC do *Demoiselle* em camadas transversais.



Desenho 2: Camadas verticais e horizontais do *Demoiselle Framework*
Fonte: 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.

7 METODOLOGIA DE DESENVOLVIMENTO

Para o desenvolvimento do sistema, foi adotada uma combinação de técnicas da metodologia ágil de desenvolvimento *Scrum*. Como o *Scrum* tem por característica visar os pontos mais importantes e as principais funcionalidades do negócio 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	

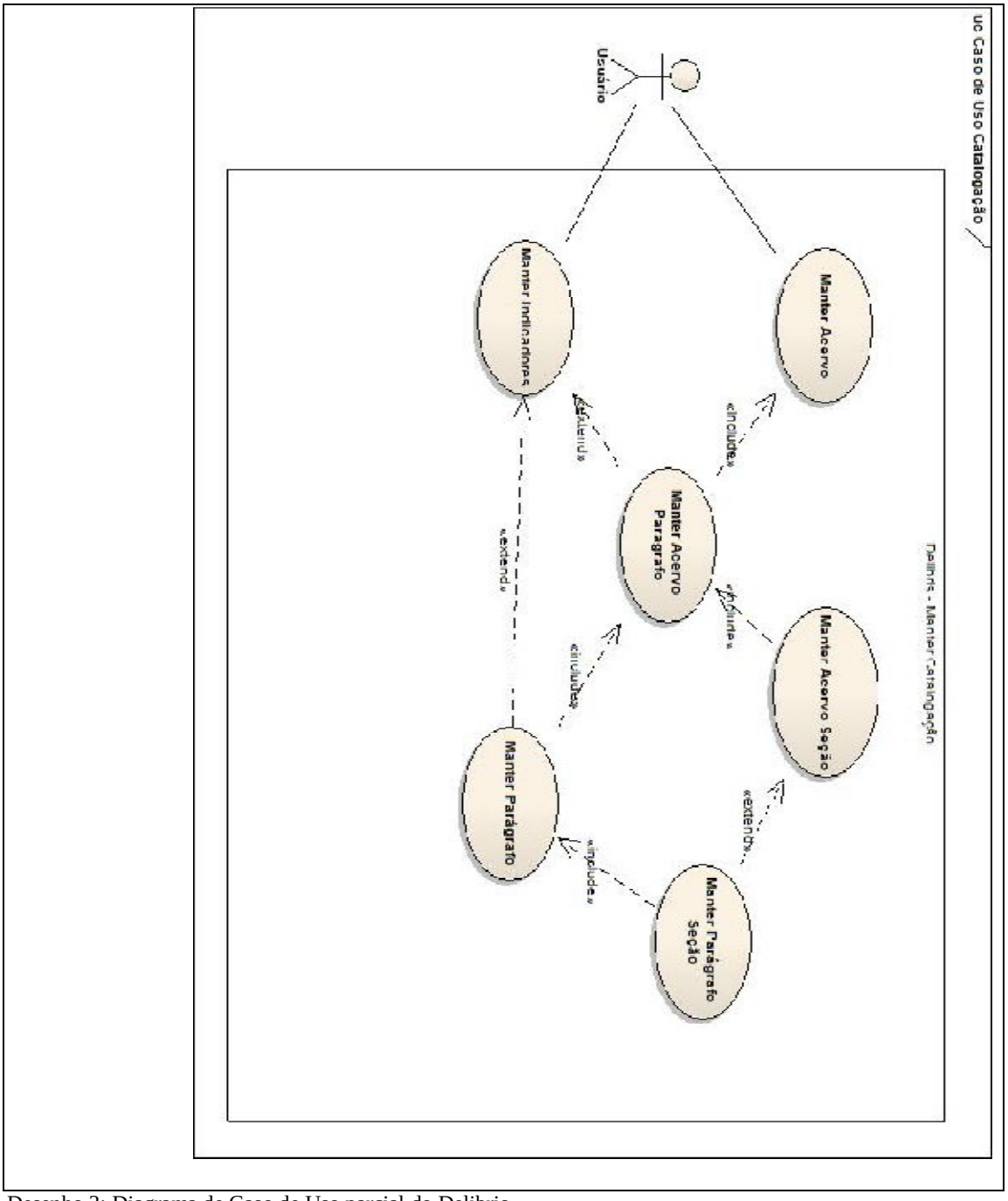
Nome	Restrição	Categoria	Desejável	Permanente
RF 1.1 Cadastros Auxiliares	Os cadastros auxiliares já devem estar cadastrados no sistema para que seja possível efetuar a catalogação.	Segurança	( )	(X)
RF 1.2 Identificação dos Exemplares	Os números gerados automaticamente deverão ser únicos, para assim 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: para possibilitar a separação das funcionalidades do sistema, o aplicativo foi divido em módulos. Estes módulos compreendem: cadastros auxiliares, catalogação, consulta, administração e circulação de materiais.
- Acompanhamento visual através do quadro *Kanban*: todos os requisitos foram colocados no quadro por meio de *post-its* da mesma cor, divididos em: “aguardando, desenvolvendo e concluído”. 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.

