UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CURSO SUPERIOR DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

KEGAN ONOMICHI

SISTEMA WEB PARA COMPOSIÇÃO DE LISTAS DE SIGLAS

TRABALHO DE CONCLUSÃO DE CURSO

PATO BRANCO
2011

KEGAN ONOMICHI

SISTEMA WEB PARA COMPOSIÇÃO DE LISTAS DE SIGLAS

Trabalho de Conclusão de Curso de graduação, apresentado à disciplina de Trabalho de Diplomação, do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, da Universidade Tecnológica Federal do Paraná, Campus Pato Branco, como requisito parcial para obtenção do título de Tecnólogo.

Orientadora: Profa. Beatriz Terezinha Borsoi

PATO BRANCO
2011

ATA Nº: **180**

DEFESA PÚBLICA DO TRABALHO DE DIPLOMAÇÃO DO ALUNO **KEGAN ONOMICHI**.

Às 09:35 hrs do dia 5 de julho de 2011, Bloco S da UTFPR, Campus Pato Branco, reuniu-se a banca avaliadora composta pelos professores Beatriz Terezinha Borsoi (Orientadora), Omero Francisco Bertol (Convidado) e Diego de Carvalho (Convidado), para avaliar o Trabalho de Diplomação do aluno Kegan Onomichi, matrícula 949965, sob o título **Sistema Web para Composição de Listas de Siglas**; como requisito final para a conclusão da disciplina Trabalho de Diplomação do Curso Superior de Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, Coordenação de Informática. Após a apresentação o candidato foi entrevistado pela banca examinadora, e a palavra foi aberta ao público. Em seguida, a banca reuniu-se para deliberar considerando o trabalho **APROVADO**. Às 10:45 hrs foi encerrada a sessão.

Profa. Beatriz Terezinha Borsoi, Dr.
Orientadora

Prof. Omero Francisco Bertol, M.Sc.
Convidado

Prof. Diego de Carvalho, Esp.
Convidado

Prof. Omero Francisco Bertol, M.Sc.
Coordenador do Trabalho de Diplomação

Prof. Edilson Pontarolo, Dr.
Coordenador do Curso

RESUMO

ONOMICHI. Kegan. Sistema *web* para composição de listas de siglas. 2011. 54 f. Monografia (graduação de Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas) - Universidade Tecnológica Federal do Paraná. Pato Branco, 2011.

A abrangência da Internet e a facilidade de acesso são fatores que tem auxiliado a caracterizá-la como um ambiente bastante adequado para executar aplicações relacionadas a negócios. Juntamente com essa expansão de uso surgem novas tecnologias e outras são adaptadas para que possam atender às exigências do novo tipo de usuário *web*. É o usuário acostumado aos recursos das aplicações *desktop*. Isso porque é esse o tipo de aplicação que vinha sendo utilizado na automatização de operações para os setores da indústria, comércio e serviços. Os esforços para atender a essas exigências tem caracterizado as aplicações denominadas *Rich Internet Application* (RIAs) ou aplicações Internet rica. O nome é decorrente, principalmente, dos recursos de interface que essas aplicações possuem. Contudo, essas aplicações também promovem redução do tráfego de rede pela renderização (recomposição apenas de partes específicas da página *web*, as que sofreram alterações). Como forma de exemplificar o uso de tecnologias para implementação de RIAs e facilitar a composição de listas de siglas, abreviaturas e acrônimos em trabalhos acadêmicos, neste trabalho é proposta uma RIA. Por ser *web* a aplicação terá seu acesso facilitado tanto por acadêmicos como por professores e mesmo para a comunidade externa. O aplicativo estará disponível vinculado ao portal da área de informática do campus de Pato Branco da Universidade Tecnológica Federal do Paraná. E tem o propósito de que seja utilizado pelos discentes e docentes como auxiliar na realização de trabalhos acadêmicos, destacando-se os trabalhos de conclusão de curso e os relatórios de estágio curricular.

Palavras-chave: RIA. Rich Internet Application. Linguagem Java. Adobe Flex. Listas de siglas.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTAGENS DE CÓDIGOS

LISTA DE ABREVIATURAS E SIGLAS

ACID	Atomicidade, Consistência, Isolamento, Durabilidade
AIR	*Adobe Integrated Runtime*
AMF	*Action Message Format*
API	*Application Programming Interface*
CRUD	*Create, Retrieve, Update, Delete*
CSS	*Cascading Style Sheets*
DHTML	*Dynamic HTML*
EJB	*Enterprise Java Beans*
HTML	*HyperText Markup Language*
HTTP	*HyperText Transfer Protocol*
IBM	*International Business Machines*
IDE	*Integrated Development Environment*
IIS	*Internet Information Services*
JDBC	*Java Database Connectivity*
JEE	*Java Enterprise Edition*
JME	*Java Mobile Edition*
JNDI	*Java Naming and Directory Interface*
JSE	*Java Standard Edition*
JSF	*Java Server Faces*
JSP	*Java Server Pages*
JTA	*Java Transaction API*
JVM	*Java Virtual Machine*
LGPL	*Lesser General Public License*
MXML	*Maximum Experience Markup Language*
RIA	*Rich Internet Applications*
RPC	*Remote Procedure Call*
SOA	*Service-Oriented Architecture*
SQL	*Structured Query Language*
UFTPR	Universidade Tecnológica Federal do Paraná
UML	*Unified Modeling Language*
XML	*Extensible Markup Language*
XSL	*Extensible Stylesheet Language*
XUL	*User Interface Language*

SUMÁRIO

1 INTRODUÇÃO

Este capítulo apresenta as considerações inicias, com uma visão geral do trabalho, os objetivos, a sua justificativa e a organização do texto.

1.1 CONSIDERAÇÕES INICIAIS

Os trabalhos acadêmicos como relatórios de estágio e monografias de conclusão de curso geralmente possuem uma listagem de siglas e abreviaturas, ou genericamente siglas. Compor essas listas é um trabalho exaustivo. Todas as siglas presentes no texto devem constar de uma listagem e no primeiro uso da sigla no texto deve ter o seu significado explicitado.

É certo que a Internet facilita na localização do significado das siglas e abreviaturas, mas um sistema que permitisse consultar o significado, além de compor uma lista facilitaria em muito o trabalho de alunos e professores.

Considerando que a área de informática do campus Pato Branco da UTFPR (Universidade Tecnológica Federal do Paraná) possui um portal, verificou-se a possibilidade de desenvolver um sistema para Internet que auxiliasse na composição de listas de siglas. A denominação siglas inclui abreviaturas e acrônimos, além das próprias siglas.

A opção pelo desenvolvimento de uma aplicação *web*, mais especificamente de uma RIA, decorre da facilidade de acesso dos usuários ao sistema. A aplicação desenvolvida será instalada em um servidor da Universidade e com acesso pelo portal acadêmico da área de informática. Os usuários podem acessá-lo a partir de qualquer computador que possua acesso à Internet e que tenha o *plugin* do Flash instalado isto porque a aplicação desenvolvida é uma RIA (*Rich Internet Application)* desenvolvida utilizando Flex.

Com esse sistema, o usuário poderá consultar o significado de determinada sigla, compor uma lista e sugerir a inclusão de uma sigla e seu significado. Um usuário com permissões de administrador ou professor valida a sigla sugerida, quando à sua grafia, significado e língua e a inclui no banco de dados. O vínculo com a língua auxilia na definição se a sigla será grafada em itálico (para termos estrangeiros) ou em formato normal para siglas em língua portuguesa.

1.2 OBJETIVOS

O objetivo geral deste trabalho se refere à finalidade do aplicativo desenvolvido e, portanto, exemplifica o ciclo de desenvolvimento de um sistema e o uso de tecnologias e

ferramentas para fazê-lo. Os objetivos específicos complementam o objetivo geral com especificidades do aplicativo desenvolvido, incluem o objetivo de exemplificação de uso das tecnologias empregadas.

1.2.1 Objetivo Geral

- Implementar um sistema *web* para cadastro de siglas e seus respectivos significados.

1.2.2 Objetivos Específicos

- Possibilitar a consulta do significado de uma determinada sigla.
- Permitir a composição de uma lista de siglas com o respectivo significado.
- Possibilitar o cadastro de siglas com significados distintos, associando os significados a áreas distintas.
- Permitir o cadastro de sugestões de siglas que serão validadas por um usuário com permissão para isso e incluídas no banco de dados.

1.3 JUSTIFICATIVA

O desenvolvimento de um aplicativo para a composição de siglas se justifica pela facilidade que proporcionará aos acadêmicos e professores na composição de listas de siglas, abreviaturas e acrônimos que devem constar em trabalhos acadêmicos como relatórios, monografias, dissertações e teses.

A opção pelo desenvolvimento de uma RIA se deve ao fato de que uma aplicação para Internet ter facilidade de acesso e por prover uma interface com melhores recursos. O aplicativo será disponibilizado para acesso por meio do portal acadêmico da área de informática. As funcionalidades de interface oferecidas por uma RIA tornam o seu uso mais facilitado. Isso porque os recursos de interface que essas aplicações podem ter oferecem facilidade de interação com o aplicativo.

O uso da linguagem Java com Flex para a implementação da RIA que possibilitará o cadastro de siglas e seus significados e composição de listas se deve pelos recursos que essas duas tecnologias juntas oferecem. A interface é tratada como objeto em Flex e a linguagem Java é também orientada a objetos, embora haja diferenças em termos de sintaxe. É assim, uma oportunidade de exemplificar o uso dessas tecnologias.

1.4 ORGANIZAÇÃO DO TEXTO

Este texto está organizado em capítulos, dos quais este é o primeiro e apresenta a ideia e o contexto do sistema, incluindo os objetivos e a justificativa.

O Capítulo 2 contém o referencial teórico que fundamenta a proposta conceitual do sistema desenvolvido. O referencial teórico está centrado em aplicações Internet com interface rica, as denominadas RIA.

No Capítulo 3 estão os materiais e o método utilizados no desenvolvimento deste trabalho, incluindo a elaboração da monografia e a modelagem e implementação do sistema.

O Capítulo 4 contém o sistema desenvolvido, com exemplos de documentação da modelagem e de implementação. A modelagem é exemplificada por documentos de análise e projeto. A implementação é exemplificada pela apresentação do sistema com telas e descrição de suas funcionalidades e ainda por partes da codificação do sistema.

No Capítulo 5 está a conclusão com as considerações finais.

2 APLICAÇÕES PARA AMBIENTE INTERNET COM INTERFACE RICA

Este capítulo apresenta o referencial teórico utilizado para fundamentar o sistema proposto. O capítulo se concentra em aplicações ricas para ambiente Internet. As RIAs seguem o paradigma cliente/servidor, mas diferentemente das aplicações *web* tradicionais, elas são capazes de transferir processamento da interface, da lógica de negócio e de dados para o cliente, utilizando comunicação assíncrona.

2.1 APLICAÇÕES INTERNET RICAS

O uso da Internet para execução de aplicativos, especialmente os voltados para operações comerciais, tem feito com que os usuários busquem novas funcionalidades às mesmas. Dentre essas funcionalidades estão recursos de interface e de prover a integração efetiva de áudio e de vídeo com altos níveis de interatividade (PRECIADO et al., 2005). Essas novas necessidades podem ser decorrentes, pelo menos em partes, do fato de os usuários de aplicativos comerciais, e mesmo outros, estarem acostumados com os recursos de interação oferecidos pelas aplicações *desktop*.

Dentre esses recursos destacam-se os componentes da interface como botões e menus e funcionalidades como a de arrastar-e-soltar. Esses componentes e funcionalidades são bem distintos de uma página *web* baseada em texto e *links*.

As aplicações *web* que procuram prover essas funcionalidades são denominadas *Rich Internet Applications* pelos recursos de interface que possuem e são considerados ricos se comparados às aplicações *web* tradicionais. As RIAs fazem com que o seu uso seja facilitado e oferecem uma interface mais rica em termos de recursos e funcionalidades (LOOSLEY, 2011).

Para Duhl (2003) as RIAs foram propostas com o objetivo de resolver problemas encontrados nas aplicações *web*. Elas provêm interface sofisticada para representar processos e dados complexos, ao mesmo tempo em que minimizam a transferência de dados entre cliente e servidor (FUKUDA, YAMAMOTO, 2008). A aplicação se comunica com o servidor, mas em escala menor porque não são transmitidas páginas inteiras a cada interação do usuário, como ocorrem com páginas *web* tradicionais. O tráfego é reduzido porque apenas as partes da página que sofreram alterações são atualizadas. Isso sugere que os clientes que possuem conexões de baixa largura de banda (por exemplo, conexões *dial up*) podem

executar aplicações Internet ricas (MULLET, 2003).

Em uma RIA, ao cliente é atribuída uma parte dos dados da computação, assim o usuário pode realizar uma interação complexa com a interface sem invocar o servidor, a menos que dados sejam trocados (COMAI, CARUGHI, 2007). Além disso, quando a interação do usuário requer uma requisição com resposta do servidor para atualizar dados, o cliente pode obter os dados seletivamente somente da informação que precisa ser alterada e reexibindo o conteúdo modificado.

O Quadro 1 compara as características de uma aplicação *desktop* (incluindo cliente/servidor), uma aplicação *web* convencional (baseada em HTML (*HyperText Markup Language*) e HTTP (*HyperText Transfer Protocol*)) e uma RIA, realçando o seu potencial. Para Bozzon e Comai (2006), esse potencial se baseia no modelo de distribuição da *web*, interfaces melhoradas e redução de sobrecarga de comunicação. Esses benefícios são relevantes para uma variedade de tarefas, como por exemplo, ordenação e filtragem de dados e renderização visual complexa.

Característica	**Cliente/Servidor (Desktop)**	**Web tradicional**	**RIA**
Cliente universal (*browser*)	Não possui	Possui	Possui
Instalação do cliente	Complexa	Simples	Simples
Capacidades de interação	Rica	Limitada	Rica
Lógica de negócio do lado do servidor	Sim	Sim	Sim
Lógica de negócio do lado do cliente	Sim	Limitada	Sim
Requer recarregamento da página completa	Não	Sim	Não
Requisição e resposta frequente ao servidor	Não	Sim	Não
Comunicação do servidor para o cliente	Sim	Não	Sim
Funcionamento desconectado	Sim	Não	Sim

Quadro 1 – Comparação entre aplicações *desktop*, *web* e RIAs
Fonte: Bozzon e Comai (2006, p. 354).

A partir das características de uma RIA, como apresentado no Quadro 1, um conjunto de características pode ser considerado típico a esse tipo de aplicação, para os objetivos de modelagem conceitual e geração de código (BOZZON, COMAI, 2006):

a) RIAs devem suportar processamento do lado cliente e reduzir a comunicação com o servidor ao mínimo;

b) Os dados utilizados pela aplicação podem ser armazenados em níveis de persistência distintos, tanto no lado cliente como servidor;

c) O processamento de dados (criação, alteração e filtragem) pode ocorrer tanto no cliente como no servidor;

d) Deve permitir o uso da aplicação tanto *online* como *offline*.

Apesar das vantagens e da importância que as RIAs vem assumindo no desenvolvimento de aplicações *web*, algumas limitações também são atribuídas às mesmas. Muitas dessas limitações são técnicas e algumas podem ser superadas, outras, contudo são próprias do tipo de aplicação que caracteriza uma RIA.

O Quadro 2 apresenta um resumo das capacidades e das limitações das abordagens para as RIAs. Esse quadro contém exemplos de tecnologias que implementam RIAs. As siglas constantes nesse quadro possuem o seu significado descrito na Lista de Abreviaturas e Siglas.

