UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE SISTEMAS DE INFORMAÇÃO – BACHARELADO

APLICATIVO WEB PARA DEFINIÇÃO DO MODELO LÓGICO NO PROJETO DE BANCO DE DADOS RELACIONAL

JUAREZ BACHMANN

BLUMENAU
2007

2007/1-11

JUAREZ BACHMANN

APLICATIVO WEB PARA DEFINIÇÃO DO MODELO LÓGICO NO PROJETO DE BANCO DE DADOS RELACIONAL

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Sistemas de Informação - Bacharelado.

Prof. Alexander Roberto Valdameri, Mestre - Orientador

BLUMENAU
2007

2007/1-11

APLICATIVO WEB PARA DEFINIÇÃO DO MODELO LÓGICO NO PROJETO DE BANCO DE DADOS RELACIONAL

Por

JUAREZ BACHMANN

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:

Presidente:	Prof. Alexander Roberto Valdameri, Mestre – Orientador, FURB
Membro:	Prof. Everaldo Artur Grahl, Mestre – FURB
Membro:	Prof. Adilson Vahldick, Especialista – FURB

Blumenau, 26 de junho de 2007

Dedico este trabalho à minha namorada, familiares e amigos, que com seu apoio, amor e carinho me ajudam a enfrentar qualquer obstáculo, em todos os aspectos da minha vida.

AGRADECIMENTOS

A Deus, por ser meu guia e refúgio.

À minha família, que fez de mim o que sou.

À minha namorada, Clauciméri, por seu amor, compreensão e companheirismo em todos os momentos.

Ao professor Adilson Vahldick, pelos conhecimentos transmitidos, indispensáveis à realização deste trabalho.

Ao meu orientador, Alexander Roberto Valdameri, por todo o auxílio concedido para a realização deste trabalho.

Se você pode sonhar, pode fazer.

Walt Disney

RESUMO

Este trabalho apresenta a implementação de um aplicativo para definição do modelo lógico nos projetos de bancos de dados relacionais, desenvolvido para o ambiente web utilizando a plataforma Java, a arquitetura MVC e alguns padrões de projeto, como DAO e *Command*. Para tornar a interface do sistema mais amigável ao usuário, foi utilizado, em alguns momentos, o conjunto de técnicas conhecido por Ajax, disponibilizando recursos semelhantes aos de aplicativos *desktop*. O aplicativo desenvolvido permite o controle de projetos de bancos de dados e suas versões, além de possibilitar a restrição do acesso dos usuários a determinados projetos. Também permite que sejam criados diagramas para modelar as tabelas, colunas e relacionamentos de forma gráfica. Por fim, há a possibilidade de gerar *scripts* com os comandos SQL necessários para criar o banco de dados modelado e também para atualizar um banco de dados já existente, aplicando as alterações efetuadas entre duas versões.

Palavras-chave: Modelo lógico de dados. Bancos de dados relacionais. Ambiente web. Ajax. Padrões de projeto.

ABSTRACT

This work presents the implementation of software to definition of the logical data model on relational database projects, developed for the web environment using the Java platform, MVC architecture and some design patterns, like DAO and Command. To make the system interface friendlier to the user, at some points it was used a set of techniques called Ajax, availability similar to the recourses that exists at the desktop applications. The developed software allows the control of database projects and versions, besides making possible the users access restriction to determined projects. It also allows creating diagrams to model tables, columns and relationships graphically. Finally, there is the possibility to generate scripts with the SQL commands needed to create a modeled database and also to bring up to date an existing database, applying the changes done between the two versions.

Key-words: Logical data model. Relational databases. Web environment. Ajax. Design patterns.

LISTA DE ILUSTRAÇÕES

SUMÁRIO

1 INTRODUÇÃO

Segundo Heuser (2000, p. 77), no mercado de software há um claro predomínio dos sistemas gerenciadores de banco de dados (SGBD) do tipo relacional, até mesmo nas plataformas de grande porte (grandes corporações), onde sistemas hierárquicos, de rede ou proprietários vêm sendo substituídos por sistemas relacionais. Esta realidade faz com que seja necessário dar um enfoque cada vez mais expressivo no processo de projeto de banco de dados relacionais, para garantir que a estrutura criada atenda requisitos como performance aceitável na recuperação dos dados, por exemplo.

De acordo com Heuser (2000, p. 9) o processo de projeto de banco de dados pode ser dividido, basicamente, em duas fases:

a) modelagem conceitual, onde é construído um diagrama que captura as necessidades da organização em termos de armazenamento de dados, sem a preocupação com a forma como esta estrutura será implementada;
b) projeto lógico, que transforma o modelo conceitual em um modelo lógico, definindo como o banco de dados será implementado em um tipo específico de SGBD.

Cougo (1997, p. 157) enfatiza que muitos autores que abordam o tema da modelagem de dados, direcionam o processo diretamente para o nível lógico. Muitas empresas desenvolvedoras de software seguem o mesmo princípio, ou seja, não executam a etapa de modelagem conceitual e, através de ferramentas do tipo *Computer Aided Software Engineering* (CASE), implementam apenas o modelo lógico.

A variedade de ferramentas CASE existentes no mercado com o propósito de auxiliar na modelagem de bancos de dados relacionais é muito grande, sendo que na sua grande maioria elas têm características em comum, como a possibilidade de modelar os dados de forma gráfica (desenhando tabelas e relacionamentos) e salvar os modelos criados em arquivos binários, que podem ser enviados a outras pessoas e utilizados por elas, desde que estas utilizem a mesma ferramenta. Outra característica marcante presente nestes softwares é a necessidade de instalá-los no computador do usuário, isto é, são aplicações *desktop*, sendo que algumas necessitam de licença para serem executadas e outras são consideradas de código aberto, podendo ser instaladas e utilizadas sem a necessidade de pagamento.

Algumas aplicações *desktop* exigem, além de um alto custo para aquisição, instalação e manutenção, uma estrutura de hardware que as comporte, sendo que o nível de exigência de

hardware é diretamente ligado ao tamanho e sofisticação dos softwares. Outro fator importante deste tipo de software é a dificuldade que pode haver em atualizá-lo, caso o mesmo esteja instalado em inúmeras máquinas dentro de uma empresa, pois a atualização precisará ser feita em cada uma delas, o que pode demandar muito tempo e recursos financeiros.