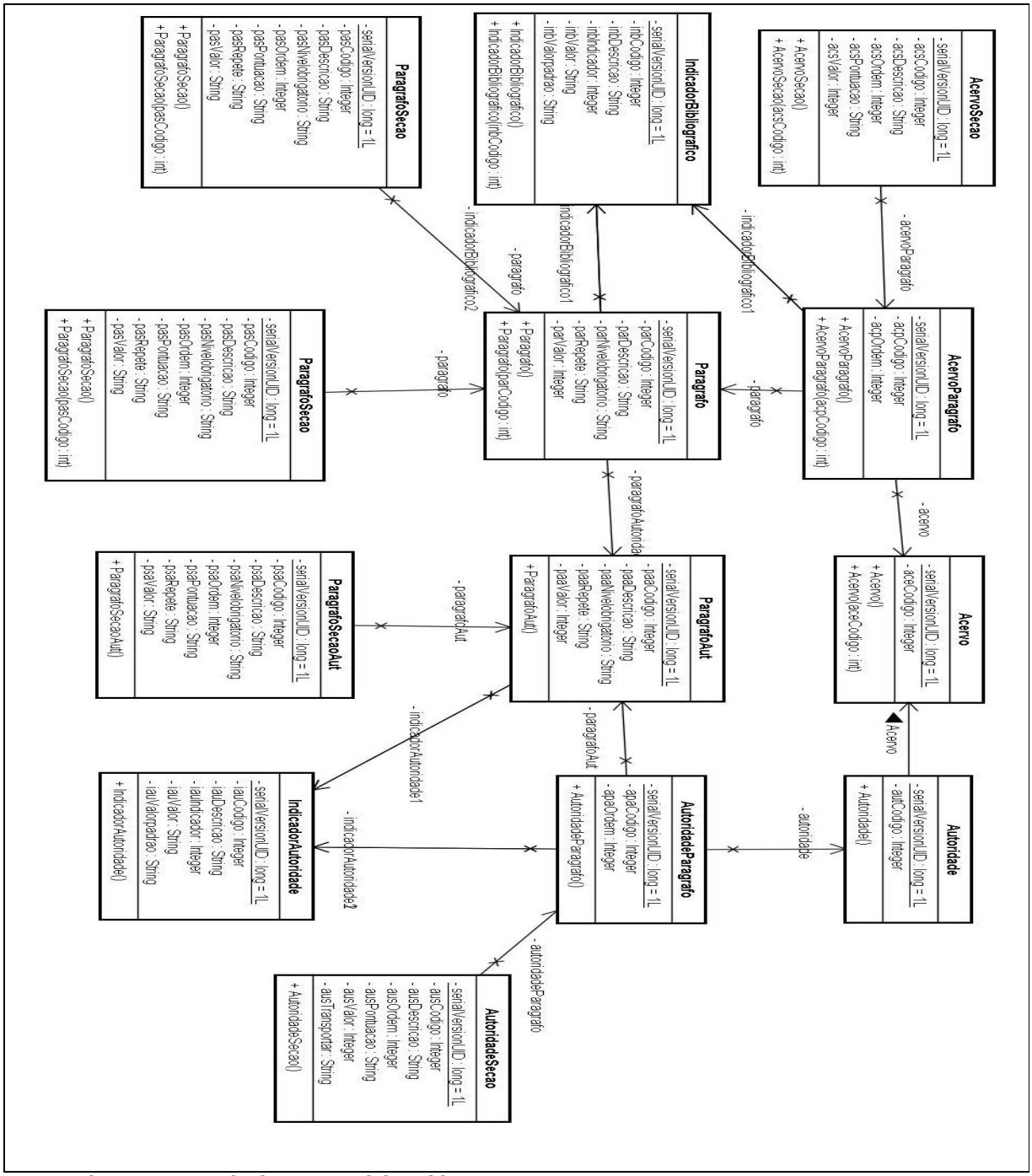
7.1 MODELAGEM E DIAGRAMAS

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.