RIA (tipo e exemplos)	**Recursos**	**Limitações**
Máquina virtual baseada em Flash Exemplos: Adobe Flex, OpenLaszlo	. Podem suportar ampla quantidade de requisições de interações do usuário. . Interação do usuário consistente, confiável e prática. . Ampla semelhança com recursos de interface de aplicações *desktop*.	. Limitadas por *sandbox* no *browser*, por questões de segurança. . Se o computador cliente não possui Flash Player, o usuário deve ter a habilidade de instalá-lo. . Flash *runtime* não disponível para todos os sistemas operacionais ou dispositivos.
Java ou máquina virtual baseada em ActiveX Exemplos: Altio, DreamFactory, Nexaweb	. Podem suportar ampla quantidade de possibilidades de interação com o usuário. . Máquinas virtuais Java disponíveis para uma grande quantidade de sistemas operacionais e dispositivos.	. Limitado por *sandbox* do *browser* por questões de segurança. . Muitos usuários não possuem máquina virtual Java ou a desabilitaram. . Implementações de máquinas virtuais não consistentes entre *browsers* e *desktops*.
Ajax/DHTML Exemplos: Adobe Spry, BackBase, Microsfot Atlas, Nexaweb	. Requisitos adicionais de tecnologia mínimos para o lado cliente ou servidor. . Podem melhorar o conhecimento existente em DHTML, HTML, XML, XSL e JavaScript. . Em teoria, podem suportar ampla faixa de dispositivos e sistemas.	. Limitado por *sandbox* do *browser* e configurações de segurança e *cookies*. . Difícil para garantir consistência, confiabilidade de interação desde que depende de versão do *browser*. . Algumas implementações são limitadas para determinados *browsers* ou por versões.
Desenvolvimento customizado .Net ou Java Exemplos: Microsoft .Net ou Windows Presentation Fundation	. Clientes ricos amplamente customizados. . Não limitados pela *sandbox* do *browser* e questões de segurança. . Suporte para Web Services e SOA. . Forte base para desenvolvedor e suporte de ferramentas.	. Deve lidar com questões de manutenção, desenvolvimento e distribuição em seu próprio código base. . Pode impor requisitos aos clientes.

Quadro 2 – Capacidades e limitações das RIA
Fonte: Schmelzer (2006, p. 10).

Uma das principais razões para utilizar aplicações RIA é que o usuário tenha a impressão que as tarefas estão sendo realizadas no instante em que elas são requisitadas. Assim, as RIAs devem prover meios de mostrar o progresso das operações sendo realizadas, e indicar, mesmo que de maneira aproximada, o tempo restante ou o percentual da operação a ser concluída. O indicador deve ser atualizado periodicamente ou em tempo real.

Existem diversas tecnologias para desenvolvimento e execução de RIAs, incluindo tecnologias proprietárias e gratuitas e de código fonte aberto (como exemplificado na primeira e na segunda colunas do Quadro 2). Há tecnologias mais simples e outras com maior quantidade de recursos incorporados. Uma das formas de desenvolver essas aplicações utilizando a linguagem Java é pela combinação de Flex para a camada de interface e BlazeDS para a transformação dos objetos de interface nos objetos Java da camada de lógica de negócio. Outra forma de implementar essas aplicações é pelo uso de *frameworks*, como o *JavaServer Faces*. Ou, ainda, de tecnologias como o JSP (*Java Server Pages*) para a implementação de interface e também, PHP com Flex, para citar alguns exemplos.

Embora diferentes em muitos aspectos, as tecnologias para implementar uma RIA podem ser classificadas em quatro categorias (BOZZON, COMAI, 2007):

a) Baseadas em *script* – nessas aplicações a lógica no lado cliente é implementada por linguagens de *scritpts*, como JavaScript, e a interface é baseada na combinação de HTML e CSS (*Cascading Style Sheets).*

b) Baseadas em *plugins* – nessas aplicações as renderizações avançadas e o processamento de eventos são realizados por *plugins* no navegador (*browser*) que interpreta XML, programas de objetivos gerais ou arquivos de mídia, como Flash, Flex e Lazlo;

c) Baseadas em *browser* – nas aplicações baseadas em *browser* a interação rica é nativamente suportada por alguns navegadores *web* que interpretam linguagens de definição de interface declarativas, as denominadas XUL (*User Interface Language*);

d) Tecnologias *desktop* baseadas em *web* – nessa categoria as aplicações são baixadas (*downloaded*) pela *web*, mas são executadas fora do navegador. É o que ocorre com as aplicações Java Web Start e Window Smart Client, por exemplo.

3 MATERIAIS E MÉTODO

Este capítulo apresenta os materiais e o método utilizados na realização deste trabalho. Os materiais se referem às tecnologias como linguagens e ferramentas para a modelagem e a implementação do sistema. O método contém as etapas com os principais procedimentos utilizados para o desenvolvimento do sistema, abrangendo do levantamento dos requisitos aos testes.

3.1 MATERIAIS

As ferramentas e as tecnologias utilizadas para as atividades de modelagem, implementação e execução do sistema são:

a) Astah* Community (ASTAH, 2011) para a documentação da modelagem que é baseada na UML;

b) Balsamiq Mockup (BALSAMIQ, 2011) para a prototipação da interface;

c) Adobe Flex (ADOBE, 2011a) versão 4 para implementação da interface do aplicativo;

d) Adobe Flex Builder como IDE (*Integrated Development Environment*) de desenvolvimento da camada de interface;

e) Linguagem Java (DEITEL, DEITEL, 2003) para implementar a camada *backend* contendo as regras de negócio do aplicativo;

f) MyEclipse como IDE de desenvolvimento da linguagem Java;

g) BlazeDS (ADOBE, 2011b) para comunicação entre Flex e Java;

h) AMF (*Action Message Format*) (ADOBE, 2011c) – como formato para serializar objetos gráficos *ActionScript*.

i) MySQL (MYSQL, 2011) para a base de dados;

j) MySQL Workbench (WORKBENCH, 2011) para a modelagem do banco de dados;

k) MySQL Admin (MYSQLADMIN, 2011) para administração do banco de dados;

l) Hibernate (HIBERNATE, 2011) para persistência de dados e o mapeamento objeto relacional;

m) Apache Tomcat (TOMCAT, 2011) para contêiner de aplicação *web*.

3.1.1 Astah* Community

Astah* Commuity é uma ferramenta para modelagem de sistemas com suporte para a UML 2.1 (*Unified Modeling Language*)(ASTAH, 2011). Além da modelagem dos diagramas, a ferramenta oferece ajustes de alinhamento e tamanho dos elementos gráficos, impressão (com a marca d'água da ferramenta) e exportação das imagens (com a marca d'água da ferramenta).

A ferramenta Astah* é apresentada em três versões: a) Astah* Community; b) Astah* UML; c) Astah* Professional. O Quadro 3 apresenta um resumo dos diagramas suportados pelas respectivas versões da ferramenta Astah*.

Diagrama	**Astah* community**	**Astah* UML**	**Astah* professional**
UML 2.x classe, caso de uso, sequência, atividade, comunicação, máquina de estado, componentes, implantação, estrutura de composição, objetos e pacotes	X	X	X
Diagrama de processo Erikson-Penker (representado por diagrama de atividade)	Somente leitura	X	X
Mapa mental	Somente leitura	X	X
Diagrama de entidade e relacionamento (IDEF1x)	Somente leitura	Somente leitura	X
CRUD (*Create, Retrieve, Uupdate Delete*)	Somente leitura	Somente leitura	X
Diagrama de fluxo de dados (com notação DeMarco/Gane-Sarson)	Somente leitura	Somente leitura	X
Fluxograma	Somente leitura	Somente leitura	X
Tabela de requisitos (incluindo hierarquia de nomes, identificadores e texto, com exportação para a planilha de cálculos Excel).	Somente leitura	Somente leitura	X
Diagrama de requisitos	Somente leitura	Somente leitura	X

Quadro 3 – Funcionalidades das versões Astah*
Fonte: http://astah.change-vision.com/en/feature/comparison-table.html

A Figura 1 apresenta a tela inicial (*print screen*) da ferramenta Astah* com o objetivo de explicitar a sua interface principal e as partes que a compõe.

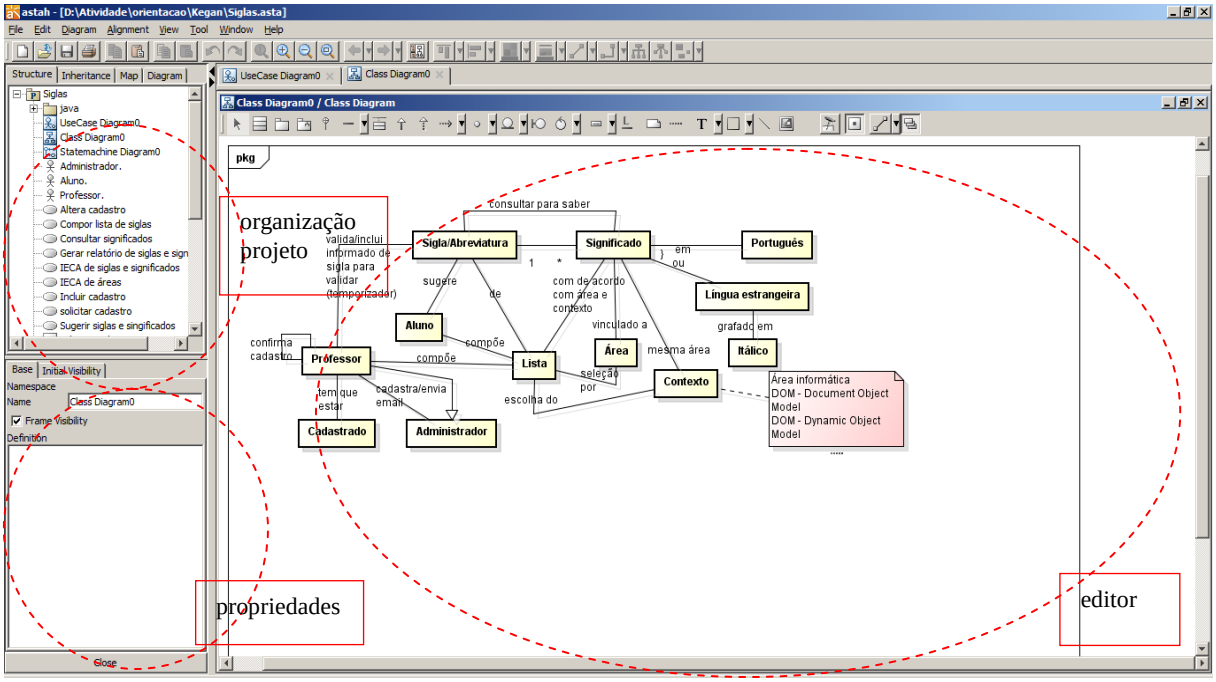

Figura 1 – Ferramenta de modelagem Astah* Community

A IDE da ferramenta Astah* Community está dividida em três partes principais, como pode ser visualizado pelas áreas marcadas na Figura 1:

a) Organização do projeto (área superior à esquerda da Figura 1) - é a área na qual estão as diferentes visões do projeto, são elas: *Structure* que é a árvore de estrutura do projeto, *Inheritance* que exibe as heranças identificadas, *Map* para mostrar todo o editor de diagramas e *Diagram* que mostra a lista de diagramas do projeto.

b) Visão das propriedades (área inferior à esquerda da Figura 1) - é a área na qual podem ser alteradas as propriedades dos elementos dos diagramas. As propriedades de um item selecionado são exibidas e podem ser editadas.

c) Editor do diagrama (área à direita da Figura 1) - é a área na qual são exibidos e construídos os diagramas. Ao ser selecionado um determinado diagrama, dos constantes na lista, o mesmo é carregado e todos os seus elementos gráficos são mostrados nessa área. Para a composição de um diagrama são utilizados os elementos (componentes) constantes na barra superior a essa área de edição. Os elementos que compõem essa barra estão de acordo com o tipo de diagrama escolhido por meio do menu "*Diagram*".

3.1.2 Balsamiq Mockups

Balsamiq Mockups (BALSAMIQ, 2011) da Balsamiq Studio é uma aplicação

desenvolvida em Flash que executa sobre o AIR (*Adobe Integrated Runtime*) da Adobe, o que permite execução em múltiplas plataformas (Mac, Linux, Windows). Essa aplicação é utilizada para desenvolver protótipos ou modelos (*mockups*), como as telas de um sistema e páginas *web*.

O aplicativo é composto basicamente por duas barras de ferramentas. Uma delas (representada na Figura 2) contém o Quick Add, que representa a forma mais fácil de encontrar e adicionar controles, além dos botões básicos como copiar, colar e agrupar.

As Figuras 2 e 3 são *print screen* da tela principal do Balsamiq Mocckups. Na primeira destaca-se a barra de ferramentas e seus componentes para composição de interface gráfica. E na segunda os controles disponíveis.

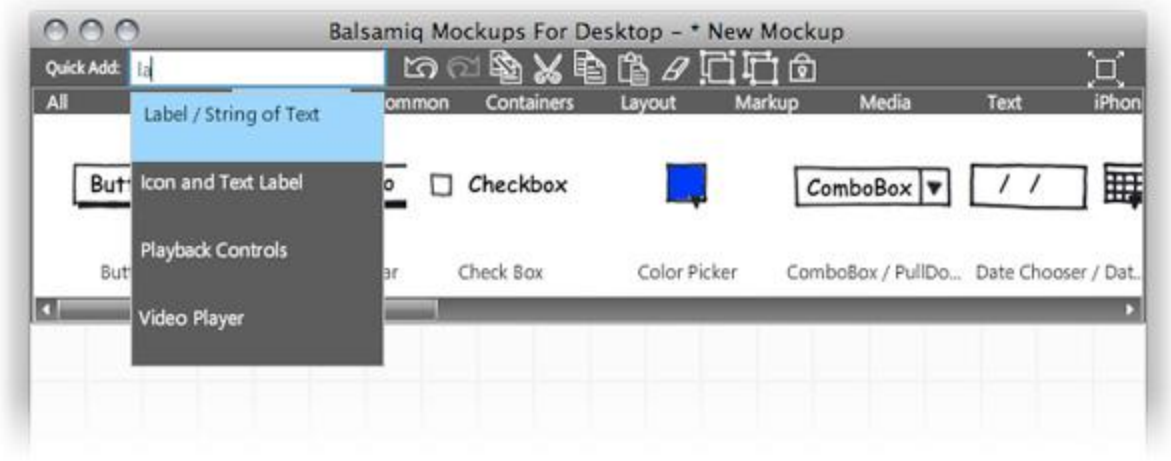

Figura 2 – Barra de ferramentas Quick Add do Balsamiq

A segunda barra de ferramentas (Figura 3) contém todos os 75 controles disponíveis, divididos por categorias como botões, leiaute e mídia. Novos controles não podem ser adicionados diretamente no aplicativo, mas é possível importar imagens e utilizar o recurso Sketch it!, que transforma a imagem em um desenho similar aos outros controles. No site http://mmockupstogo.net podem ser encontrados diversos *templates* gratuitos criados pela comunidade e novos controles podem ser solicitados diretamente aos criadores da ferramenta Balsamiq Mockups.

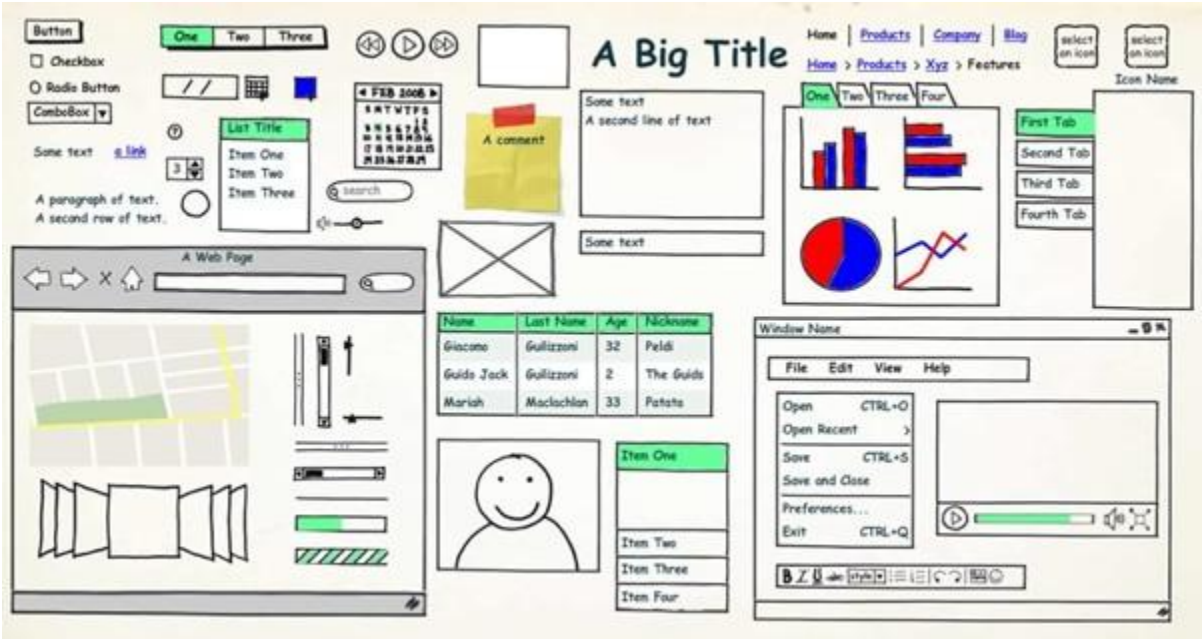

Figura 3 – Barra de ferramentas completa do Balsamiq

Para criar um modelo ou protótipo (*mockup*), basta arrasta e soltar os controles na página. Todos os controles possuem uma caixa de propriedades que permite alterar fonte, cor, alinhamento e outras propriedades exclusivas de cada controle. A ferramenta permite exportar os modelos no formato *png* e ainda copiar e colar diretamente em aplicativos como o Word e o PowerPoint.