Uma alternativa interessante para amenizar eventuais problemas decorrentes destas características dos aplicativos *desktop* é a internet, que vem passando por transformações significativas nos últimos anos. Neste momento, a web está virando uma plataforma: todo tipo de aplicativo é executado no *browser* (FORTES, 2006, p. 44). Inicialmente, foram os serviços de e-mail que saíram do *desktop* e foram disponibilizados na grande rede mundial de computadores. Agora, já há editores de textos, planilhas, agendas, diários, álbuns de fotos, enciclopédias e muitos outros serviços como estes. Além disso, diversos softwares corporativos são executados nos navegadores, como os portais de compras, por exemplo. Kurniawan (2002, p. XIV), ratifica esta situação, afirmando que um *browser* não é mais utilizado apenas para exibir páginas estáticas na internet, pois é muito comum vê-lo sendo utilizado como um cliente de um aplicativo.

Diante deste contexto, pode-se dizer que há uma grande tendência de transformar os aplicativos *desktop* em ferramentas para o ambiente web e os softwares de modelagem de banco de dados também podem ser enquadrados nesta situação.

1.1 OBJETIVOS DO TRABALHO

O principal objetivo deste trabalho é criar um aplicativo no ambiente web para definição dos modelos lógicos nos projetos de banco de dados relacional.

Os objetivos específicos do trabalho são:

a) avaliar o conjunto de técnicas para programação de páginas web denominado *Asyncronous Javascript and XML* (Ajax) na implementação da ferramenta, analisando seus benefícios e limitações na criação de páginas dinâmicas em conjunto com a especificação Java Enterprise Edition (JEE), até pouco tempo conhecida como Java 2 Enterprise Edition (J2EE);
b) utilizar a arquitetura *Model-View-Controller* (MVC) na implementação da ferramenta, visando estruturar o código-fonte de forma que a lógica e as regras de

negócio fiquem separadas da interface;

c) desenvolver a ferramenta permitindo que através dela seja possível definir a estrutura de um banco de dados relacional (tabelas, colunas e relacionamentos) de forma independente do SGBD onde o banco será criado, utilizando tipos de dados para as colunas e recursos para definição da estrutura próprios desta ferramenta, ao invés de tipos e recursos específicos de algum SGBD.

1.2 ESTRUTURA DO TRABALHO

Este trabalho está organizado em quatro capítulos.

O primeiro é onde se tem a introdução, a justificativa do trabalho, o objetivo geral e objetivos específicos.

No segundo, há a fundamentação teórica do trabalho, discorrendo a cerca de temas como projeto de banco de dados e bancos de dados relacionais, além de apresentar as principais tecnologias utilizadas no desenvolvimento do sistema, bem como trabalhos correlatos ao presente.

O terceiro capítulo descreve o processo de análise e implementação do aplicativo, sendo ilustrado por diagramas que visam facilitar a compreensão do mesmo. Também contem um exemplo da operacionalidade do aplicativo, onde a modelagem de um projeto de banco de dados relacional é demonstrada passo-a-passo no sistema.

Por fim, há o quarto capítulo, onde constam a conclusão e sugestões de extensões deste trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo visa disponibilizar a descrição de conceitos, métodos, técnicas e tecnologias envolvidas na realização do trabalho.

As primeiras seções trazem os conceitos básicos relacionados à área de banco de dados, como projeto de banco de dados e bancos de dados relacionais. As seções seguintes abordam as técnicas e tecnologias utilizadas na implementação da ferramenta, como Ajax, por exemplo. Por fim, há a seção que versa sobre trabalhos correlatos, relacionando suas características mais importantes e as semelhanças existentes entre eles e o presente trabalho.

2.1 SISTEMA GERENCIADOR DE BANCO DE DADOS (SGBD)

Um SGBD é, de forma simplificada e segundo Date (2000, p. 37), um software que trata de todo o acesso ao banco de dados. De forma semelhante, Heuser (2000, p. 5) descreve o SGBD como um software que incorpora as funções de definição, recuperação e alteração de dados em um banco de dados, sendo que além destas funcionalidades, Date (2000, p. 38) cita a segurança e a integridade dos dados como fatores de responsabilidade do SGBD.

Há vários tipos de SGBD, dentre os quais pode-se citar o relacional, o hierárquico, o de rede e o orientado a objetos, cada qual possuindo características diferenciadas na implementação das funcionalidades descritas anteriormente.

2.2 BANCO DE DADOS RELACIONAL

Date (2000, p. 67) descreve um banco de dados relacional como sendo uma coleção de tabelas, também chamadas de relações. Estas tabelas são formadas por colunas (ou atributos), sendo que cada uma delas pode receber um valor pertencente a um domínio. “Um domínio não é nada mais nada menos que um **tipo de dados** (ou tipo, para abreviar) – possivelmente um tipo simples **definido pelo sistema** como INTEGER ou CHAR [...]” (BOND et al., 2003, p. 98, grifo do autor). Cada conjunto de atributos forma uma linha, também chamada de

registro, sendo que uma tabela pode possuir inúmeros registros.

Cougo (1997, p. 162) cita outro conceito básico relativo aos bancos de dados relacionais: o conceito de chave primária. A chave primária garante que não existirão dois registros iguais dentro da mesma tabela, sendo formada por atributos que, por análise prévia ou conhecimento do ambiente que está sendo modelado, sabe-se que não terão valores repetidos em dois ou mais registros.

Além dos anteriormente citados, o conceito de chave estrangeira também é fundamental no banco de dados relacional, pois é através dele que os relacionamentos entre tabelas são estabelecidos (COUGO, 1997, p. 166). A chave estrangeira nada mais é do que a chave primária (conjunto de atributos únicos) de uma tabela que é repetida em outra tabela, sendo que os valores da chave na segunda tabela devem existir na tabela original para garantir a integridade das informações.

2.3 PROJETO DE BANCO DE DADOS

Segundo Date (2000, p. 287), o projeto de banco de dados é o processo através do qual se define uma estrutura lógica adequada para armazenar algum conjunto de dados que precisa ser representado num banco de dados. Heuser (2000, p. XIII) afirma que este processo pode ocorrer em três etapas. A primeira é chamada de modelagem conceitual, que procura capturar formalmente quais as informações que precisarão ser gravadas num banco de dados, sem a preocupação com o tipo de SGBD a ser utilizado.