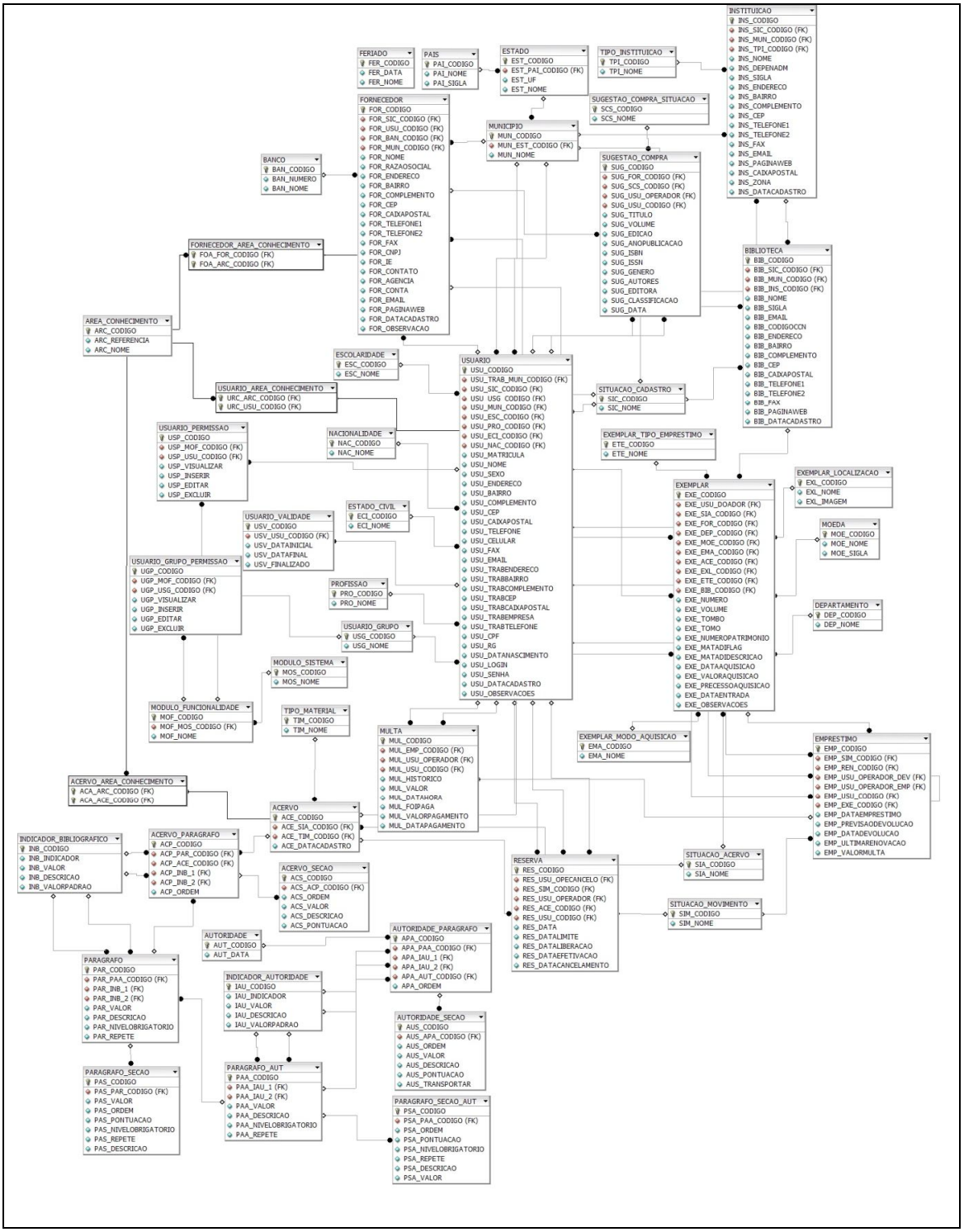
Desenho 3: Diagrama de Caso de Uso parcial do Delibris
Fonte: Os autores (2010).