3.1.3 Adobe Flex

Flex utiliza uma linguagem de marcação, a MXML (*Maximum Experience Markup Language*) e a ActionScript. ActionScript 3.0 é a linguagem que define a lógica da aplicação que executa no Flash Player. Flex possui vários componentes MXML (FUKUDA, YAMAMOTO, 2008), como: *TextInput*, *DataGrid*, *Button*; e componentes RPC (*Remote Procedure Call*), como: *HttpRequest*, *WebService*, *RemoteObject*. As aplicações Flex possuem a extensão *.mxml* e podem ser criadas em qualquer editor de texto que permita a produção de texto não formatado.

Atualmente seu uso para a *web* é bastante evidenciado pela possibilidade que oferece de produzir interface semelhante às aplicações *desktop*. Essas interfaces são denominadas ricas em decorrência do tipo de recurso (elementos gráficos) que podem apresentar.

Aplicações Flex trabalham com o modelo de arquitetura orientada por eventos. Em

Flex há duas formas de associar eventos com seus manipuladores (FUKUDA, YAMAMOTO, 2008):

a) Cada componente MXML possui seus próprios eventos de envio como nomes de atributos e funções de manipulação de eventos que são definidos como valores de atributos.

b) Cada componente MXML possui um atributo identificador. Assim, é possível identificar cada componente, referir-se o mesmo por meio dos atributos desse identificador. E utilizar um método *addEventListener* para essas associações.

Para que Flex possa integrar-se com Java, é necessário um *gateway* que converta os tipos de dados nativos do Flex para os tipos de dados nativos do Java e vice-versa utilizando o protocolo AMF, por exemplo.

O Flex é executado a partir de uma máquina virtual, o Flash Player. Assim não é necessário implementar na aplicação compatibilidades entre navegadores *web*. A interface é programada com recursos de orientação a objetos, isso provê o reuso de componentes.

3.1.4 Adobe Flex Builder

O Adobe Flex Builder 3™ é uma ferramenta de desenvolvimento baseada no Eclipse, da IBM (*International Business Machines*) (ADOBE, 2009a). O Adobe Flex Builder 3™ é um ambiente para o desenvolvimento da interface do *software* e os seus componentes podem ser facilmente estendidos e customizados. O resultado é uma aplicação que pode executar em ambiente *web*, *desktop* e em dispositivos portáteis que suportem o Flash Player.

O Flex oferece dois ambientes ao desenvolvedor. Um deles é o modo projeto (*design*) que habilita o modo de trabalho visual, permitindo o uso de componentes padrão e de componentes customizados. A interface do projeto é configurada pelo usuário, assim como o comportamento, e os efeitos dos componentes são definidos utilizando a paleta de propriedades. O outro é o modo *source* que exibe o código fonte. Os componentes visuais são transformados em *tags* MXML e seguem uma hierarquia no padrão XML. Arquivos da extensão ‘.as’ não habilitam o modo design. Esses arquivos podem ser classes ou apenas trecho de código.

A Figura 4 apresenta a tela inicial do Flex Builder 3. As principais dessa ferramenta as funcionalidades são listadas e descritas após a figura. Essa figura é um *print screen* do próprio aplicativo Adobe Flex Builder, um ambiente de desenvolvimento Flex.

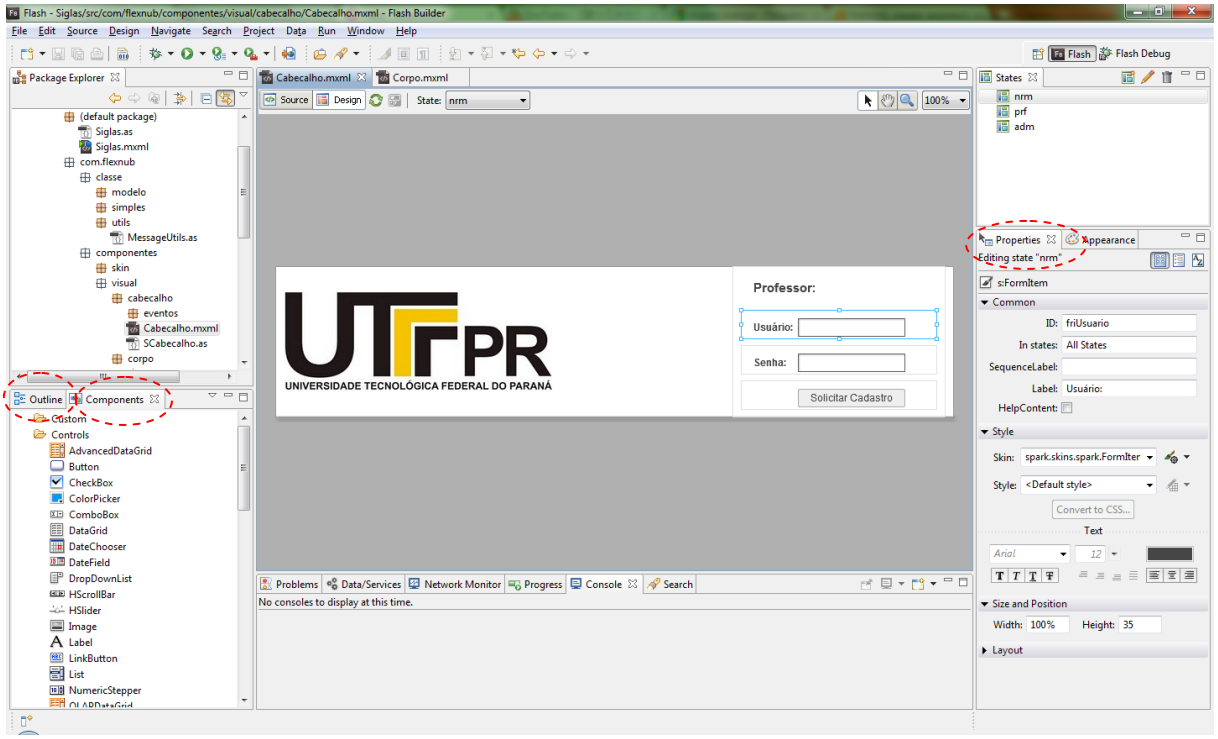

Figura 4 – Tela inicial do Flex Builder

As partes destacadas da Figura 4:

a) *Outline* - caso esteja em modo *source* exibe os métodos e as variáveis, caso seja o modo *design* então exibe a hierarquia dos componentes partindo da aplicação.

b) *Components* – exibe os componentes padrão que acompanham o Flex Builder e também os componentes criados ou estendidos pelo usuário.

c) P*roperties* - exibe as propriedades do objeto visual selecionado, podendo ser customizado facilmente. É possível ao desenvolvedor criar novas propriedades.

3.1.5 Linguagem Java

Java é uma linguagem orientada a objetos e a maior parte dos elementos de um programa Java são objetos. Como exceção cita-se os tipos básicos, como o *int* e o *float*. O código Java é organizado em classes, que podem estabelecer relacionamentos de herança simples entre si.

Um objeto é uma entidade que possui uma identidade que permite individualizá-lo dos demais objetos de maneira inequívoca. Para Booch (1998) um objeto possui estado, exibe algum comportamento bem definido e possui uma identidade única. O estado é definido pelos atributos ou propriedades do objeto, o comportamento são as operações que o objeto realiza e

a identidade é a forma de identificação única de um objeto, como, por exemplo, o código acadêmico de um aluno. Uma classe de objetos descreve um grupo de objetos com propriedades (atributos) similares, comportamento (operações) similares, relacionamentos comuns com outros objetos e uma semântica comum.

Java provê o gerenciamento de memória por meio de *garbage colletor* (coleta de lixo). Sua função é a de verificar a memória de tempos em tempos, liberando automaticamente os blocos que não estão sendo utilizados. Esse procedimento pode deixar o sistema de *software* com execução mais demorada por manter uma *thread* paralela à execução do programa. Porém, esse procedimento otimiza o uso da memória, evitando problemas como referências perdidas e avisos de falta de memória quando ainda há memória disponível na máquina.

Quando um programa Java é compilado um código intermediário é gerado, chamado de *bytecode*. Esse *bytecode* é interpretado pelas máquinas virtuais Java (*Java virtual Machine* - JVM) existentes para a maioria dos sistemas operacionais. A máquina virtual é a responsável por criar um ambiente multiplataforma.

Entre outras funções, a máquina virtual Java também é responsável por carregar de forma segura todas as classes do programa, verificar se os *bytecodes* aderem à especificação JVM e se eles não violam a integridade e a segurança do sistema.

Java é dividida em três grandes edições:

a) *Java Standard Edition* (JSE) - é a tecnologia Java para computadores pessoais, computadores portáteis e arquiteturas com capacidade de processamento e memória consideráveis. Várias APIs (*Application Programming Interface*) acompanham essa versão e tantas outras podem ser obtidas no site da Sun (http://www.sun.com). É com elas que a maioria das aplicações são construídas e executadas.

b) *Java Mobile Edition* (JME) – é a tecnologia Java para dispositivos móveis com limitações de memória e/ou processamento. Possui APIs que visam economia de espaço, de memória e de processamento. São utilizadas em sistemas como, por exemplo, aparelhos celulares, *palm tops, pocket pcs, smartphones* e *javacards*.

c) *Java Enterprise Edition* (JEE) - é a tecnologia Java para aplicações corporativas que podem estar na Internet ou não. Possui um grande número de APIs em que a segurança é a principal preocupação. É ideal para a construção de servidores de aplicação, integração de sistemas ou distribuição de serviços para terceiros.

3.1.6 MyEclipse

MyEclipse é uma IDE para Java e é construída sobre a plataforma Eclipse (MYECLIPSE, 2011). MyEclipse integra no ambiente de desenvolvimento soluções proprietárias e de código fonte aberto.

MyEclipse possui duas versões básicas, que são a profissional e padrão. A versão padrão possui os recursos básicos do Eclipse. A versão profissional adiciona à versão padrão ferramentas de banco de dados, projeto visual para aplicações *web*, ferramentas de persistência, ferramentas Spring, Struts e JSF (*Java Server Faces*) e vários outros recursos para o desenvolvedor. A Figura 5 apresenta a tela principal da versão profissional do MyEclipse. Essa figura é um *print screen* da ferramenta MyEclipse.

Recentemente MyEclipse está oferecendo uma versão customizada para produtos IBM. Essa versão é denominada MyEclipse Blue Edition e adiciona suporte específico para desenvolvimento com Web Sphere e Rational. Atualmente, MyEclipse Blue Edition está somente disponível para Windows (MYECLIPSE, 2011).

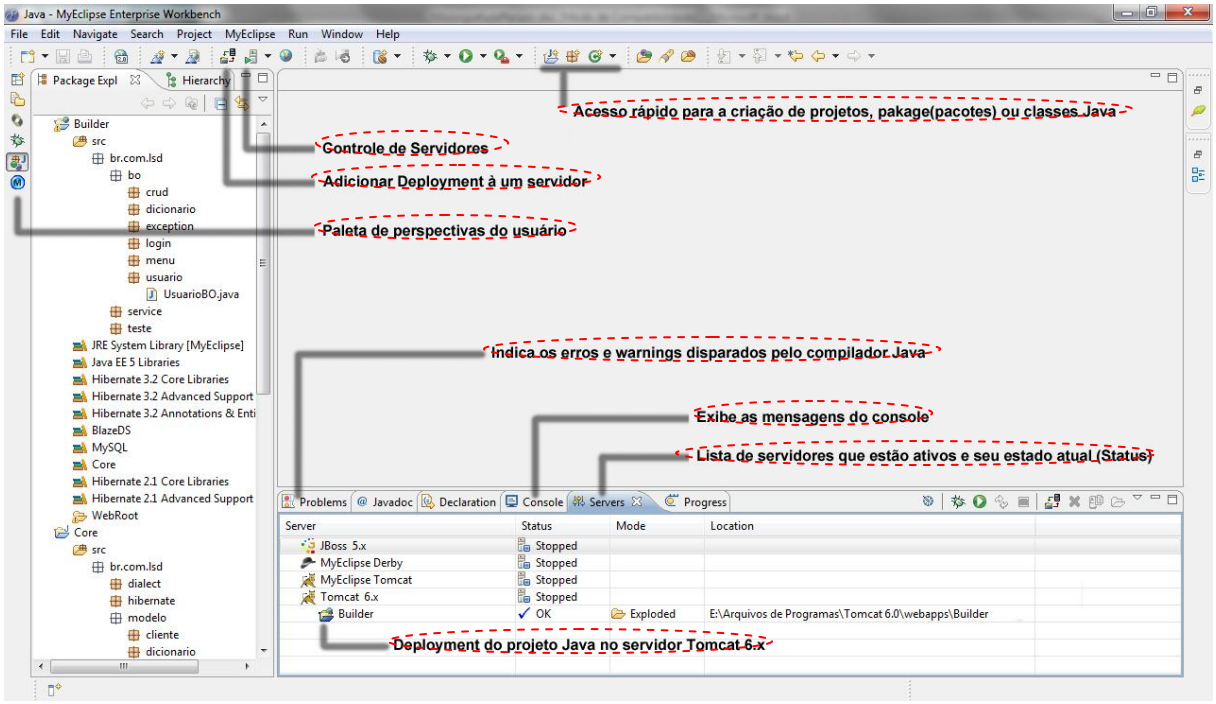

Figura 5 – Tela principal do MyEclipse

As elipses pontilhadas na Figura 5 apresentam as principais partes e/ou funcionalidades da interface do MyEclipse. Os próprios comentários nessas elipses são autoexplicativos.

É possível ter várias perspectivas em um *workspace* do MyEclipse assim como no

Flex Builder que é feito sobre a plataforma Eclipse. Os servidores podem ser iniciados em dois modos, o normal e o *debug*. No modo *debug*, quando houver *breakpoints* todos os detalhes do escopo poderão ser visualizados. E o comportamento é como o *debug* do Flex Builder. Para um projeto do tipo Java para *web* é necessário que seja criado um *deployment* que é o envio do projeto já compilado para a pasta do servidor.

3.1.7 Adobe BlazeDS

O Adobe BlazeDS é um produto *OpenSource* (Licença LGPL v3 (GNU *Lesser General Public License*)) que corresponde à tecnologia Java *server side* com suporte tanto para o *remoting* como para o *messaging* de objetos trocados entre Java e Flex/Flash por meio do protocolo AMF (ADOBE, 2011c). O BlazeDS possibilita a conexão de aplicativos Flex à infra-estrutura de *backend* de servidor Java e de dados distribuídos.

O BlazeDS permite conexão entre a camada de visualização e o *backend* (modelo e controle) e provê a transferência de informações em tempo real entre as camadas da aplicação.