A segunda etapa é o projeto lógico, que define as estruturas de dados necessárias para atender os requisitos identificados na modelagem conceitual, sendo que estas estruturas dependerão diretamente do tipo de SGBD utilizado, porém, segundo Date (2000, p. 287) este projeto é ou deve ser bastante independente de algum SGBD específico.

Finalmente, a terceira etapa é o projeto físico, que define os parâmetros físicos de acesso ao banco de dados, relacionados principalmente às configurações do SGBD e do hardware, procurando otimizar a performance do sistema como um todo.

2.4 MODELO LÓGICO DE DADOS

De acordo com Heuser (2000, p. 7), “um modelo lógico é a descrição de um banco de dados no nível de abstração visto pelo usuário do SGBD”, ou seja, é um modelo dependente do tipo particular de SGBD (relacional, hierárquico, rede, etc.) que está sendo usado. Cougo (1997, p. 29), segue a mesma linha, definindo o modelo lógico como sendo aquele em que os objetos, características e relacionamentos são representados de acordo com regras de implementação e limitações impostas por algum tipo de tecnologia, isto é, algum tipo de SGBD.

No caso dos SGBD do tipo relacional, o modelo lógico “deve definir quais as tabelas que o banco contém e, para cada tabela, quais os nomes das colunas” (HEUSER, 2000, p. 7). Cougo (1997, p. 29) acrescenta ainda que este modelo deve ser elaborado respeitando conceitos como chaves de acesso, controles de chaves duplicadas e integridade referencial.

2.5 PADRÕES DE PROJETO DE SOFTWARE

No desenvolvimento de um software é comum que os analistas e projetistas deparem-se com situações ou problemas pelos quais já passaram anteriormente, ao desenvolver outros softwares ou rotinas. Na maioria destes casos, utilizam sempre o mesmo tipo de solução para o mesmo tipo de problema, ou seja, há um padrão que pode ser seguido.

Bond et al. (2003, p.719), descreve os padrões dentro da área de software como uma forma de capturar parte da experiência dos arquitetos e projetistas de sucesso, para que ela possa ser aplicada mais amplamente, ou seja, “um padrão é uma idéia reutilizável sobre como resolver um problema em particular, encontrado no domínio da arquitetura ou projeto” (BOND et al., 2003, p. 721).

Os padrões podem ser divididos em tipos, sendo que segundo Bond et al. (2003, p. 722), os mais comuns são:

a) os padrões de arquitetura, que definem o estilo geral do sistema, como a quantidade de camadas que a aplicação terá e o relacionamento entre elas, por exemplo;
b) os padrões de projeto, que se encontram no nível dos artefatos como classes e

componentes.

Além destes, também há os chamados padrões do JEE, que são um conjunto de padrões que têm sido identificados dentro das soluções com base no JEE, sendo que alguns destes são adaptações de padrões gerais já identificados e utilizados anteriormente em outras plataformas (BOND et al., 2003, p. 723).

A seguir, são descritos os padrões utilizados neste trabalho.

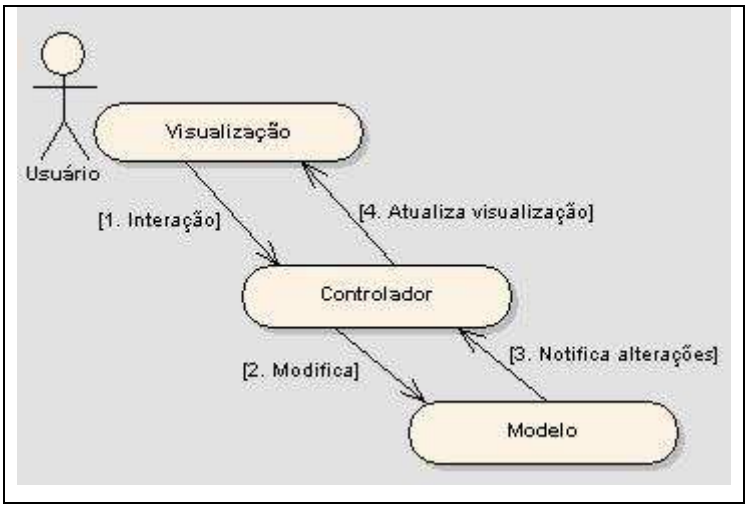
2.5.1 *Model-View-Controller* (MVC)

Gamma et al. (2000, p. 20), descreve MVC como sendo um padrão de arquitetura que divide o software em três tipos de objetos. O primeiro é o Modelo (*Model*), que consiste na camada onde ocorre o processamento das informações e os dados são gravados nos bancos de dados e recuperados dos mesmos, ou seja, é no Modelo que ficam as regras de negócio. O segundo tipo de objeto é a Vista ou Visão (*View*), cuja função é apresentar as informações ao usuário, ou seja, é a interface do sistema. O terceiro e último tipo é o Controlador (*Controller*), que define como a interface reage às entradas do usuário. A Visão sempre deve refletir a situação atual do Modelo, sendo que o Controlador é responsável por controlar o fluxo entre as duas outras camadas.

Ainda segundo Gamma et al. (2000, p. 20) “MVC separa esses objetos para aumentar a flexibilidade e a reutilização”, pois esta separação permite que o mesmo Modelo possa ser utilizado em inúmeras aplicações, através de Visões diferentes. Além disso, esta arquitetura facilita a implementação, evitando que as regras de negócio e acessos ao banco de dados façam parte do mesmo código-fonte responsável pela interface do sistema.

A principal regra do MVC é que as camadas de Visão e Modelo não devem ter acesso direto uma à outra: o Controlador deve intermediar as ações entre estas duas camadas. Por exemplo: quando o usuário preencher um formulário e pressionar o botão para gravar os dados, a Visão (Interface) avisará ao Controlador que o botão foi pressionado. Neste momento, o Controlador acionará o Modelo para que ele efetue as alterações necessárias. Em seguida, se o Controlador perceber que as alterações no Modelo exigem uma atualização da Interface, esta última será informada sobre a forma como deverá atualizar-se (CRANE; PASCARELO; JAMES, 2007, p. 62).

A figura 1 ilustra o funcionamento do padrão MVC.