O Diagrama de Caso de Uso, conforme Desenho 3, 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 4: Diagrama de classes parcial do Delibris
Fonte: Os autores (2010).

O Diagrama parcial de Classes (Desenho 4) 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 5: Diagrama entidade-relacionamento do Delibris
Fonte: Os autores (2010).

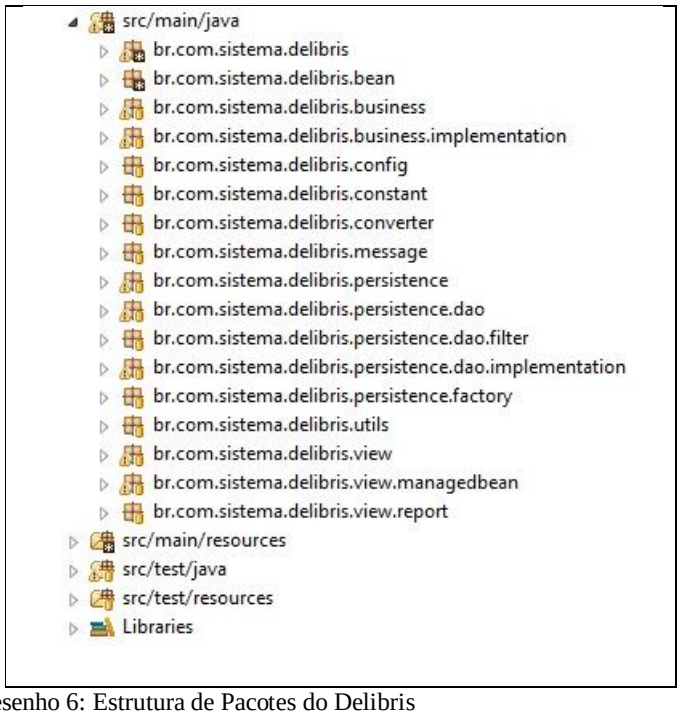
Desenvolveu-se um diagrama entidade-relacionamento conforme os requisitos levantados por meio da ferramenta *DBDesignerFork*. Esta ferramenta possibilita modelar banco de dados de forma visual, conforme mostra o Desenho 5.

8 DELIBRIS

Para o desenvolvimento da aplicação fez-se necessário a utilização da IDE (*Integrated Development Environment*) *open source* Eclipse em sua versão 3.5. Juntamente com o Eclipse, foram instalados os *plug-ins* do *framework Demoiselle*, do *AspectJ* (para a programação orientada a aspectos), do JSF e do SVN *Subversion* (necessário para o controle de versão e gerenciamento de código fonte em trabalhos feitos em equipes). A base de dados por definições do *framework* foi desenvolvida com o *PostgreSQL* 8.4, que é um SGDB (Sistema Gerenciador de Banco de Dados) gratuito e muito robusto para aplicações em geral.

8.1 DELIBRIS - ARQUITETURA

Ao utilizar o *Demoiselle* no desenvolvimento de uma aplicação, como ponto de partida acontece uma divisão do projeto em três camadas distintas: visão, negócio e persistência.



Desenho 6: Estrutura de Pacotes do Delibris
Fonte: Os autores (2010).