3.1.8 AMF

O Action Message Format é um formato compacto binário que é usado para serializar objetivos gráficos Action*Script* (ADOBE, 2011c). Uma vez serializado um objeto gráfico codificado em AMF pode ser usado para persistir e consultar o estado público de uma aplicação por meio de sessões que permitem a dois elementos se comunicarem pela troca de dados fortemente tipados. A versão do AMF, referida como AMF 3 e coincidente com o versão 3 do Action*Script*, melhora o AMF 0 por enviar atributos dos objetos e *strings* por referencia em adição a instâncias de objetos. AMF 3 é especificada pela Adobe por meio da AMF 3 Specification (ADOBE, 2011c).

AMF melhora o desempenho da aplicação pela grande compressão que ele faz dos dados sendo transmitidos. Ele analisa sintaticamente dados binários por meio do acesso à sua estrutura (*parsing*) em objetos na memória por isso é mais eficiente que o *parsing* de dados do XML.

3.1.9 MySQL

O MySQL é um servidor e gerenciador de banco de dados relacional, que utiliza a linguagem SQL (MYSQL, 2011). Ele é de licença dupla (*software* livre e paga), projetado inicialmente para trabalhar com aplicações de pequeno e médio portes, mas atualmente atende também aplicações de grande porte.

A seguir, algumas das principais características incorporadas na versão 5 do MySQL (MILANI, 2007):

a) Visões – são consultas pré-programadas ao banco de dados que permitem unir duas ou mais tabelas e retornar uma única tabela como resultado. Além disso, podem ser utilizadas para filtrar informações, exibindo somente os dados específicos de uma determinada categoria de uma ou mais colunas da tabela. Com o uso de visões, operações frequentes com uniões de tabelas podem ser centralizadas. É possível também utilizá-las para controle de acesso, permitindo que determinados usuários acessem dados de uma visão, mas não as tabelas utilizadas por esta, restringindo acesso a informações.

b) Cursores - cursores possibilitam a navegação em conjuntos de resultados. De forma simples, é possível navegar pelos registros de uma tabela a partir de laços de repetição, permitindo realizar operações necessárias e transações à parte para cada linha da tabela.

c) *Information Schema* – são tabelas responsáveis apenas pela organização dos recursos do banco de dados, conhecidos como dicionário de dados, ou metadados. Desta forma, é possível realizar consultas sobre a estrutura do banco de dados por meio dessas tabelas.

d) Transações distribuídas XA – transações distribuídas XA, que são uma espécie de extensão da ACID (Atomicidade, Consistência, Isolamento, Durabilidade), fornecem a possibilidade de gerenciamento dessas transações com a união de múltiplos bancos de dados (transações distribuídas) para a execução de uma mesma transação.

e) Integridade referencial - os relacionamentos entre tabelas distintas são gerenciados pelo banco de dados nos momentos de inclusão, alteração e exclusão.

f) Clusterização - a clusterização é baseada na integração e sincronismo de dois ou mais servidores para dividirem a demanda de execuções entre si. Além da sincronização de um *cluster*, é possível especificar um balanceador de cargas.

3.1.10 MySQL Workbench

MySQL Workbench (WORKBENCH, 2011) é uma ferramenta gráfica para modelagem de dados. A ferramenta possibilita trabalhar diretamente com objetos *schema*, além de fazer a separação do modelo lógico do catálogo de banco de dados.

Toda a criação dos relacionamentos entre as tabelas pode ser baseada em chaves estrangeiras. Outro recurso que a ferramenta possibilita é realizar a engenharia reversa de esquemas do banco de dados, bem como gerar todos os scripts em SQL.

Com essa ferramenta, a modelagem do banco de dados pode assumir níveis conceituais, lógicos e físicos. MySQL Workbench apresenta uma arquitetura extensível, sendo possível visualizar a representação de tabelas, funções, entre outros.

A Figura 6 (um *print screen* do MySQL Workbench) apresenta os três serviços disponíveis pelo MySQL Workbench, circulado em pontilhado.

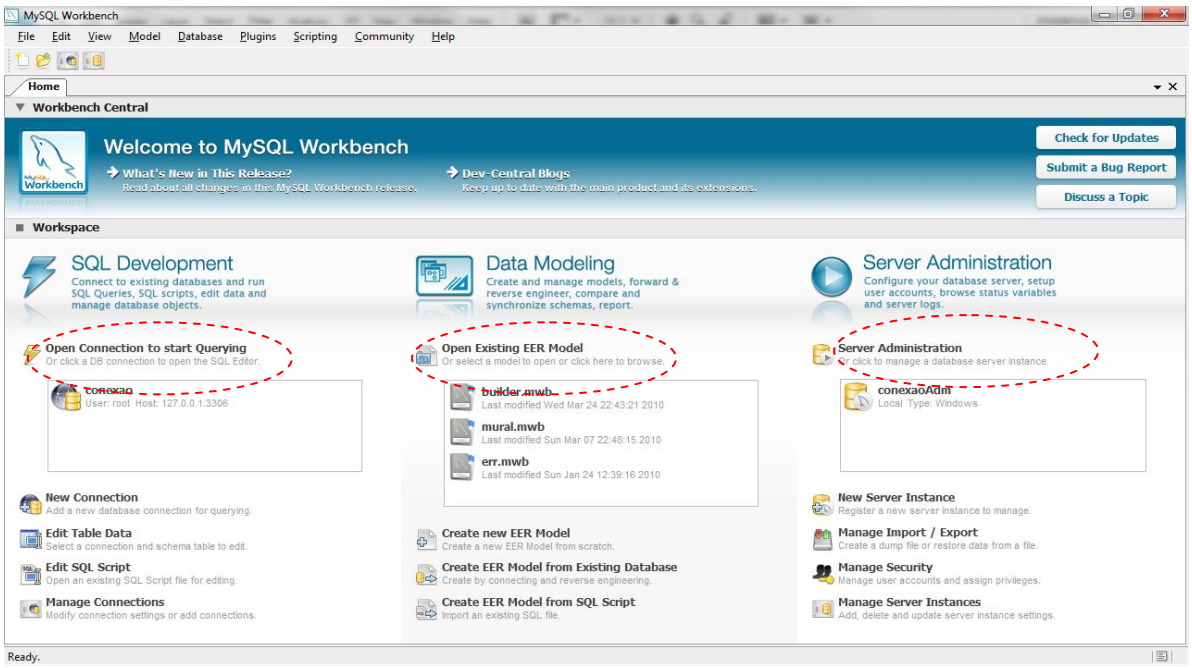

Figura 6 – Tela dos serviços do MySQL Workbench

As partes circuladas na Figura 4 indicam:

a) *SQL Development* é uma área em que se pode conectar a bases de dados registradas no servidor possibilitando a execução de *query*, *scripts* SQL e editar dados destas bases. O editor SQL avisa o usuário caso haja erros de sintaxe sem a necessidade de executar o comando.

b) *Data Modeling* possibilita ao usuário criar e gerir modelos de dados, executar engenharia reversa utilizando uma base de dados registrada para gerar um novo diagrama.

Gera os scripts completos ou apenas os que o usuário escolher para executar no *SQL Development*. Criar um modelo de dados é simples bastando arrastar e soltar as tabelas, ao clicar duas vezes em uma tabela, as propriedades desta aparecem na parte inferior para adicionar colunas, indexes e demais componentes de uma tabela.

c) *Server Administration* contém as configurações gerais do servidor, manipulação de usuários, controle de portas e conexões gráficos do sistema e do servidor, *backups* e restaurações de bases de dados.

3.1.11 MySQL Admin

MySQL Admin como uma abreviatura de MySQL Administrator é um programa para executar operações administrativas, como configuração, monitoramento e início e parada de um servidor MySQL, gerenciamento de usuários e conexões, executar *backups* e diversas outras tarefas administrativas (MYSQL ADMINTRATOR, 2011). A maioria das tarefas pode ser executada utilizando um cliente de linha de comando.

O MySQL Administrator apresenta as seguintes vantagens: a sua interface gráfica permite fazer toda a composição e gerenciamento do banco, proporciona uma melhor visão das configurações que são essenciais para a performance, confiabilidade e segurança de seus servidores MySQL, e exibe indicadores de desempenho graficamente, tornando mais fácil para definir e ajustar as configurações de servidor.

3.1.12 Hibernate

O Hibernate (HIBERNATE, 2011) é um *framework* para o mapeamento objeto-relacional escrito na linguagem Java, mas também é disponibilizado em .Net como o nome NHibernate. Esse programa facilita o mapeamento dos atributos entre uma base de dados relacional e o modelo de objetos de uma aplicação, mediante o uso de arquivos XML e anotações.

O Hibernate é um *software* livre de código aberto distribuído com a licença LGPL. O objetivo do Hibernate é diminuir a complexidade dos programas Java, baseado no modelo orientado a objeto, que precisam trabalhar com um banco de dados do modelo relacional (presente na maioria dos sistemas gerenciadores de bancos de dados). Em especial, no

desenvolvimento de consultas e atualizações dos dados.

O Hibernate transforma os dados tabulares de um banco de dados em um grafo de objetos definido pelo desenvolvedor. Sua principal característica é a transformação das classes Java para tabelas de dados (e dos tipos de dados Java para os da *Structured Query Language* (SQL)). Ele gera as chamadas SQL, mantendo o programa portável para qualquer banco de dados padrão SQL.

Apesar de existirem APIs no Hibernate para operações de controle transacional, ele delegará essas funções para a infra-estrutura na qual foi instalado. Assim, o gerenciamento de transações e a tecnologia de acesso à base de dados são de responsabilidade de outros componentes do programa.

Em aplicações construídas para serem executadas em servidores, o gerenciamento das transações é realizado de acordo com o padrão JTA (*Java Transaction API*). Nas aplicações *desktop*, o programa delega o tratamento transacional ao *driver* JDBC (*Java Database Connectivity*). Hibernate pode ser utilizado em aplicações Java *desktop* ou em aplicações JEE, utilizando *servlet* ou sessões EJB (*Enterprise Java Beans*).

3.1.13 Apache Tomcat

O Apache Tomcat (TOMCAT, 2011) é um contêiner *servlet*, um servidor de aplicações Java utilizado para interpretar aplicações escritas em Java para *web*. Ele não implementa um contêiner EJB, mas abrange parte da especificação JEE com tecnologias como servlet e JSP e tecnologias de apoio relacionadas, como segurança, JNDI (*Java Naming and Directory Interface*) Resources (API para acesso a diretórios) e JDBC DataSources.

Tomcat pode atuar também como servidor *web*, ou pode funcionar integrado a um servidor *web* dedicado como o Apache ou o IIS (*Internet Information Services*). Como servidor *web*, ele provê um servidor *web* HTTP puramente em Java. O servidor inclui ferramentas para configuração e gerenciamento, o que também pode ser feito editando-se manualmente arquivos de configuração formatados em XML.

3.2 MÉTODO

O método utilizado para a realização do trabalho está baseado nas fases de análise,

projeto, codificação e testes do modelo sequencial linear exposto em Pressman (2005). Essas fases foram utilizadas como base, mas sofreram inclusões e alterações para adaptar-se aos objetivos deste projeto. Ressalta-se que as atividades realizadas não foram estritamente sequenciais. .As principais fases ou etapas definidas para o ciclo de vida do sistema são:

a) Planejamento – planejamento do projeto, definindo as principais etapas e atividades. Esse planejamento foi feito com a orientadora deste trabalho visando desenvolver o sistema e a monografia em paralelo. Reuniões periódicas com a orientadora permitiam avaliar e ajustar as atividades planejadas e estabelecer prazo para o término de cada atividade que seria realizada.

As reuniões iniciais tiveram como objetivo definir os requisitos do sistema e determinar como seria a forma de interação do usuário. Também foi avaliada a forma de disponibilização do sistema, uma vez que, atualmente, o mesmo não pode ser hospedado no mesmo servidor em que está o portal da área de informática. Verificou-se, assim, que seria disponbilizado em outro servidor com acesso por meio de um *link* vinculado ao portal dessa área.

b) Requisitos – a definição dos requisitos do software foi realizada com a ajuda da orientadora. Em reuniões foram identificados os requisitos inicias para o sistema e posteriormente os seus refinamentos. O professor Omero Francisco Bertol também participou da definição dos requisitos.

Como forma de expor a ideia geral do sistema um diagrama com a visão geral do mesmo foi definido. Esse diagrama representado por um conjunto de conceitos conectados entre si permitiu visualizar o sistema como um todo, sem preocupação em definir classes e/ou informações armazenadas.

À medida que o sistema era implementado, novas reuniões foram realizadas para verificar se os requisitos planejados estavam sendo atendidos e para esclarecer dúvidas. Para melhor entender a forma de interação do usuário com o sistema, as telas foram prototipadas utilizando a ferramenta Balsamiq Mockups. E as tabelas do banco de dados foram representadas por um diagrama de entidades e relacionamentos. Após a elaboração desse diagrama verificou-se que uma das tabelas definidas não seria mais necessária. Um campo a mais em uma outra tabela seria suficiente para controlar se uma determinada sigla estava sendo sugerida ou inserida no banco de dados.

c) Análise – a análise consistiu na definição dos requisitos funcionais e representá-los sob a forma de casos de uso. Esses requisitos foram complementados com aspectos de qualidade, definindo os requisitos não funcionais. Esses requisitos definiram o problema, ou

seja, o que seria implementado pelo sistema.

d) Projeto – tendo como base os requisitos definidos na fase de análise foram definidos os modelos para solucionar o problema, ou seja, definir como o sistema atenderia aos requisitos (funcionalidades e requisitos não funcionais) definidos na análise. Essa solução foi modelada basicamente por meio de diagramas de classes, entidade-relacionamento da base de dados, protótipo de telas. Para a modelagem dos dados foi utilizada a ferramenta MySQL Workbench.

e) Implementação – na implementação foi realizada a codificação do sistema e realização de testes pelo programador, o autor deste trabalho. A codificação baseou-se na criação da interface do sistema com componentes Flex, a implementação das regras de negócio (camada da aplicação) com a linguagem Java, utilizando BlazeDS e o protocolo AMF para a transformação (serialização) dos objetos Flex para objetos Java e vice-versa (desserialização). O MySQL foi utilizado como banco de dados e o Hibernate para o mapeamento entre os objetos Java e as tabelas e campos que representam uma estrutura relacional do banco de dados.

f) Testes – os testes foram informais e realizados pelo próprio programador, o autor deste trabalho.

4 RESULTADOS E DISCUSSÃO

Este capítulo apresenta o sistema que foi desenvolvido como resultado deste trabalho. Inicialmente é apresentada a descrição do mesmo. Em seguida é apresentada a sua modelagem Por fim, o sistema é apresentado, por meio de telas e explicação do seu funcionamento e a implementação é exemplificada por meio de partes da codificação.

4.1 APRESENTAÇÃO DO SISTEMA

O sistema para composição de listas de siglas desenvolvido como resultado da realização deste trabalho de conclusão de curso visa oferecer uma forma prática para professores e alunos da Universidade Tecnológica Federal do Paraná campus Pato Branco e de outras pessoas de consultar o significado de siglas, abreviaturas e acrônimos e compor listas com a abreviatura e com o respectivo significado.

Uma sigla pode ter significados distintos. Assim, quando do cadastramento da sigla é necessário informar uma área de conhecimento a que essa sigla pertence. E, também, possível cadastrar significados distintos dentro de uma mesma área de conhecimento para uma mesma sigla. Recursos de aplicações Internet ricas foram utilizados para facilitar a escolha das siglas que compõem uma lista. Dentre esses recursos destaca-se o de arrastar-e-soltar, à semelhança de um carrinho de compras, e a listagem de siglas por ordem alfabéticas em abas distintas.

O sistema será disponibilizado por meio do portal acadêmico da área de Informática do campus Pato Branco. Além de consultar, os usuários podem sugerir siglas e seus significados. As siglas sugeridas devem ser validadas por um usuário professor ou administrador para que possam ser efetivamente incluídas no banco de dados.

O sistema possui três níveis de acesso: administrador, professor e padrão. O administrador possui acesso a todas as funcionalidades do sistema. O professor inclui, exclui, altera, consulta e valida siglas sugeridas. O usuário que desejar permissões de professor faz um pré-cadastramento no sistema informado *login* e senha. Esse cadastro é validado pelo usuário administrador. O usuário padrão não precisa estar cadastrado. E pode consultar o significado de siglas, sugerir siglas e respectivos significados e compor listas. A denominação “padrão” é apenas para indicar um nome específico para um usuário sem a necessidade do cadastro acessar determinadas funcionalidades do sistema.