Fonte: adaptado de Crane, Pascarello e James (2007, p. 62)

Figura 1 – Funcionamento do padrão MVC

Na prática, o MVC permite que os analistas e programadores responsáveis pelo Modelo (regras de negócio, gravação de dados, etc.) não se preocupem com a interface do sistema e vice-versa. A única coisa que precisa existir é um acordo sobre a forma como vão se comunicar, que é tratada no Controlador. Por exemplo: supondo que se trate de um sistema de controle de estoques e que em determinado momento a interface precise exibir uma tela com todos os produtos cadastrados, a Visão solicitará a lista de produtos ao Controlador, que por sua vez acionará o Modelo e este lhe fornecerá a respectiva listagem, que será devolvida à Visão. O acordo, neste caso, é que o Controlador precisa fornecer uma listagem à interface, sendo que a forma como será feita a exibição e a carga dos dados fica a critérios dos responsáveis por cada camada e isto não afeta as demais, o que ratifica que este padrão facilita em muito a reutilização e a flexibilidade do software.

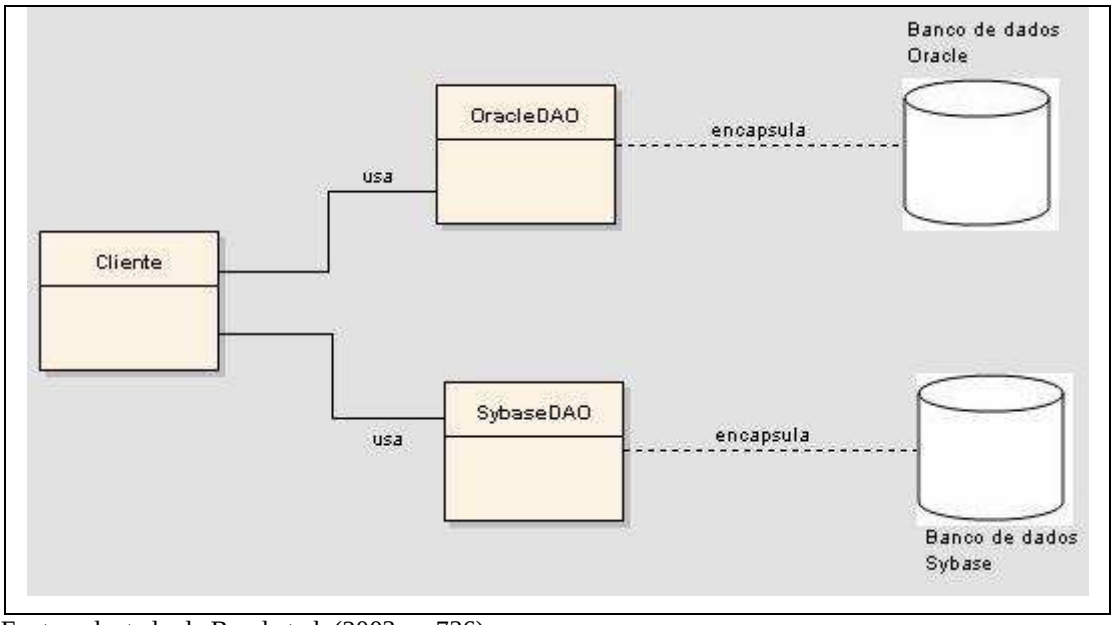
2.5.2 *Data Access Object* (DAO)

Segundo Bond et al. (2003, p. 728) o DAO é um padrão do JEE que visa criar objetos que encapsulem o acesso aos dados por trás de uma interface comum, que pode ser implementada de diferentes formas para diferentes fontes de dados.

Sullivan (2003) explica que os desenvolvedores JEE utilizam este padrão para separar a lógica de acesso a dados (baixo nível) da lógica das regras de negócio (alto nível). Com isso, de acordo com Bond et al. (2003, p.735), ficam amenizados dois problemas que ocorrem com freqüência nos sistemas onde o código de acesso a dados é mesclado com a lógica do negócio: dificuldade de manutenção e pouca flexibilidade. Isto ocorre porque ao utilizar este padrão do

JEE, uma alteração no código de acesso a dados só precisará ser feita no método específico da classe DAO em questão, não sendo necessário procurar e alterar inúmeros pontos do sistema que possuem os mesmos comandos SQL, por exemplo. Além disso, a fonte de dados do sistema se torna plugável, ou seja, após definir a interface Java que será implementada (métodos de criação, recuperação, atualização e exclusão de dados) é possível estendê-la para tratar diferentes tipos de fontes de dados. Para facilitar, pode ser implementada uma classe que funcione como uma fábrica que determina o tipo de DAO que será usado, permitindo que a seleção da implementação DAO a ser utilizada seja feita facilmente em tempo de execução.

A figura 2 exemplifica o funcionamento do padrão DAO.



Fonte: adaptado de Bond et al. (2003, p. 736)

Figura 2 – Funcionamento do padrão DAO

2.5.3 *Command*

Segundo Gamma et al. (2000, p. 222), o padrão *Command* visa encapsular uma solicitação como um objeto, permitindo parametrizar clientes com diferentes solicitações, enfileirar ou fazer o registro (*log*) de solicitações e também suportar operações que podem ser desfeitas.

Este padrão, também conhecido como *Action* ou *Transaction*, possibilita também estruturar um sistema em torno de operações de alto nível, estrutura esta muito comum em sistemas que suportam transações. Assim, pode-se dizer que o padrão *Command* fornece uma maneira de modelar transações (GAMMA et al., 2000, p. 225).

O princípio básico desse padrão é a criação de uma classe abstrata `Command`, com uma operação abstrata `execute()`, sendo que esta operação é implementada nas subclasses concretas de `Command`.

A figura 3 apresenta um exemplo de diagrama das classes `Command` de um editor de textos.