A estrutura de pacotes, conforme mostra o Desenho 6, 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. Os

pacotes responsáveis por implementar as interfaces tiveram um papel muito importante no desenvolvimento da aplicação porque abstraem o objeto para os pacotes de implementação.

8.1.1 Camada de Persistência

Esta camada tem por objetivo efetuar o tratamento e manipulação 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).

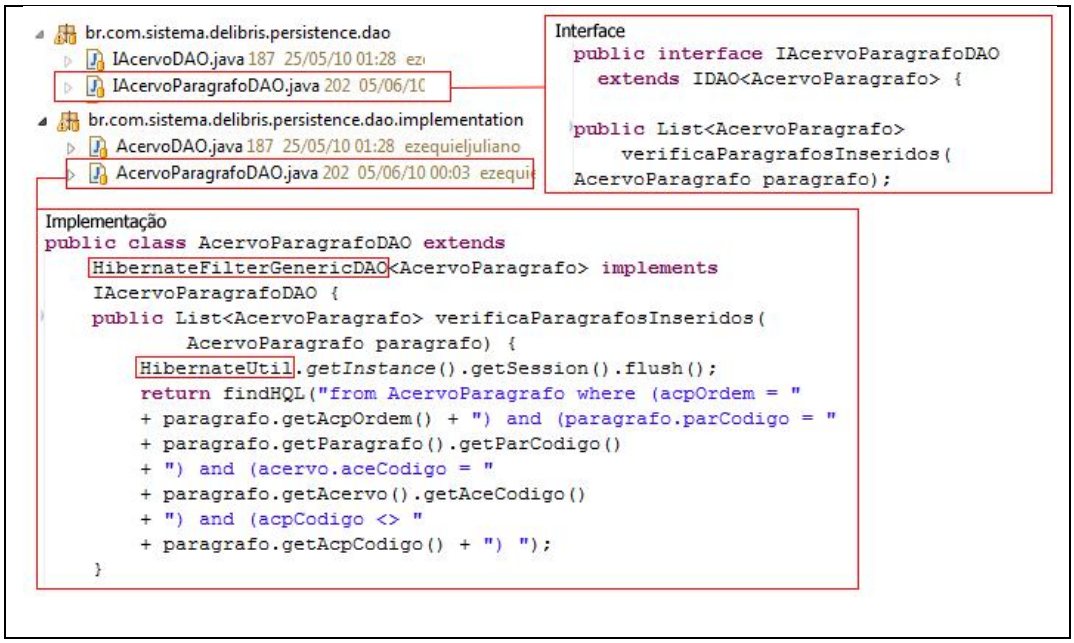
O pacote *Bean* representa as classes do tipo Pojo, que são as entidades da aplicação. Utilizando o *Hibernate* foi realizado o Mapeamento Objeto Relacional com o uso de *Annotation* conforme mostra o Desenho 7.

```
@Entity
public class Acervo implements Serializable,
        br.gov.framework.demoiselle.core.bean.IPojo {
    private static final long serialVersionUID = 1L;

    @Id
    @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")
    private Timestamp aceDatacadastro;

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

    // Bi-Direcional muitos-pra-um associacao com Tipo de Material
    @ManyToOne(fetch = FetchType.EAGER)
    @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 7: Mapeamento com *Annotation*
Fonte: Os autores (2010).

Conforme pode ser acompanhado no Desenho 7, 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 para a camada de persistência que são acessadas através do *HibernateUtil*. O

HibernateFilterGenericDAO tem como objetivo simplificar a criação das interfaces IDAO oferecendo métodos de consulta, paginação, inserção, alteração e exclusão.



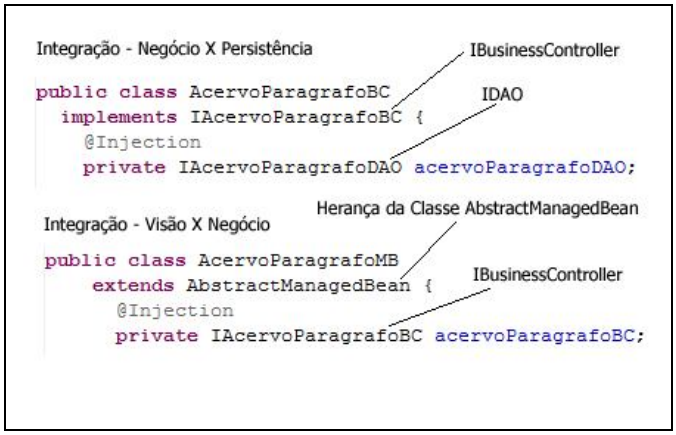
Desenho 8: DAO e *DAO.Implementation* utilizadas no Delibris
Fonte: Os autores (2010).

Ainda na nesta camada foi utilizado o componente *Hibernate Filter* para a geração de filtros de pesquisas. Por meio deste componente, é possível definir campos a serem filtrados aplicando um conjunto de critérios como: *addEquals, addNotEquals, addLike, addLikeIgnoreCase, addBetween, addIsNull*.

8.1.2 Camada de Negócio

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 por meio do mecanismo de injeção de dependência.

Quando a integração ocorre 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 9: 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*). Este componente se destaca por poder ser utilizado em todas as camadas do sistema e isto é possível por meio da utilização da Orientação a Aspectos (AOP). Ele se caracteriza por definir 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 10.

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

Desenho 10: 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*. 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.*

8.1.3 Camada de Visão

A camada de visão é responsável por apresentar as informações aos usuários e controlar suas interações com o sistema. Quando se utiliza o *Demoiselle* no desenvolvimento de uma aplicação o *Java Server Faces* (JSF) é o responsável pelo módulo de visão e controle.

Existem dois arquivos importantes que precisam ser configurados para o funcionamento do JSF: o *web.XML* e o *faces-config.XML*. Nestes também devem ser disponibilizadas as páginas JSP com os componentes JSF mapeados para os *ManagedBeans*.

Por meio de uma requisição HTTP o JSF permite ao usuário a interação com a página no navegador, o *ManagedBean* por meio do componente que se encontra mapeado (Desenho 11) efetua o tratamento desta requisição.