4.2 MODELAGEM DO SISTEMA

Inicialmente foi definido um diagrama com a visão geral do sistema, representada na Figura 7. Esse diagrama foi definido com o objetivo de que o sistema pudesse ser visualizado como um conjunto de conceitos relacionados.

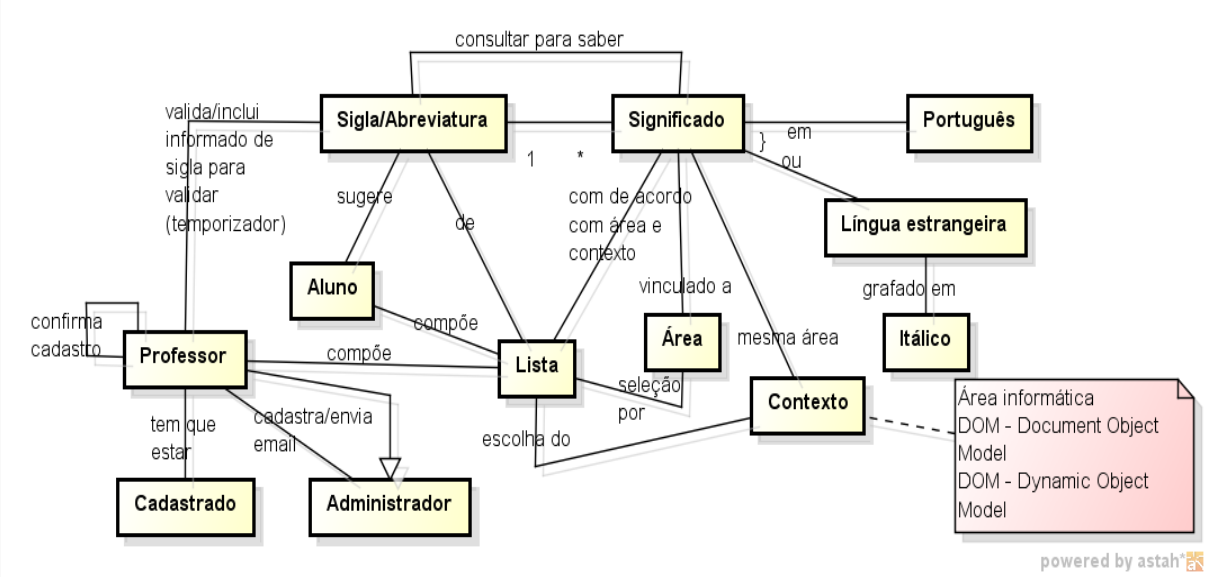

Figura 7 – Visão geral do sistema

A nota constante na Figura 1 com uma observação para o significado da sigla “DOM”, enfatiza que uma mesma sigla pode ter significados distintos em uma mesma área. A partir da visão geral do sistema representada na Figura 1, foram definidos os requisitos para o sistema,

Requisitos funcionais:

a) Cadastrar siglas e seu significado, indicando em língua nacional (português) ou estrangeira. As siglas podem ser pré-cadastradas (sugeridas) por usuários com permissões de usuário padrão e somente são inseridas no banco de dados após serem validadas por um usuário professor.

b) Cadastrar áreas, exemplo: Informática, Administração ou Física.

c) Compor uma lista no formato:

Sigla <marca de tabulação> significado

“marca de tabulação” indica alguma forma de manter a coluna à direita alinhada ou já colocadas em uma tabela sem as linhas visíveis.

d) Atribuição de permissão de inclusão e exclusão no sistema para usuários do tipo professor.

e) Permitir a consulta do significado de uma determinada sigla. Nesse caso trazer

todos os significados da respectiva sigla, indicando a área a que pertencem.

Requisitos não funcionais:

a) Uma sigla pode ter significados distintos em uma mesma área ou para áreas distintas. Permitir a indicação da área no cadastro da sigla e a inserção de significados diferentes para uma sigla da mesma área.

b) Definir uma maneira que facilite ao usuário escolher as siglas que irão compor a lista. Poderia ser semelhante a um carrinho de compras em que os itens (siglas) são escolhidos e posteriormente comporão a lista (carrinho de compras).

c) Na lista, o significado das siglas em inglês (ou outra língua que não o português) deve estar em itálico.

d) O sistema deverá possuir usuários distintos. Um usuário de consulta (denominado "padrão" como forma de diferenciação) que pode compor listas e sugerir siglas e seus respectivos significados. Um usuário professor, por exemplo, que pode incluir siglas e significados e validar (aceitar inclusão no banco) de termos sugeridos por usuários de consulta. Um usuário administrador que concede permissões de inclusão e validação de termos a usuários professor. O administrador faz um pré-cadastro do professor. Envia mensagem por e-mail que para este confirmar e completar o seu cadastro.

e) Definir a forma mais adequada de composição da lista, opções:

e.1) O usuário inicialmente seleciona a área. São mostrados somente os termos da respectiva área.

e.2) O usuário opta por visualizar as siglas de todas as áreas para escolher.

f) Uma sigla pode ser indicada por um usuário, informando a sigla, o significado, *email* (opcional). Não é necessário validar o *email* indicado. Contudo, informar que é preciso indicar um *email* para retorno de que a sigla foi cadastrada, mas que este não é obrigatório.

g) Usuários professor validam siglas informadas por usuários padrão.

h) Definir uma mensagem padrão informando que a sigla foi cadastrada para o *email* do usuário que sugeriu. Não é necessário verificar se o *email* efetivamente foi enviado.

i) Armazenar sigla sem pontuação e validar se a mesma existe no banco por aproximação. Técnica de aproximação por letras iguais, removendo caracteres especiais.

j) Validar na digitação da consulta por siglas e trazer por aproximação a partir de letras digitadas.

k) Opção de ordenar ou não a lista. A ordenação será feita pela coluna do *grid* gerado.

l) Deverá haver um temporizador para informar o usuário logado como professor que há siglas para validar.

g) Controlar os estados de uma sigla.

Os requisitos funcionais do sistema foram representados sob a forma de um diagrama de casos de uso. Esse diagrama está apresentado na Figura 8.

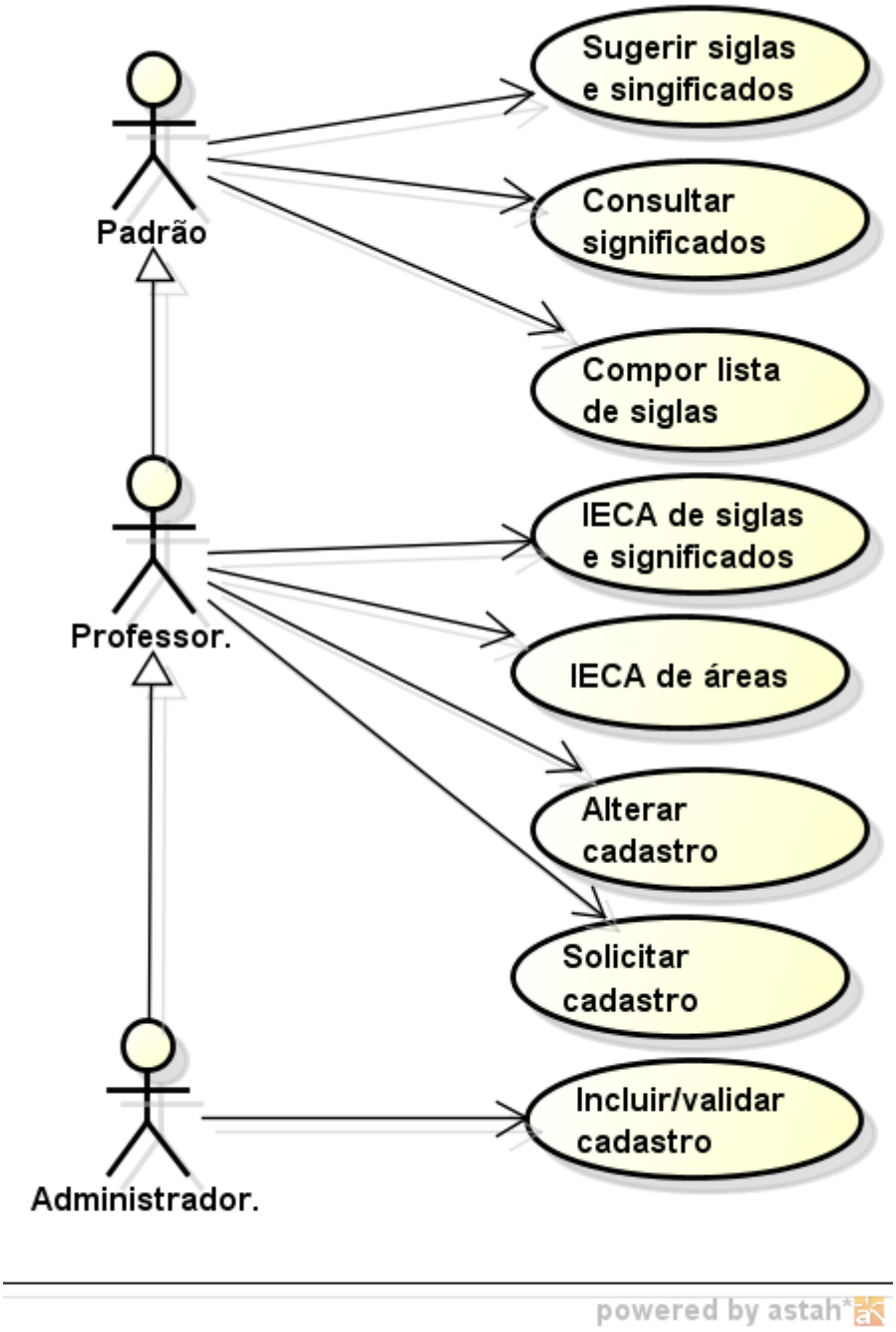

Figura 8 - Diagrama de casos de uso do sistema

A Figura 9 apresenta um diagrama de estados para indicar os possíveis estados que uma sigla pode assumir durante o seu ciclo de vida. Esse diagrama completo se aplica a uma sigla sugerida para inclusão no banco de dados. Uma sigla incluída por um usuário professor ou administrador está no estado “incluída no banco” e desse estado pode ser excluída.

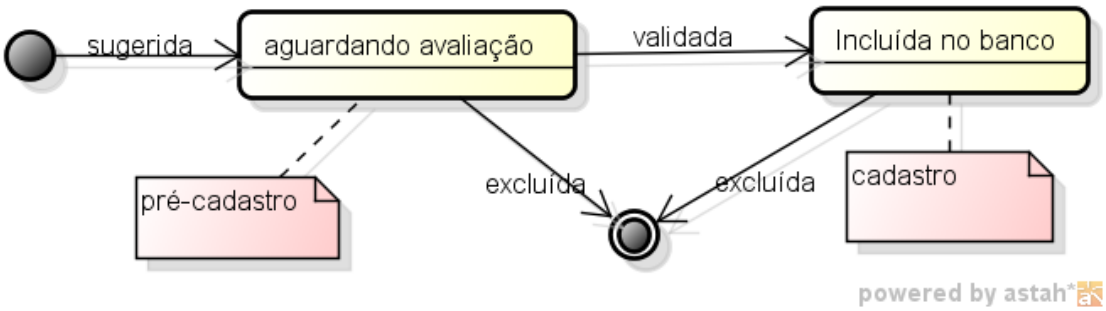

Figura 9 – Diagrama de estados da sigla

Para melhor entender a forma de interação do usuário com o sistema, os protótipos das telas foram definidos utilizando a ferramenta Balsamiq Mockup. As Figuras 10 a 14 apresentam os protótipos elaborados para as telas. Por meio dessa ferramenta é possível definir a forma de interação do usuário com o sistema, a disposição dos elementos da interface e mesmo verificar e validar os requisitos e auxiliar a definir os campos para banco de dados e os atributos de classes. Vários protótipos foram gerados até obter as telas com o leiaute e os recursos considerados adequados às funcionalidades e objetivo do sistema. Devido aos componentes existentes na ferramenta Balsamiq Mockup, os protótipos gerados ficaram muito semelhantes às telas elaboradas utilizando Flex.

A Figura 10 apresenta protótipo da tela para compor listas.

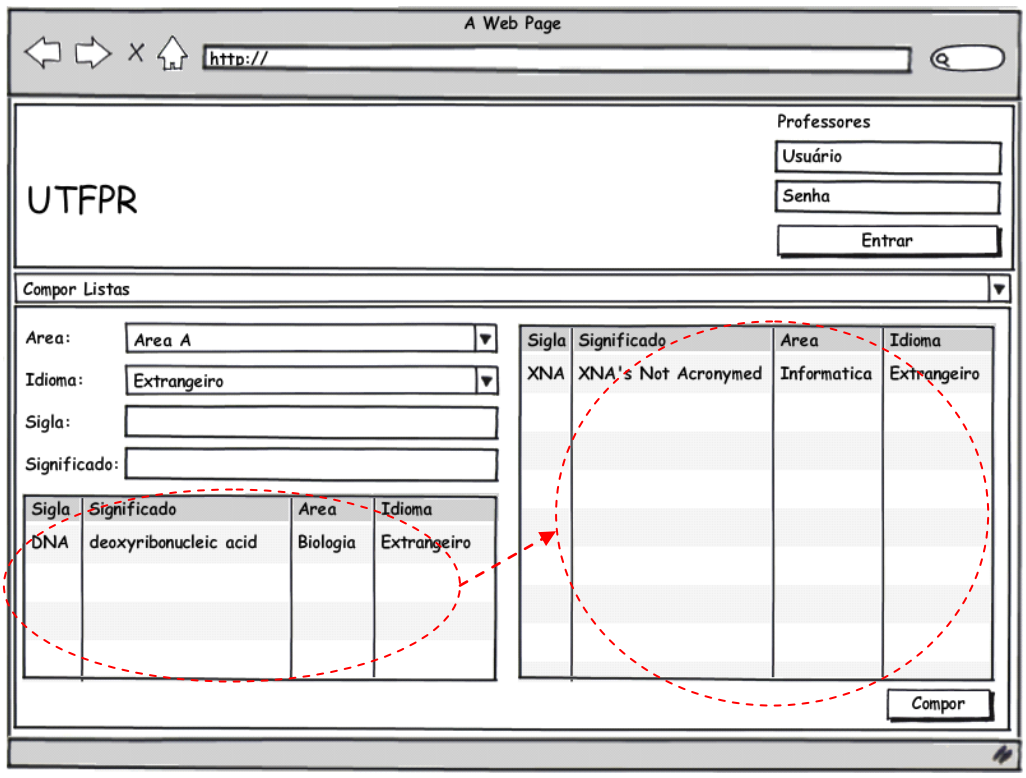

Figura 10 – Protótipo da tela para composição de listas de siglas

As áreas circuladas na parte inferior da Figura 10 ressaltam a listagem das siglas obtidas a partir dos filtros aplicados e a lista sendo composta. A lista é composta por meio do recurso arrastar-e-soltar. Na Figura 11 está o protótipo da tela para cadastrar siglas.