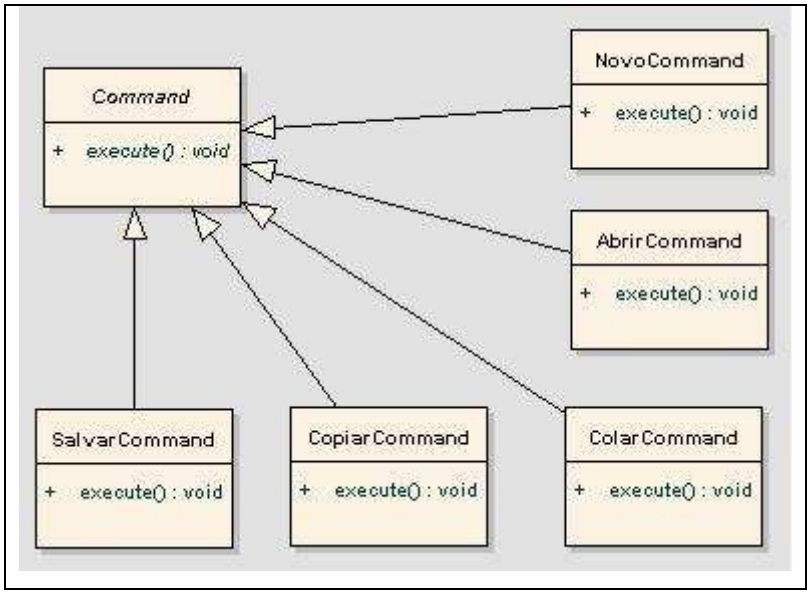


Figura 3 – Exemplo de diagrama de classes `Command` de um editor de textos

No editor de textos representado na figura 3, cada botão do menu que fosse acionado criaria uma instância de uma subclasse de `Command`, de acordo com sua funcionalidade. Um botão chamado Salvar Documento, por exemplo, instanciaria um objeto da classe `SalvarCommand` e em seguida dispararia o seu método `execute()`, sendo que o mesmo aconteceria para os outros botões e classes. Desta forma, cada operação é executada na sua totalidade por uma classe específica, facilitando a compreensão e manutenção do sistema.

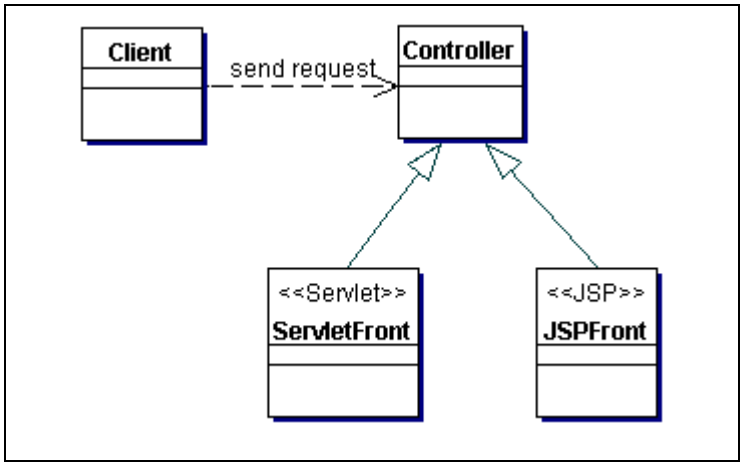
2.5.4 *Front Controller*

Em Bond et al. (203, p.726) *o Front Controller* é definido como um padrão JEE onde um componente intercepta a requisição do usuário e direciona ou agrega valor a mesma.

A definição encontrada em Sun Microsystems (2002b) diz que este padrão consiste em definir um componente simples que seja responsável pelo processamento das requisições da aplicação. Este componente recebe a requisição, encaminha-a para o componente responsável por processá-la (outra classe ou *servlet*) e após o processamento seleciona a página que deve ser apresentada ao usuário. Além disso, este componente centraliza funções relativas à

segurança e tratamento de erros, facilitando a manutenção do software (SUN MICROSYSTEMS, 2002a).

O diagrama de classes na figura 4 apresenta a estrutura deste padrão.



Fonte: Sun Microsystems (2002a)

Figura 4 – diagrama de classes do padrão *Front Controller*

2.6 AJAX

O termo Ajax foi criado por Jesse James Garret em fevereiro de 2005, como um acrônimo para *Asynchronous JavaScript and XML*, mas atualmente é usado para englobar todas as tecnologias que possibilitam ao navegador se comunicar com o servidor sem atualizar a página atual (ASLESON;SCHUTTA, 2006, p. 14).

Segundo Asleson e Schutta (2006, p. 13), o Ajax pode ser considerado como uma técnica e não uma tecnologia específica. Esta técnica agrega tecnologias como JavaScript e *eXtensive Markup Language* (XML) para tornar as páginas web mais dinâmicas, disponibilizando recursos como criação de menus de contexto que aparecem ao clicar com o botão direito do *mouse*, possibilidade de deslizar controles (arrastar e soltar) e trocar informações com o servidor de forma assíncrona, sem a necessidade de atualizar toda a página ao fazê-lo. Estes recursos possibilitam que as aplicações web tenham uma interface semelhante à das aplicações *desktop*, tornando-as mais atrativas para usuários adaptados aos recursos existentes nas aplicações residentes em seus computadores.

Crane, Pascarello e James (2007, p. 22) enfatizam que as quatro principais tecnologias que compõem o Ajax são:

a) JavaScript: linguagem de criação de *scripts* de uso geral, projetada para ser

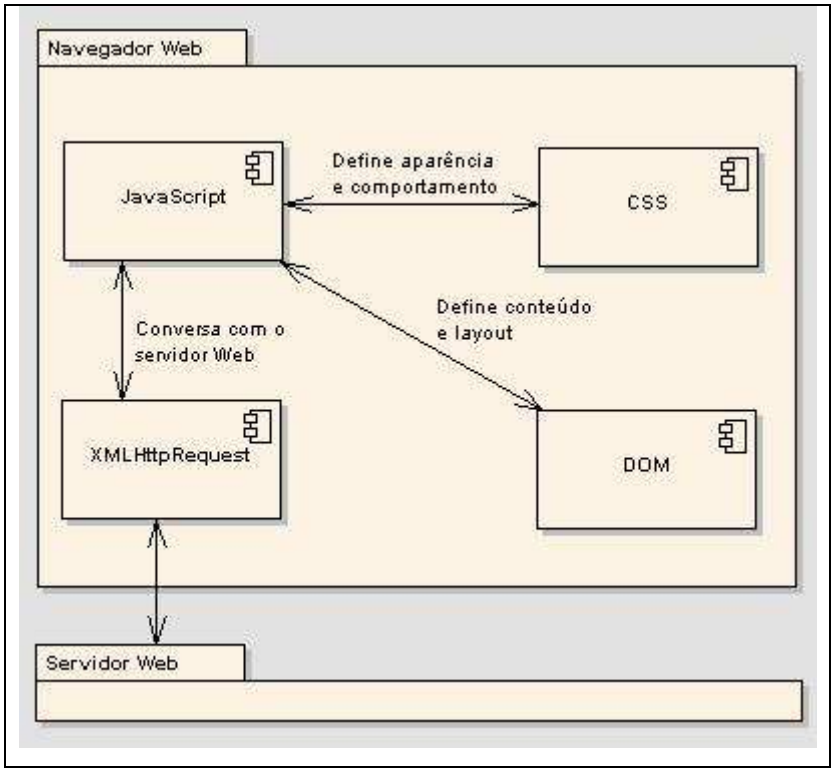
embutida nos aplicativos. É utilizada para tratar eventos gerados no navegador;