```
<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 11: Mapeamento do *ManagedBean* com JSF
Fonte: Os autores (2010).

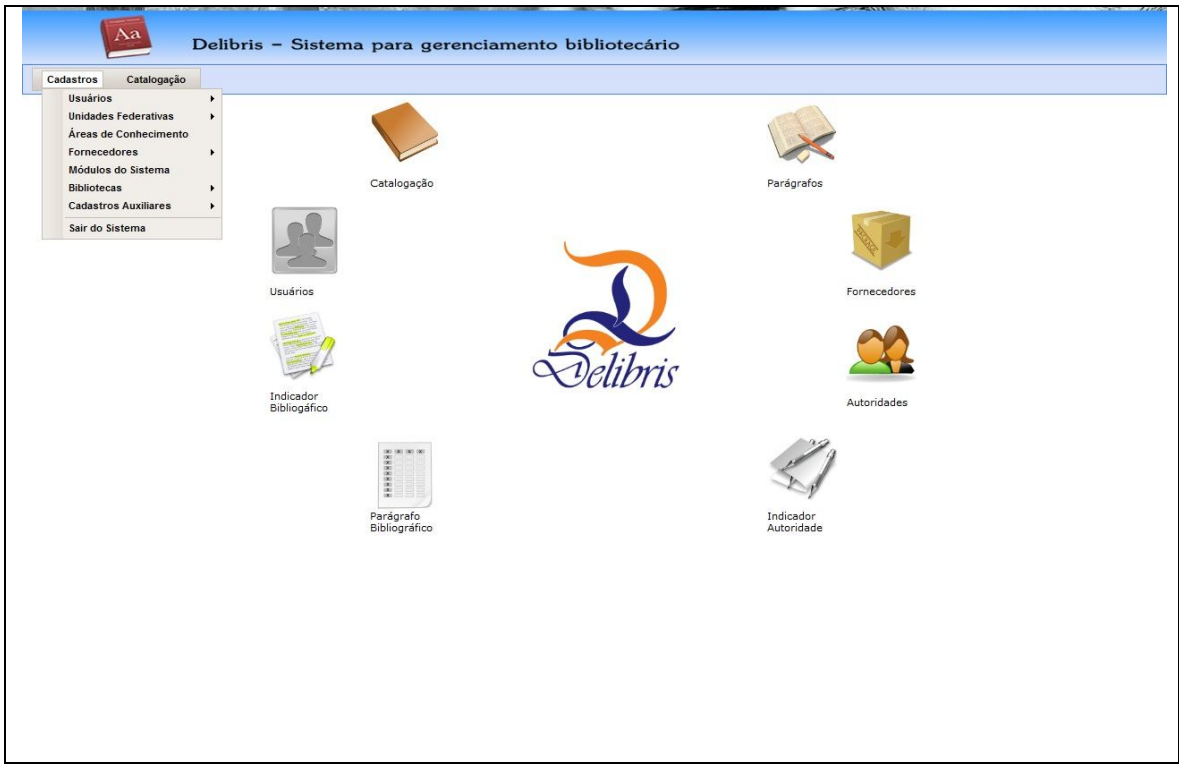
Desta forma, o *ManagedBean* manda o Pojo mapeado para a camada de negócio, que realiza o tratamento das informações e retorna os dados para o *ManagedBean*. Posteriormente os dados são formatados e a página *web* é construída exibindo-os ao usuário.