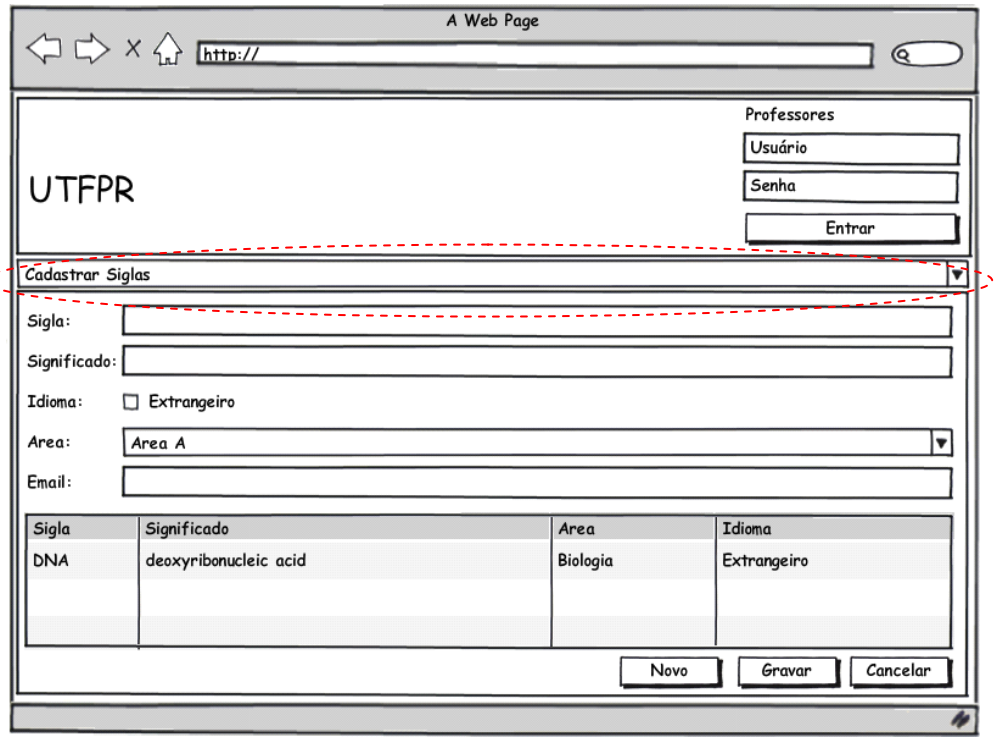

Figura 11 – Protótipo da tela para cadastro de siglas

Na Figura 12 está o protótipo da tela para a aprovação de siglas.

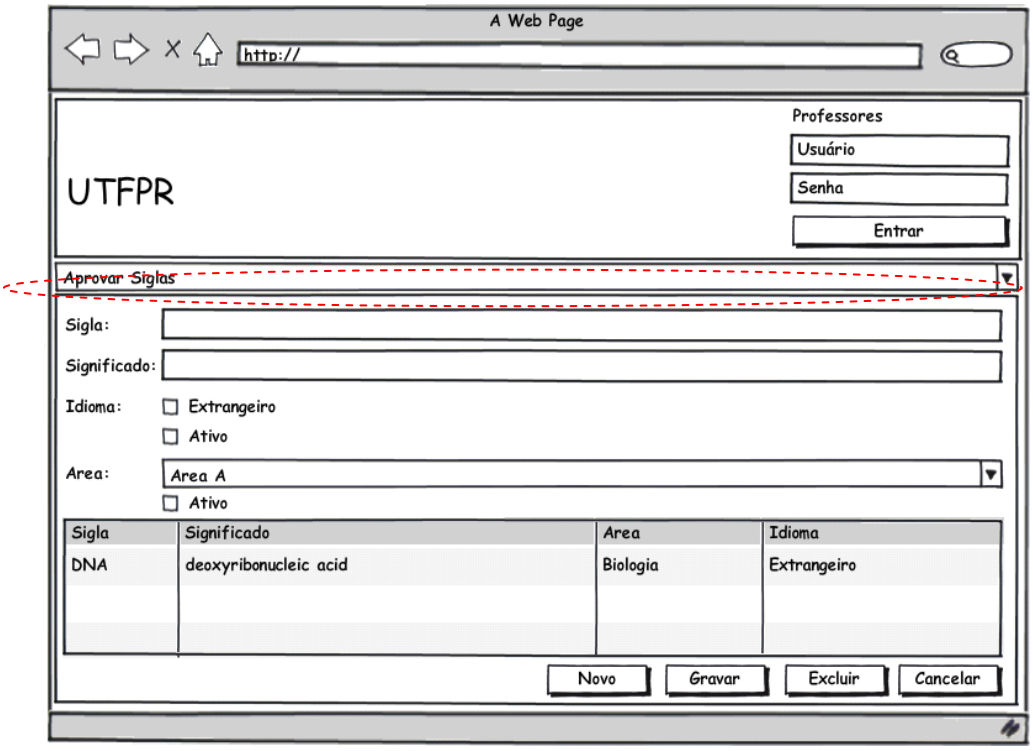

Figura 12 – Protótipo da tela para aprovação de siglas

A Figura 13 apresenta o protótipo da tela para cadastro de professor.

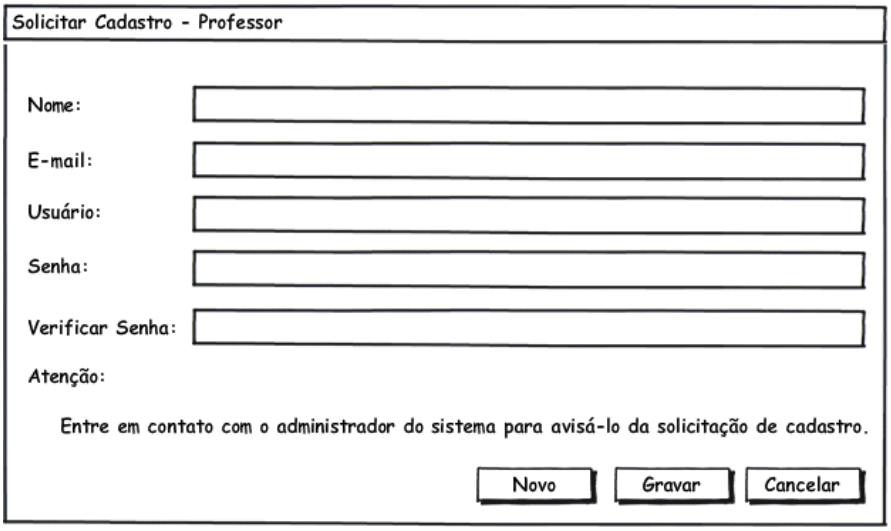

Figura 13 – Protótipo da tela para cadastro de usuário professor

Na Figura 14 está o protótipo gerado da tela para aprovação de professor.

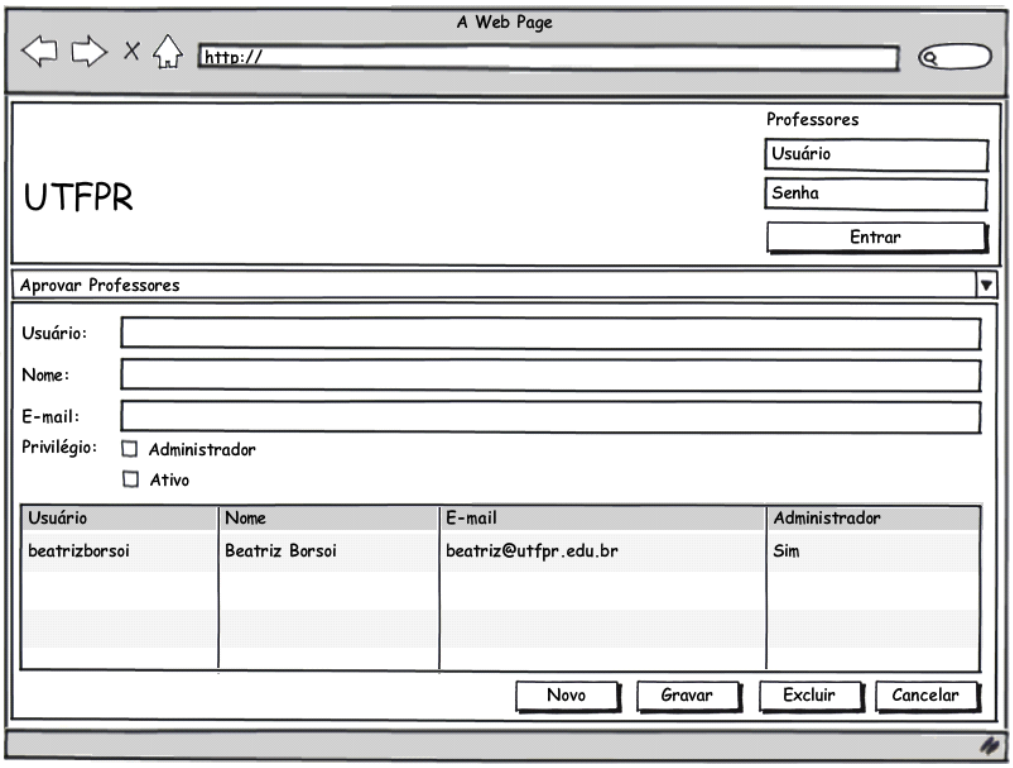

Figura 14 – Protótipo da tela para aprovação de usuário professor

O banco de dados é composto das tabelas “área”, “sigla”, “professor“ e “significado”. A Figura 15 contém o diagrama de entidades e relacionamentos definido para o banco de dados.

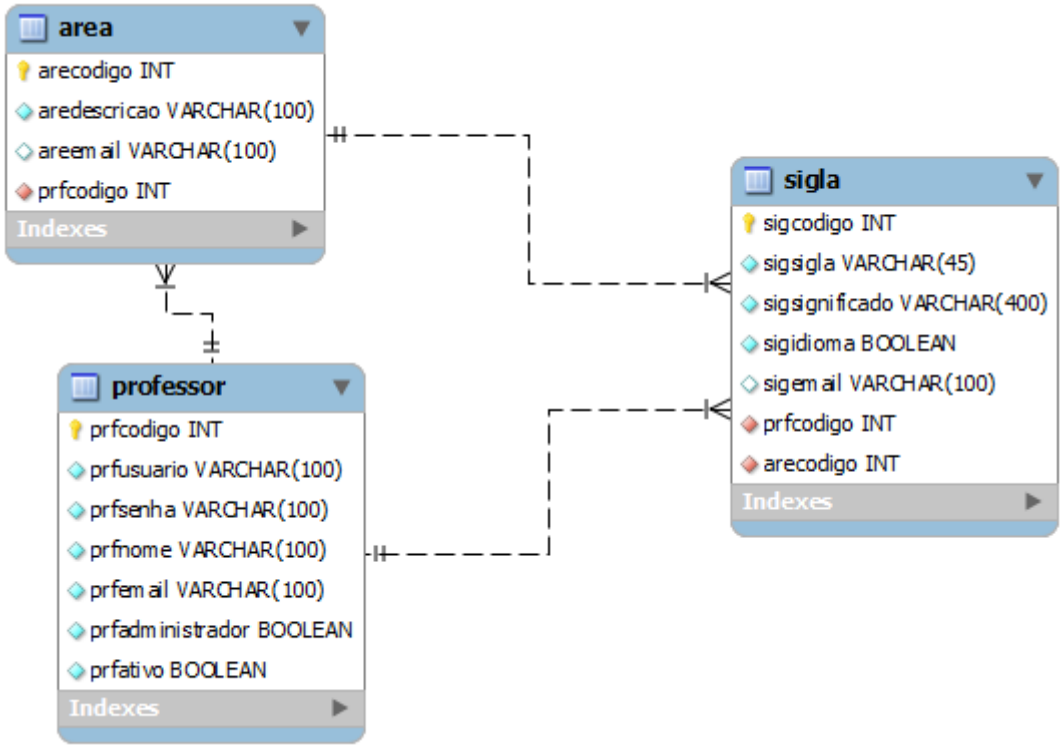

Figura 15 – Diagrama de entidades e relacionamentos do banco de dados

Uma sigla pode ser sugerida por um usuário (que não é professor). Essa sigla é armazenada no banco de dados. Se ela esse sigla não foi sugerida por professor o campo "prcodigo" não estará preenchido e isso significa que a mesma ainda não foi validada. As tabelas "sigla" e "area" possuem um campo "email" para identificar quem sugeriu a sigla e para o mesmo ser avisado quando a mesma é efetivamente inserida no banco de dados. No momento da validação a sigla pode não ser inserida se a sugestão não for pertinente ao escopo e propósito do sistema.

4.3 DESCRIÇÃO DO SISTEMA

A seguir estão as principais telas do sistema visando exemplificar a forma de interação com o sistema e as principiais funcionalidades implementadas. A Figura 16 apresenta a tela para compor listas. Essas listas são uma composição de siglas e seus respectivos significados. É um relatório do sistema, com os itens escolhidos pelo usuário.

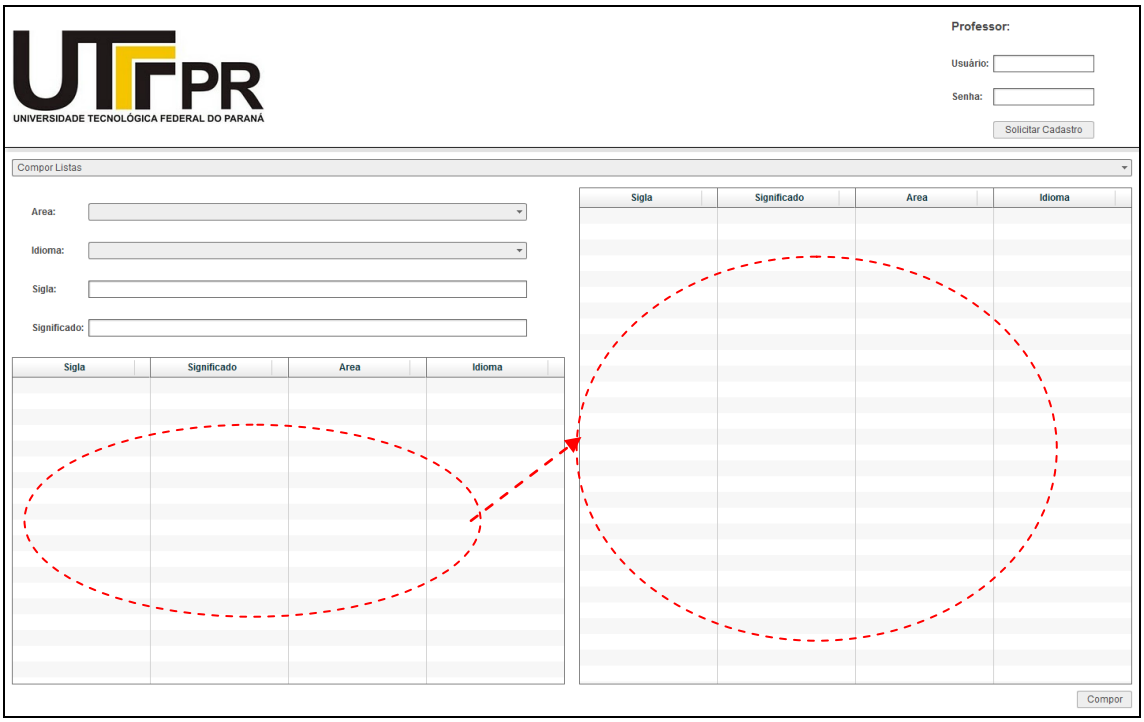

Figura 16 – Tela para composição de listas de siglas

A tela de composição de listas (Figura 16) é sempre a primeira a ser exibida ao usuário quando este acessar o sistema. Para poder encontrar a sigla desejada, os filtros de Área, Idioma, Sigla e Significado são aplicados para apresentar itens que compõem o *grid* na parte inferior esquerda da tela. Ao encontrar a sigla desejada, basta que o usuário clique e arraste-a para o *grid* da direita. Essa sigla fará parte da lista gerada.

Depois de adicionar todos os itens desejados, caso o usuário acione o botão compor, será exibida uma tela com os itens selecionados no formato de uma tabela em HTML, podendo então ser copiado para um documento qualquer. Esses itens atendem ao padrão definido como requisito não funcional do sistema.

A Figura 17 apresenta a tela para cadastrar siglas.