b) *Cascading Style Sheets* (CSS): CSS oferecem uma forma de definir estilos visuais reutilizáveis para elementos das páginas web. Em um aplicativo Ajax, a estilização da interface com o usuário pode ser modificada interativamente por meio de CSS;

c) *Document Object Model* (DOM): o DOM permite que através do JavaScript a estrutura das páginas web possa ser alterada, como se fosse um conjunto de objetos programáveis. Com isso, é possível que o aplicativo Ajax modifique a interface com o usuário instantaneamente, redesenhando partes da página;

d) objeto `XMLHttpRequest`: este objeto permite a comunicação do navegador com o servidor web de forma assíncrona, ou seja, como uma atividade que ocorre em segundo plano, normalmente com os dados trafegando no formato XML.

O relacionamento entre estes quatro elementos numa aplicação Ajax pode ser visto na figura 5.



Fonte: adaptado de Crane, Pascarello e James (2007, p. 23)

Figura 5 – Relacionamento entre os componentes do Ajax

Todas estas tecnologias são tratadas no *browser*, ou seja, "o Ajax é uma abordagem do lado cliente e pode interagir com a J2EE, .NET, PHP, Ruby e scripts CGI – realmente não depende do servidor" (ASLESON;SCHUTTA, 2006, p. 13).

A tecnologia mais recente (e talvez mais importante) relacionada ao termo, é o objeto `XMLHttpRequest` (XHR), que foi lançado no Internet Explorer 5, em 1999, como um

controle ActiveX e que, inicialmente, só era suportado por este navegador, mas atualmente é suportado por quase todos os navegadores disponíveis (ASLESON;SCHUTTA, 2006, p. 13). Este objeto é responsável pela comunicação assíncrona entre o navegador e o servidor web, sendo que os passos para a sua utilização serão descritos a seguir.

Em primeiro lugar, é necessário criar um objeto `XMLHttpRequest` usando JavaScript, sendo que para tanto são necessários alguns cuidados, pois o navegador Internet Explorer implementa `XMLHttpRequest` como um objeto ActiveX enquanto o Firefox, Safari e Opera o implementam como um objeto JavaScript nativo (ASLESON;SCHUTTA, 2006, p. 23). O quadro 1 mostra um exemplo de código-fonte utilizado para criar o objeto.

```
var xmlHttp;

function createXMLHttpRequest(){
  if (window.ActiveXObject) { //indica que o navegador é o IE
      xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
  }
  else if (window.XMLHttpRequest){
      xmlHttp = new XMLHttpRequest();
  }
}
```

Fonte: Asleson e Schutta (2006, p. 24).

Quadro 1 – Criação de uma instância da classe `XMLHttpRequest`

Para realizar a comunicação com o servidor, o objeto `XMLHttpRequest` dispõe de alguns métodos e propriedades específicos. Os principais métodos estão relacionados no quadro 2.

Método	**Descrição**
`abort()`	Aborta a solicitação atual.
`open("método", "url")`	Configura o estágio de uma chamada ao servidor. O argumento de método pode ser GET, POST ou PUT. Possui ainda três parâmetros opcionais: um indicando se a requisição deve ser assíncrona ou não e outros indicando o nome de usuário e a senha específicos.
`send(content)`	Envia a solicitação ao servidor. Se a solicitação for declarada como assíncrona (no método `open`), ele retornará imediatamente, caso contrário aguardará até receber a resposta do servidor.
`setRequestHeader("header","value)`	Configura o cabeçalho especificado com o valor fornecido. Deve ser chamado após uma chamada a `open()`.

Fonte: adaptado de Asleson e Schutta (2006, p. 25).

Quadro 2 – Métodos do objeto `XMLHttpRequest`

Da mesma forma, as principais propriedades do objeto `XMLHttpRequest` são descritas a seguir, no quadro 3.

Propriedade	Descrição
`onreadystatechange`	Manipulador de eventos acionado a cada mudança de estado
`readyState`	O estado da solicitação. Tem cinco valores possíveis, a saber: 0 – não-inicializada; 1 – carregando; 2 – carregada; 3 – interativa; e 4 – concluída.
`responseText`	A resposta do servidor na forma de uma string
`responseXML`	A resposta do servidor no formato XML.
`status`	O código de status HTTP do servidor, ou seja, 200 para OK, 404 para Página Não Encontrada e assim por diante.
statusText	A versão em texto do código de status HTTP.

Fonte: adaptado de Asleson e Schutta (2006, p. 26).

Quadro 3 – Propriedades do objeto `XMLHttpRequest`

O funcionamento de uma interação assíncrona entre o navegador e o servidor será exemplificado a seguir. Supondo que tenhamos um formulário com um campo onde o usuário informa seu endereço de e-mail, como no quadro 4.

```
<input type=”text” id=”mail” name=”mail” onchange=”valEmail()”;>
```

Quadro 4 – exemplo de campo em formulário

Ao alterar o valor deste campo, será disparada a função JavaScript denominada `valEmail()`, a qual criará um objeto `XMLHttpRequest` e enviará a solicitação ao servidor. O código dessa função pode ser parecido com o que está no quadro 5.

```
var xmlHttp;

function valEmail(){
 var email = document.getElementById(“mail”); // campo e-mail
 var url = “validar?email=” + escape(email.value); //servlet e parâmetro
 //cria o objeto XMLHttpRequest
 if (window.ActiveXObject) { // indica que o navegador é o IE
   xmlHttp = new ActiveXObject(“Microsoft.XMLHTTP”);
 }
 else if (window.XMLHttpRequest){
    xmlHttp = new XMLHttpRequest();
 }
 xmlHttp.open(“GET”, url); //inicia a solicitação
 xmlHttp.onreadystatechange = trataEstado; //método p/ tratar o retorno
 xmlHttp.send(null); //envia a solicitação ao servidor
}
```