8.2 DELIBRIS – INTERFACE

A preocupação com a automatização dos processos bibliotecários cresce na mesma proporção em que surgem novas bibliotecas. Frente a isso, desenvolveu-se um sistema capaz de automatizar o processo de catalogação envolvido no universo bibliotecário. O sistema oferece toda a estrutura de cadastros auxiliares e cadastros bases para o gerenciamento do

acervo bibliográfico, além de disponibilizar toda a estrutura de cadastramento do formato MARC para lançamento do acervo e das autoridades.

No Desenho 12 é apresentada a visualização do usuário após o *Login* no sistema, onde o "Menu" central é disponibilizado com ícones grandes destacando as principais funcionalidades do sistema, visando uma maior facilidade na interação com o sistema. O "Menu" superior fica sempre disponível, pois os itens são carregados de forma dinâmica na parte central. Com isso o usuário tem mais facilidade para abrir as demais telas de lançamento de informações do Delibris.

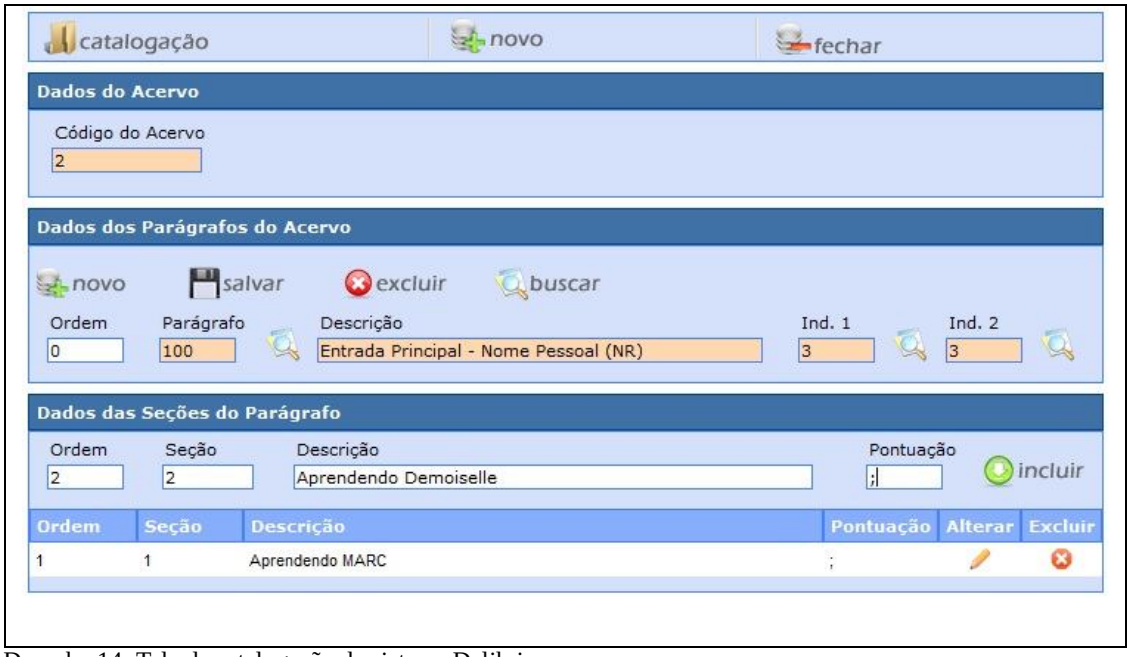


Desenho 12: Tela principal do sistema Delibris
Fonte: Os autores (2010).

O Delibris disponibiliza cadastros auxiliares (Escolaridade, Profissão, Situações entre outros); cadastros base para funcionamento (Instituições, Bibliotecas, Fornecedores entre outros) e referente ao padrão MARC existem os cadastros de indicadores e parágrafos tanto no modo bibliográfico como no modo de autoridades. Exceto a tela de Catalogação e de Autoridades, as demais telas do sistema seguem um mesmo padrão de lançamento de informações conforme mostra o Desenho 13.

Desenho 13: Tela padrão para cadastros do sistema Delibris
Fonte: Os autores (2010).

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 14.



Desenho 14: Tela de catalogação do sistema Delibris
Fonte: Os autores (2010).

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.

9 RESULTADOS PARCIAIS

Como resultado do desenvolvimento, até a presente data, foi possível apreender e explorar as potencialidades do *Demoiselle Framework*, tais como padronização, reuso de código e métodos, baixo acoplamento, divisão em camadas, componentização 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 envolvido no universo bibliotecário.

A grande dificuldade quanto a sua utilização, foi a 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.

Porém, é importante destacar que o *Demoiselle* se torna uma opção muito boa para o desenvolvimento de aplicações que possuem como ambiente operacional a *Web*. A integração de tecnologias especialistas resulta no aumento da produtividade e garante uma maior facilidade na manutenção do sistema com ele desenvolvido. A sua utilização no desenvolvimento do Delibris foi vista de forma muita positiva.