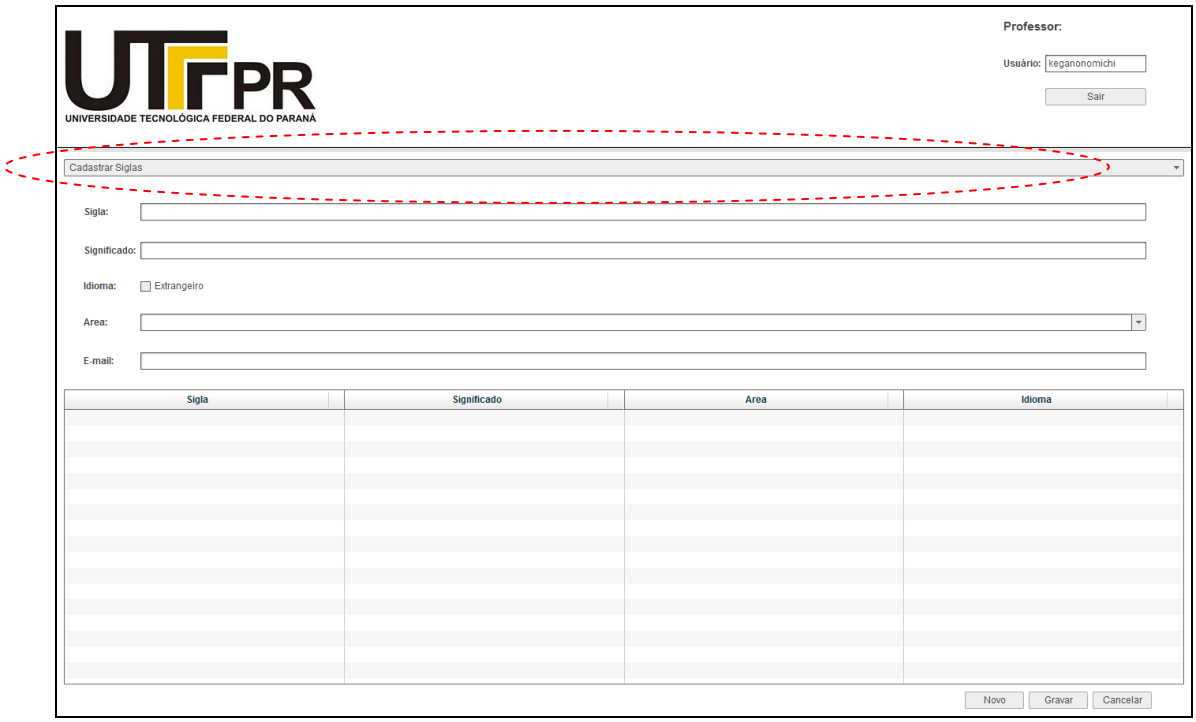

Figura 17 – Tela para cadastro de siglas

A tela de cadastro de siglas é apresentada ao usuário que selecionar “Cadastrar Siglas” ou “Sugerir Siglas” na caixa de combinação (*combobox*) que está próxima ao cabeçalho do sistema (circulada na Figura 17). Essa tela é utilizada para que o usuário possa cadastrar ou sugerir siglas, de acordo com suas permissões de uso do sistema. Caso o usuário tenha permissão de professor ou administrador, as siglas cadastradas já estarão ativas para utilização por outros usuários. Caso contrário, a sigla sugerida deverá ser verificada por algum outro usuário que tenha permissão de professor ou administrador.

Na Figura 18 está a tela para a aprovação de siglas.

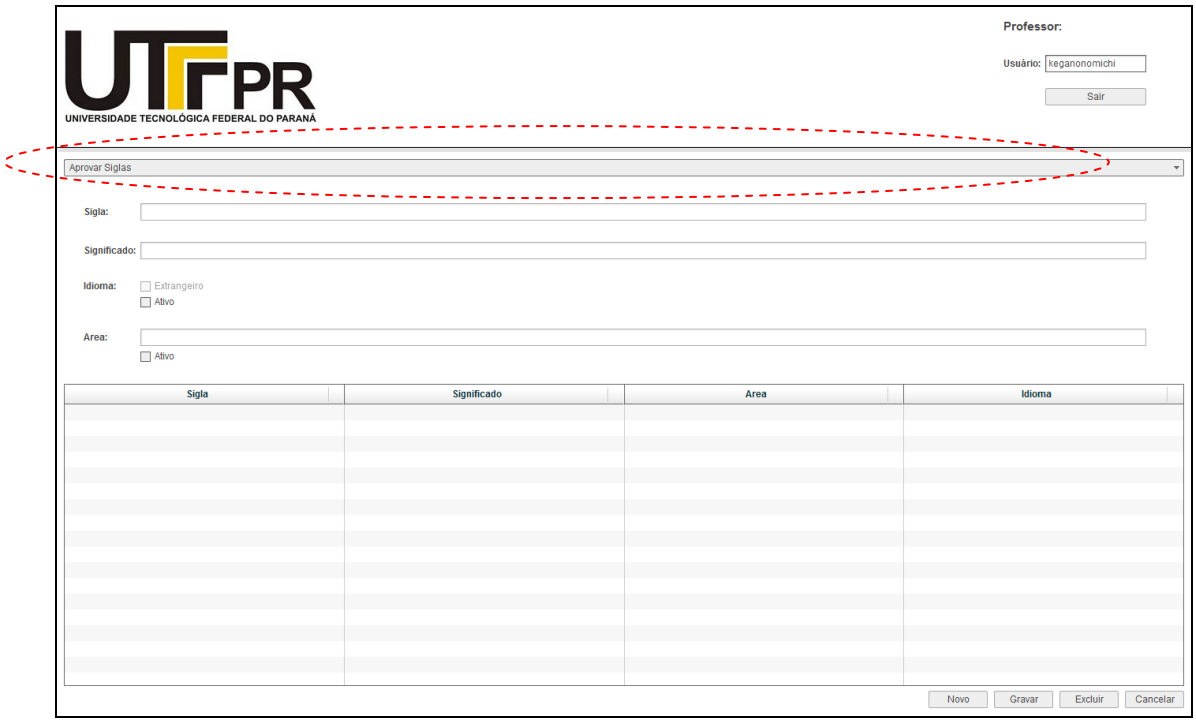

Figura 18 – Tela para aprovação de siglas

A tela do sistema para aprovar siglas é apresentada ao usuário que selecionar a opção “Aprovar Siglas” na caixa de combinação situada próximo ao cabeçalho do sistema e possuir permissão de um professor ou administrador. Essa tela é utilizada para aprovar as siglas e as áreas sugeridas por usuários sem permissões. O usuário pode aprovar apenas a sigla, apenas a área ou ambos. Caso a área seja uma ainda não cadastrada no sistema, esta também deverá ser aprovada para que a sigla seja aprovada. Caso não seja marcada nenhuma das caixas de verificação (*checkbox*) de “Ativo”, o registro permanecerá inativo e não será apresentado na tela de compor listas. Caso não haja a intenção de manter uma sugestão, o usuário poderá excluí-la por meio do botão homônimo na parte inferior da tela.

A tela da Figura 19 apresenta o cadastro de professor.

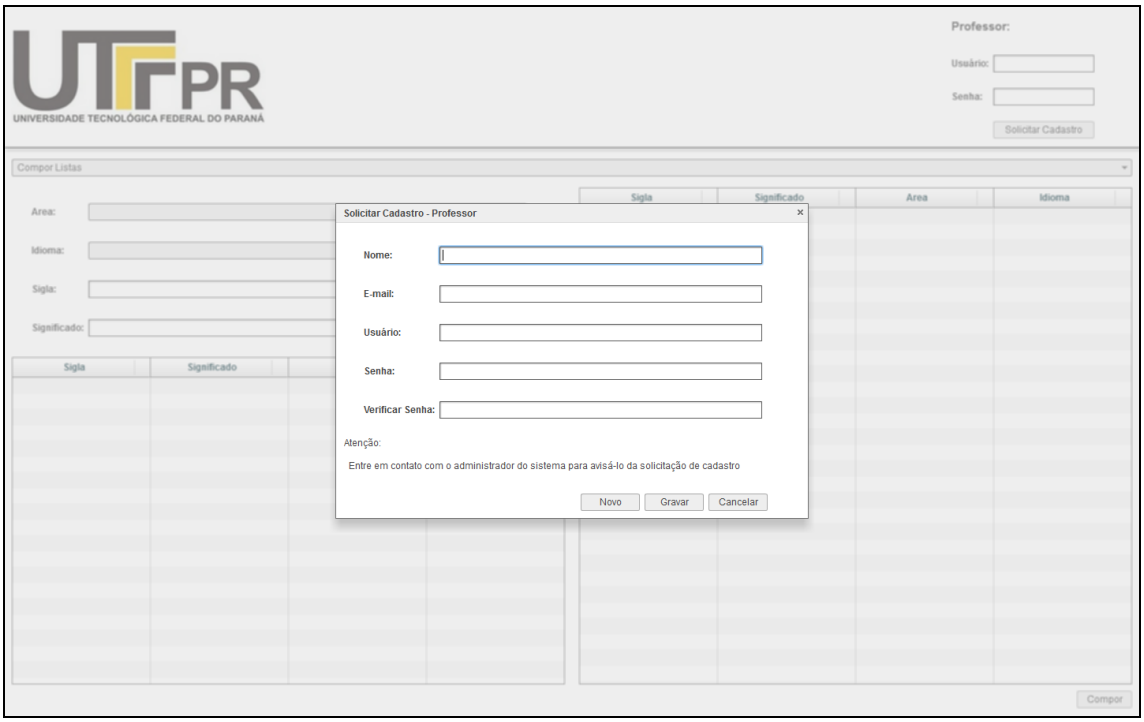

Figura 19 – Tela para cadastro de usuário professor

A tela de cadastro do professor é apresentada de forma modal ao usuário que acionar o botão “Solicitar Cadastro” na tela principal do aplicativo, ou ao usuário cadastrado no sistema e logado que acionar a opção “Alterar Cadastro” na mesma tela.

Quando o sistema é acessado por um usuário sem estar logado, ou seja, não é do tipo professor ou administrador, essa tela é utilizada para efetuar a solicitação de cadastro para professor. A solicitação terá que ser validada por um usuário Administrador para que seja ativada. Quando acessado por um usuário logado no sistema como professor ou administrador, a tela serve para alterar informações do cadastro, como o e-mail, o usuário (*login*) e a senha. O usuário deverá ser único no sistema.

A Figura 20 apresenta a tela para aprovação de usuário professor.

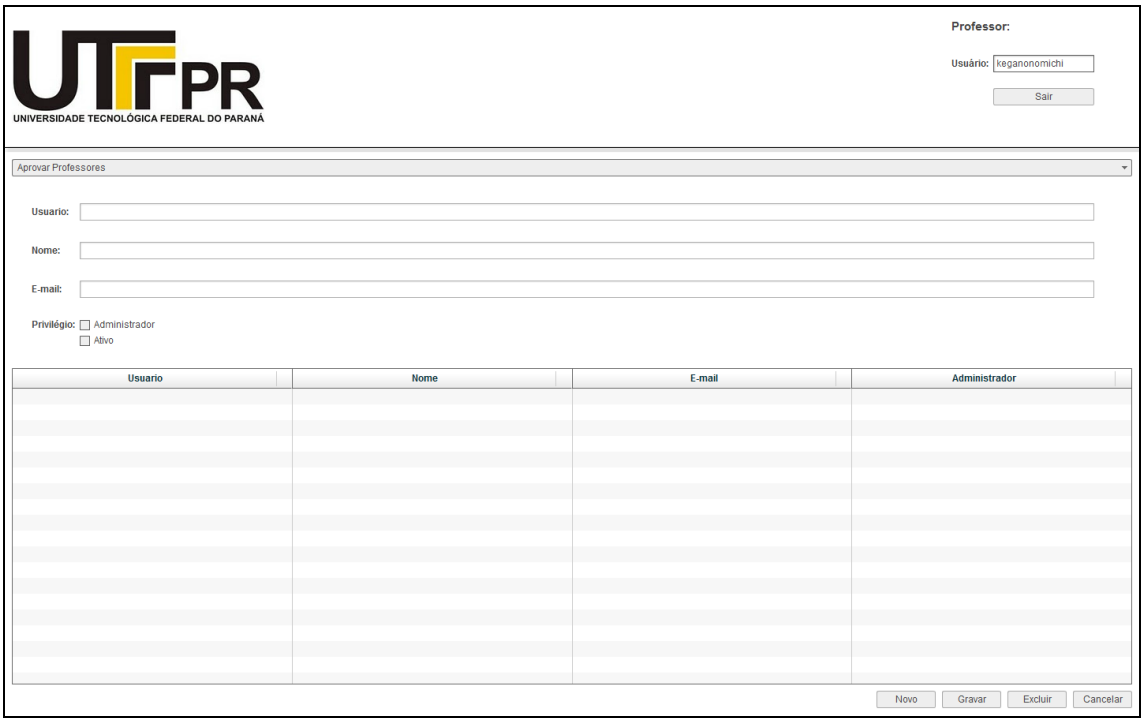

Figura 20 – Tela para aprovação de usuário professor

A tela de aprovar professores é apresentada ao usuário que selecionar "Aprovar Professores" na caixa de combinação que está situada próxima ao cabeçalho do sistema e possuir permissão de administrador do sistema. Nesta tela é possível alterar a situação ou excluir o cadastro de um professor. Não é possível cadastrar um professor novo por esta tela. O pré-cadastro é feito pelo próprio usuário que desejar acesso como professor.

4.4 IMPLEMENTAÇÃO DO SISTEMA

Nesta seção é apresentada a implementação do sistema Siglas, mostrando e explorando os principais trechos de código utilizados no desenvolvimento do aplicativo. O sistema foi construído utilizando as tecnologias Flex e Java e o servidor de aplicativos Apache Tomcat, sendo, portanto um aplicativo RIA. A parte do aplicativo desenvolvido em Flex é responsável, principalmente, pela parte visual do sistema, mas também realiza algumas consistências antes de enviar os dados para o servidor, programado em Java e responsável principalmente pelas validações e persistência dos dados. Pela divisão das funcionalidades da interface, lógica de negócio e operações com dados, pode-se considerar que o aplicativo é multicamadas.

O sistema possui três níveis de acesso, sendo eles: padrão, professor e administrador. O controle de permissões é feito por meio da *flag* administrador no cadastro dos professores.

É usuário padrão aquele que não possui cadastro. Usuário professor aquele que possui cadastro, porém não possui a *flag* ativada. E usuário administrador, aquele que possui cadastro e tem a *flag* administrador ativada. Por meio dessa informação, o aplicativo visual desenvolvido em Flex altera seu estado (*state)*, apresentando-se diferente para cada tipo de usuário.