Quadro 5 – função exemplo `valEmail()`

A solicitação feita no navegador dispara, por exemplo, a execução de um *servlet* no servidor, sendo que assim que a mesma for processada, será enviado um retorno ao navegador, retorno este que pode ser um arquivo-texto ou um XML. Neste exemplo, este retorno será tratado pela função `trataEstado()`, que foi informada na propriedade

`onreadystatechange` do objeto `XMLHttpRequest`. Esta função poderia ter o código semelhante ao exposto no quadro 6.

```
function trataEstado(){
  if (xmlHttp.readyState == 4) { //servidor recebeu a solicitação
    if (xmlHttp.status == 200){ //solicitação atendida sem problemas
          //faz algo interessante aqui
    }
  }
}
```

Quadro 6 – função exemplo `trataEstado()`

Com a execução da função `trataEstado()`, se encerra a interação entre o navegador e o servidor.

O detalhe mais importante dessa interação é que, ao contrário dos aplicativos web comuns, após a solicitação ser feita ao servidor não ocorre a recarga total da página, o que incrementa em muito a usabilidade do sistema, sendo este um dos benefícios da utilização do Ajax para obtenção de uma interface rica para as aplicações web.

2.7 TRABALHOS CORRELATOS

Para a elaboração deste trabalho, o universo de pesquisa de trabalhos correlatos utilizado foi o de Trabalhos de Conclusão de Curso (TCC) da Universidade Regional de Blumenau (FURB), onde foram encontradas duas obras.

Bernardi (1997) desenvolveu uma ferramenta que, assim como o trabalho ora apresentado, possibilitava a modelagem de um banco de dados relacional, cadastrando as tabelas com suas colunas e relacionamentos. Esta ferramenta foi desenvolvida em Delphi, sendo um aplicativo *desktop* cujo principal objetivo era, além do cadastro da estrutura do banco, a geração de um arquivo-texto (*script*) com os comandos na linguagem *Structured Query Language* (SQL) necessários para criar o banco em um SGBD específico, o Interbase.

Já Gois (2006) utilizou a especificação JEE juntamente com as técnicas Ajax para desenvolver uma aplicação web para monitoramento de ambientes, na qual foram implementados recursos típicos de aplicações *desktop* (menus sensíveis ao contexto, *drag-and-drop*, etc.) e também requisições ao servidor feitas de forma assíncrona, assim como ocorre no presente trabalho.

3 DESENVOLVIMENTO DO APLICATIVO

O presente capítulo descreve o processo de desenvolvimento do aplicativo, apresentando a análise dos requisitos, a especificação do sistema e a sua implementação. Além disso, mostra como o Ajax e padrões de projeto foram utilizados em conjunto com a especificação JEE.

As etapas do desenvolvimento do software foram:

a) definição dos requisitos: as principais funcionalidades necessárias ao sistema foram identificadas e documentadas;
b) especificação do sistema: foram criados os diagramas da *Unified Modeling Language* (UML) pertinentes ao sistema, além da definição do modelo do banco de dados da aplicação;
c) implementação: com os requisitos definidos e com o término da especificação do sistema, ocorreu o desenvolvimento do mesmo, com testes sendo executados conjuntamente.

3.1 REQUISITOS

Os softwares de modelagem de banco de dados já existentes no mercado, como o PowerDesigner da Sybase, por exemplo, serviram como fonte para obtenção de alguns requisitos funcionais. A estes, foram adicionados outros, como o controle de versões de projetos e a restrição de acesso dos usuários aos projetos. O quadro 7 apresenta os requisitos funcionais do sistema.

REQUISITOS FUNCIONAIS
RF001: O sistema deverá permitir o cadastramento dos usuários que poderão acessá-lo.
RF002: O sistema deverá possibilitar o cadastramento dos projetos controlados.
RF003: O sistema deverá possibilitar a ligação entre os projetos e os usuários que terão permissão de acesso aos mesmos.
RF004: O sistema deverá permitir o controle das versões de cada projeto.
RF005: O sistema deverá permitir o cadastramento das tabelas existentes em cada versão, assim como suas colunas e relacionamentos. Para as colunas, deverá ser possível informar um tipo de dado, bem

como definir se o mesmo é parte da chave primária da tabela ou não. Deverá ser possível efetuar este cadastramento graficamente, na forma de diagramas.
RF006: O sistema deverá possibilitar a geração de arquivos-texto com os comandos SQL para a criação do banco de dados nos SGBD Oracle e MSSqlServer.
RF007: O sistema deverá permitir a geração de arquivos-texto com os comandos SQL para a atualização do banco de dados nos SGBD Oracle e MSSqlServer, baseado na comparação das diferenças entre duas versões do projeto.
RF008: O sistema deverá emitir um relatório com a descrição das diferenças existentes entre uma versão e outra do projeto, identificando tabelas e colunas que tenham sido incluídas, alteradas ou excluídas.

Quadro 7 – Requisitos funcionais do sistema

O quadro 8 demonstra os requisitos não funcionais.

REQUISITOS NÃO FUNCIONAIS
RNF001: O sistema deverá ser compatível com a versão 6.0 ou superior do navegador Internet Explorer da Microsoft.
RNF002: O sistema deverá ser desenvolvido utilizando o SGBD MySql.
RNF003: O sistema deverá ser desenvolvido na linguagem Java para o ambiente web (JEE).
RNF004: O sistema deverá tratar dois tipos de usuários: administradores (controle total) e analistas de sistemas.
RNF005: O acesso ao sistema deverá ser permitido somente mediante informação de um código e uma senha de acesso.
RNF006: A senha de acesso deverá ser armazenada de forma criptografada no banco de dados.
RNF007: Os arquivos-texto gerados pelo sistema, com comandos SQL para criação e atualização do banco de dados, deverão ser gerados em conformidade com as particularidades de cada SGBD no que tange aos tipos de dados e outros comandos cuja sintaxe possa ser diferenciada entre eles.
RNF008: O sistema deverá, quando possível, utilizar Ajax para efetuar a troca de informações entre o cliente (navegador) e o servidor, fazendo com que não seja necessário atualizar a página inteira ao fazê-lo.
RNF009: O sistema deverá ser desenvolvido de acordo com a arquitetura MVC, separando claramente a interface das regras de negócio.