Quanto à implantação, na última visita realizada a Secretária de Cultura de São Miguel do Oeste, ficou definido que seria realizada uma avaliação dos *hardwares* disponíveis hoje na biblioteca e posteriormente formulada uma sugestão de compra de equipamentos para o bom funcionamento do sistema. Esta sugestão foi elaborada e entregue para a Secretaria de Cultura. Entrou-se em contato para verificar o andamento da aquisição dos equipamentos e a informação repassada até o momento é que estes equipamentos ainda não se encontram disponíveis, assim que os equipamentos forem adquiridos será providenciada a implantação do sistema.

10 CONCLUSÃO

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

Destaca-se 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.

A utilização de tecnologias desconhecidas e de um formato internacional de catalogação fez deste trabalho um desafio muito grande para os desenvolvedores. A utilização

da linguagem de programação orientada objetos Java, por meio do *framework* integrador *Demoiselle* do Governo Federal na implementação do sistema, agregou conhecimento aos desenvolvedores possibilitando um crescimento pessoal e profissional.

DELIBRIS: Librarian System Using Demoiselle Framework

Abstract

This article aims to present the Delibris library management system developed in order to 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 use tools development whose operating environment is the Web, such as the Java programming language, through the integrator framework Demoiselle of the Federal Government with the pattern of architectural MVC (Model-View-Controller) and ORM (Object-Relational-Mapping) to ensure that the information stored are used in the form of object in the relational database PostgreSQL.

Keywords: Demoiselle. Framework. Library. Java. Delibris.

REFERÊNCIAS

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

BRASIL. Serviço Federal de Processamento de Dados. Serpro. **Demoiselle Framework.** 2009. Disponível em: <http://www.frameworkdemoiselle.gov.br/>. 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.

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.

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

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.

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.

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