A forma de implementação do *state* segue na Listagem 1.

```
<s:states>
    <s:State name="nrm" enterState="this.onEnterStateHandler(event)"/>
    <s:State name="prf" enterState="this.onEnterStateHandler(event)"/>
    <s:State name="adm" enterState="this.onEnterStateHandler(event)"/>
</s:states>
```

Listagem 1 - MXML dos states implementados no sistema

Como o sistema utiliza dois componentes separados para a parte visual, o *state* está presente em ambas. Para sincronizá-los foi necessário implementar um evento. A implementação do evento é apresentada na Listagem 2.

```
package com.flexnub.componentes.visual.event
{
    import flash.events.Event;

    public class ApplicationStateControlEvent extends Event
    {
        public static const STATE_CHANGE : String = 'stateChange';
        public var newState : String;

        public function ApplicationStateControlEvent(type:String, newState:String, bubbles:Boolean=false, cancelable:Boolean=false)
        {
            if (newState == null || newState != 'nrm' || newState != 'prf' || newState != 'adm'){
                this.newState = newState;
            } else {
                this.newState = 'nrm';
            }

            super(type, bubbles, cancelable);
        }
    }
}
```

Listagem 2 - Evento ApplicationStateControlEvent

O evento *ApplicationStateControl* é disparado quando o usuário efetua *login* no sistema, no componente cabeçalho. Por meio do clique no botão Entrar, o sistema captura os valores informados nos campos “Usuário” e “Senha” e caso eles estejam diferentes de “nulo” e “vazio”, é montado o objeto para enviar ao *backend*, no qual será efetuada a validação. A validação dos campos, montagem do objeto e a chamada ao método remoto *validarUsuarioSenha()* podem ser observados na Listagem 3.

```
private function validarUsuarioSenha() : void
{
    if (this.txiUsuario.text == null    || this.txiSenha.text == null ||
        this.txiUsuario.text == ""      || this.txiSenha.text == "")
    {
        Alert.show("Informe o usuários e a senha.", "Aviso:");
    } else {
        this.professor.prfusuario   = this.txiUsuario.text;
        this.professor.prfsenha     = this.txiSenha.text;

        this.professorService.validarUsuarioSenha(this.professor);
    }
}
```

Listagem 3 - Chamada ao método remoto validarUsuarioSenha()

No *backend*, o sistema procura por um registro Professor que possua o usuário e a senha informados e que esteja com o *flag* "ativo" no banco de dados. Essa busca é realizada por meio de um objeto do tipo Dao. O resultado da execução do comando SQL é montando em uma lista e retornado para o cliente. O método *validarUsuarioSenha(Professor professor)* executado no *backend* pode ser visualizado na Listagem 4.

```
public List<?> validarUsuarioSenha(Professor professor)
{
    this.setDao(new GenericDAO());
    this.getDao().abreTransacao();
    List<?> listaProfessores = this.getDao().listaSQL(this.getSQLBuscaProfessorUsuarioSenha(professor), new Professor());
    this.getDao().fechaTransacao();

    return listaProfessores;
}
```

Listagem 4 - Método validarUsuarioSenha(Professor professor)

A instrução SQL (*query*) executada pode ser observada na Listagem 5. Essa instrução é utilizada para recuperar usuários com permissões de acesso denominadas como "professor". Nessa instrução os itens usuário e senha são informados pelo usuário que está acessando o sistema.

```
private String getSQLBuscaProfessorUsuarioSenha(Professor professor)
{
    String sql =
            "select " +
                "professor.prfcodigo as \"prfcodigo\", " +
                "professor.prfusuario as \"prfusuario\", " +
                "professor.prfsenha as \"prfsenha\", " +
                "professor.prfnome as \"prfnome\", " +
                "professor.prfemail as \"prfemail\", " +
                "professor.prfadministrador as \"prfadministrador\", " +
                "professor.prfativo as \"prfativo\" " +
            "from siglas.professor " +
            "where " +
                "professor.prfusuario = '"  + professor.getPrfusuario() + "' and " +
                "professor.prfsenha = '"    + professor.getPrfsenha()   + "' and " +
                "professor.prfativo = true";
    return sql;
}
```

Listagem 5 - Query para recuperar professor com usuário e senha informados.

Ao retornar do método remoto, o sistema verifica o tamanho da lista retornada e caso este esteja vazia, emite uma mensagem ao usuário informado que o usuário e a senha informados são inválidos. Caso contrário, o sistema apanha o primeiro registro da lista e monta no objeto local utilizado para controle, validando *login* em seguida a propriedade "prfadministrador" que indica se o usuário é um administrador ou não, alterando o *state* do componente cabeçalho de acordo com esta propriedade. O trecho de código implementando este aspecto está apresentado na Listagem 6.

```
/**
 * Ao retornar do método remoto validarUsuarioSenha, verifica
 * os dados com o intuito de atualizar o state de acordo com a validade
 * das informações.
 */
private function onProfessorServiceValidarUsuarioSenhaResult(event : ResultEvent) : void
{
    var listaProfessor : ArrayCollection = ArrayCollection(event.result);

    if (listaProfessor.length == 0)
    {
        Alert.show("Usuário ou Senha inválidos.", "Aviso:");
    } else {
        this.professor = listaProfessor.getItemAt(0) as Professor;
        if (this.professor.prfadministrador) {
            this.currentState = 'adm';
        } else {
            this.currentState = 'prf';
        }
    }
}
```

Listagem 6 - Método responsável por alterar o state de acordo com astributos do usuário

Ao alterar o *state* do componente cabeçalho, um evento *EnterState* é disparado pelo próprio Flex, chamando o método *onEnterStateHandler()*. Esse método é responsável por

conduzir o processamento para os métodos específicos de troca de estado que efetuará as ações necessárias para adequar o componente cabeçalho ao novo estado. E, além disso, de disparar o evento que avisará o restante do aplicativo da alteração, sincronizando a outra parte do aplicativo, neste caso o componente Corpo. A implementação do método *onEnterStateHandler()* está na Listagem 7.

```
/**
 * Efetua os tratamentos necessários à troca de states.
 * @param FlexEvent
 */
protected function onEnterStateHandler(event : FlexEvent):void
{
    switch(this.currentState)
    {
        case 'nrm': this.onEnterNrmState(); break;
        case 'prf': this.onEnterPrfState(); break;
        case 'adm': this.onEnterAdmState(); break;
    }
}
```

Listagem 7 - Método para direcionar processamento de acordo com estado

O exemplo da implementação do método que tratará a troca para um estado específico pode ser observado na Listagem 8. Esse método é o *onEnterNrmState()*. Neste método é tratada a troca para o estado 'nrm'.

```
/**
 * Efetua os tratamentos necessários no início da entrada
 * ao state Nrm
 */
protected function onEnterNrmState() : void
{
    this.verificaBtnLogin();
    this.dispatchEvent(new ApplicationStateControlEvent(ApplicationStateControlEvent.STATE_CHANGE, 'nrm', true));
}
```

Listagem 8 - Exemplo de um método responsável pela troca de estado

O evento disparado nos métodos que tratam a troca de estado é capturado no Application pelo método *applicationStateChangeHandler()*, que acessa os componentes Cabeçalho e Corpo e altera seus estados. A Listagem 9 mostra o método *applicationStateChangeHandler()*.

```
/**
 * Sincroniza o aplicativo com o novo estado
 * @param ApplicationStateControlEvent
 */
protected function applicationStateChangeHandler(event:ApplicationStateControlEvent):voi
{
    this.cabecalho.changeState(event.newState);
    this.corpo.changeState(event.newState);
}
```

Listagem 9 - Método responsável por alterar o estado dos componentes do aplicativo

O sistema trabalha com registros gravados em bancos de dados. Esses registros são persistidos pelo *backend* programado em linguagem Java e utilizando o *framework* Hibernate para persistir os dados. O aplicativo Siglas utiliza um XML que contém as configurações para o Hibernate efetuar os mapeamentos necessários e abrir conexão com o banco de dados. O conteúdo do arquivo XML pode ser observado na Listagem 10.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration
DTD 3.0//EN" "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>

        <!-- Data Connection Settings -->
        <property name="hibernate.connection.dialect">org.hibernate.dialect.MySQLDialect</property>
        <property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
        <property name="hibernate.connection.url">jdbc:mysql://localhost:3306/siglas?autoReconnect=true</property>
        <property name="hibernate.connection.username">root</property>
        <property name="hibernate.connection.password">root</property>

        <!-- Hibernate session -->
        <property name="current_session_context_class">thread</property>

        <!-- Write SQL to log or console -->
        <property name="show_sql">true</property>

        <!-- Mapping Class -->
        <mapping class="com.flexnub.modelo.Professor"/>
        <mapping class="com.flexnub.modelo.Area"/>
        <mapping class="com.flexnub.modelo.Sigla"/>
    </session-factory>
</hibernate-configuration>
```

Listagem 10 - Arquivo hibernate.cfg.xml - configurações do framework Hibernate

As classes que representam a entidade do banco de dados possuem anotações em suas propriedades indicando que são colunas, se possuem alguma restrição, se são chaves e em qual tabela deve ser persistida. A partir dessas anotações, o *framework* identifica a forma como as propriedades devem se relacionar com o banco de dados e pode persistir diretamente o objeto instanciado a partir da classe. Um exemplo de classe com anotações pode ser vista na Listagem 11.

```java
/**
 * @author Kegan
 *
 */
@Entity
@Table(name="professor")
public class Professor implements Serializable {

    private static final long serialVersionUID = 1L;
    private Integer prfcodigo;
    private String  prfusuario;
    private String  prfsenha;
    private String  prfnome;
    private String  prfemail;
    private Boolean prfadministrador;
    private Boolean prfativo;

    @Id
    @GeneratedValue
    @Column(name="prfcodigo")
    public Integer getPrfcodigo() {
        return prfcodigo;
    }

    public void setPrfcodigo(Integer prfcodigo) {
        this.prfcodigo = prfcodigo;
    }

    @Column(name="prfusuario")
    public String getPrfusuario() {
        return prfusuario;
    }

    public void setPrfusuario(String prfusuario) {
        this.prfusuario = prfusuario;
    }
```

Listagem 11 - Trecho da classe Professor mostrando as anotações do Hibernate

O Hibernate possui classes próprias para efetuar a persistência no banco de dados. As mais importantes são as classes *SessionFactory* e *Session*. A primeira é responsável por abrir ou iniciar uma sessão com o banco de dados. A segundo é responsável por permitir ao programador utilizar métodos para abrir e fechar transações, assim como efetuar consultas SQL e operações no banco passando diretamente o objeto como parâmetro. Na Listagem 12 é possível observar o objeto Professor sendo persistido diretamente pelo método gravar do objeto *GenericDAO*. Esse método apenas acessa o método *save* do objeto *Session*.

```
public void gravarSolicitacaoProfessor(Professor professor) throws Exception
{
    try {
        this.setDao(new GenericDAO());
        this.getDao().abreTransacao();
        this.getDao().gravar(professor);
        this.getDao().commit();
        this.getDao().fechaTransacao();
    } catch (Exception ex) {
        this.getDao().rollback();
        this.getDao().fechaTransacao();
        String mensagem = ex.getMessage();
        if (ex instanceof ConstraintViolationException){
            mensagem = "Usuário já cadastrado.";
        }
        throw new Exception(mensagem);
    }

}
```

Listagem 12 - Método que persiste o objeto professor

O Flex permite identificar a partir de que classe um determinado objeto foi instanciado no *backend*. Isso é feito a partir da *tag* [RemoteClass(alias=””)]. Nessa *tag*, entre as aspas deve estar o endereço da classe equivalente no Java. Utilizando esse recurso é possível enviar um objeto instanciado no Java para o Flex, convertendo-o para a classe equivalente na outra linguagem e vice-versa. A implementação da classe Professor na linguagem ActionScript, com a *tag* RemoteClass indicando o endereço da classe equivalente em Java, é apresentada na Listagem 13.

```
package com.flexnub.classe.modelo
{
    [Bindable]
    [RemoteClass(alias="com.flexnub.classe.modelo.Professor")]
    public class Professor
    {
        public var prfcodigo        : Object;
        public var prfusuario       : String;
        public var prfsenha         : String;
        public var prfnome          : String;
        public var prfemail         : String;
        public var prfadministrador : Boolean;
        public var prfativo         : Boolean;

        public function Professor()
        {
        }
    }
}
```

Listagem 13 - Implementação da classe professor em ActionScript

5 CONCLUSÃO

O objetivo pretendido com esse trabalho foi alcançado. Um sistema *web*, denominado RIA pelos conceitos e recursos utilizados em sua implementação, foi desenvolvido. Para que esse desenvolvimento pudesse ser realizado alguns conceitos teóricos foram identificados como necessários, incluindo aplicações Internet ricas.

Com o desenvolvimento deste trabalho percebeu-se que o uso da ferramenta Balsamiq Mockups é bastante útil, porque ela permite esboçar as telas do aplicativo de forma relativamente semelhante a um processo manual. Com isso foi possível experimentar leiautes distintos, com a facilidade de simplesmente colocar componentes na tela. Essa ferramenta é interessante para projetar a interface de aplicativos e especialmente para aplicativos *web*. Isso pela diversidade de componentes que as RIAs podem ter na composição da sua interface.

Os recursos que Flex oferece para a composição da interface tornam a interação do usuário mais facilitada e mais agradável. O agradável é colocado no sentido que no lugar de *links* podem ser colados botões, *grids* e uma série de elementos que eram pertinentes somente às aplicações *desktop*. Em termos técnicos, do ponto de vista do programador. Uma das grandes vantagens do uso de Flex é o tratamento dos componentes da interface como objetos. À semelhança dos objetos em uma linguagem orientada a objetos, como Java, por exemplo.

Não foram encontrados problemas ou dificuldades críticos com o uso das tecnologias Java e Flex durante a implementação do sistema. A relativa experiência do autor deste trabalho com o uso dessas tecnologias auxiliou que o desenvolvimento ocorresse com certa fluência. Uma das principais dificuldades foi entender a melhor forma de representar o sistema em termos de classes e tabelas de banco de dados e mesmo da sua interface. A ferramenta Balsamiq auxiliou nessas definições pela possibilidade de realizar os esboços do leiaute do aplicativo.

O sistema que foi implementado será instalado em uma máquina servidora na própria Universidade e o acesso será permitido por meio do portal do aluno. É possível que por meio do uso por usuários efetivos sejam identificadas alterações e melhorias. Contudo, de uma análise do ponto de vista dos requisitos definidos, o sistema em sua versão atual atende a esses requisitos.

REFERÊNCIAS

ADOBE. **Adobe flex 3**. **Now building on flex**. Disponível em: <http://www.adobe.com/products/flex/>. Acesso em: 12 mar. 2011a.

ADOBE. **Adobe open source blazeds**. Disponível em: <http://opensource.adobe.com/wiki/display/blazeds/BlazeDS>. Acesso em: 12 mar. 2011b.

ADOBE. **AMF 3 specification. Action message format - AMF 3**. Category ActionScript Serialization. Adobe Systems Inc. Disponível em <http://opensource.adobe.com/wiki/download/attachments/1114283/amf3_spec_05_05_08.pdf>. Acesso em: 12 mar. 2011c.

ASTAH. **Astah community**. Disponível em <http://astah.change-vision.com/en/product/astah-community.html>. Acesso em: 08 mar. 2011.

BALSAMIQ. **Balsamiq mockup**. Disponível em <http://www.balsamiq.com/products/mockups>. Acesso em: 29 mar. 2011.

BOOCH G. **Object-oriented analysis and design. With applications**, 2a. ed. Addison-Wesley, 1998.

BOZZON, A. COMAI, S. **Conceptual modeling and code generation for rich internet applications.** 6th International Conference on Web Engineering (ICWE '06), 2006, p. 353-360.

COMAI, S. CARUGHI, G. T. **A behavioral model for Rich Internet, Lecture Notes in Computer Science**, vol. 4607/2007, 2007, p. 364-369.

DEITEL, H.M.; DEITEL, P.J. **Java: como programar**. Porto Alegre: Bookman, 2001.

DUHL J., Rich internet applications, **IDC White Papers**. Disponível em http://www.idc.com, 2003.

FUKUDA, H., YAMAMOTO, Y. **A system for supporting development of large scaled rich internet applications**. In: 23rd IEEE/ACM International Conference on Automated Software Engineering (ASE 2008), 2008, p. 459-462.

HIBERNATE. **Relational persistence for java and .net.** Disponível em: <https://www.hibernate.org>. Acesso em: 12 abr. 2011.

LOOSLEY, C. **Rich internet applications: design, measurement and management chalenges** 2006. Disponível em: <http://www.keynote.com/docs/whitepapers/> Acesso em: 5 abr. 2011.

MILANI, A. **MySQL – guia do programador**. São Paulo: Novatec, 2007.

MULLET, K. The essence of effective rich internet applications. **Macromedia Whitepapers**. 2003.

MYECLIPSE. **MyEclipse**. Disponível em: <http://www.myeclipseide.com>. Acesso em: 29 mar. 2011.

MYSQL ADMINISTRATOR. **Introdução a mysql administrator**. Disponível em: <http://dev.mysql.com/doc/administrator/pt/mysql-administrator-introduction.html>. Acesso em: 20 abr. 2011.

MYSQL. **MysSQL.** Disponível em: <http://www.mysql.com>. Acesso em: 27 abr. 2011.

PRECIADO, J.C. et al. **Necessity of methodologies to model rich internet applications.** In: Seventh IEEE International Symposium on *Web* Site Evolution (WSE'05), 2005, p. 7-13.

PRESSMAN, R. **Engenharia de software**, 5ª ed. 2002. Rio de Janeiro: McGrawHill.

SCHMELZER, R. **Rich internet applications: market trends and technologies.** Market Trends and Approaches, jul. 2006, p. 1-22.

TOMCAT. A**pache tomcat**. Disponível em <http://tomcat.apache.org/>. Acesso em: 12 mar. 2011.

WORKBENCH. **MySQL workbench**. Disponível em <http://wb.mysql.com/>. Acesso em: 02 fev. 2011.