Quadro 8 – Requisitos não funcionais do sistema

3.2 ESPECIFICAÇÃO

Após a definição dos requisitos, foi desenvolvida a especificação do software, através da geração dos diagramas da UML com a ferramenta Enterprise Architect (EA) e do diagrama do modelo da base de dados com a ferramenta PowerDesigner.

Os diagramas desenvolvidos são apresentados nas seções a seguir.

3.2.1 Diagrama de casos de uso

De acordo com o requisito não funcional RNF004, o sistema deve tratar dois tipos de usuário: administradores e analistas. Dessa forma, foram desenvolvidos dois diagramas de caso de uso: um com os casos de uso onde o único ator é o administrador e outro com os casos de uso comuns aos dois atores.

Os casos de uso exclusivos do administrador aparecem na figura 6.

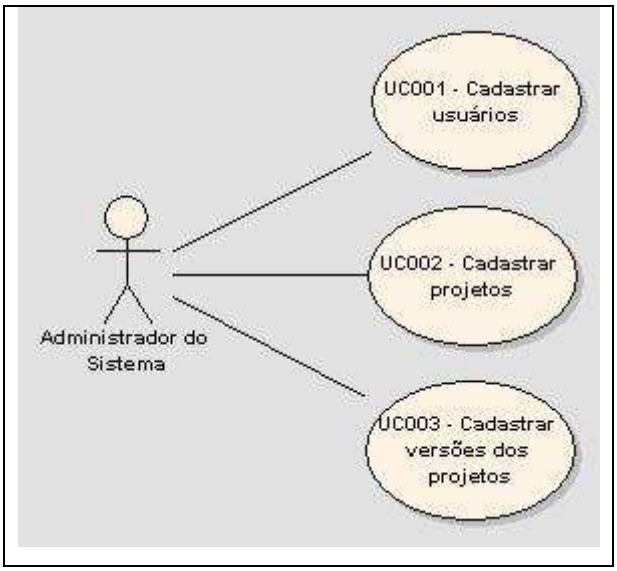


Figura 6 – Diagrama dos casos de uso exclusivos do administrador

A figura 7 ilustra os casos de uso comuns ao administrador e ao analista de sistemas.

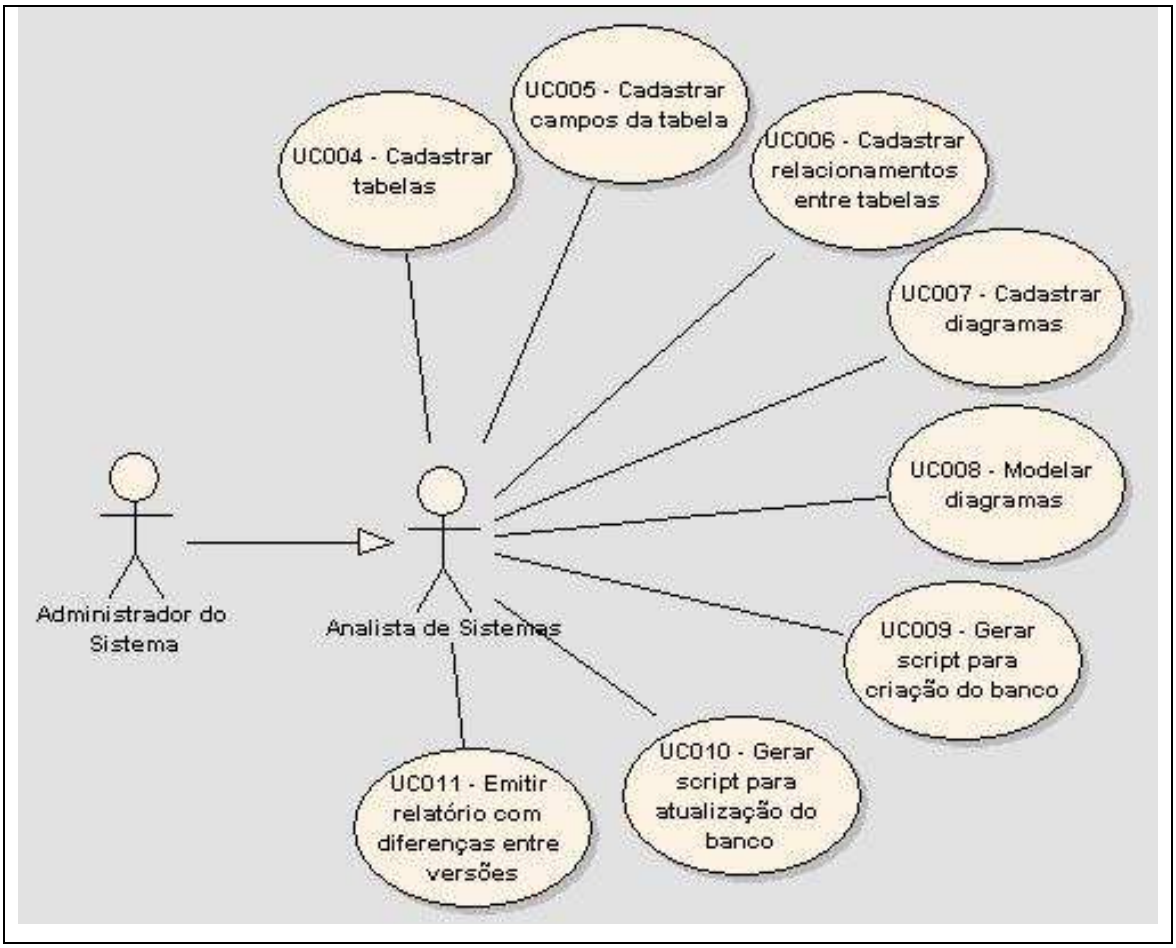


Figura 7 – Diagrama de casos de uso comuns ao administrador e ao analista

O quadro 9, a seguir, ilustra a matriz de relacionamento entre os requisitos funcionais e os casos de uso, cujo objetivo é garantir que todos os requisitos estejam sendo tratados por algum caso de uso, possibilitando também o rastreamento da relação entre eles.

	UC 001	UC 002	UC 003	UC 004	UC 005	UC 006	UC 007	UC 008	UC 009	UC 010	UC 011
RF001	■										
RF002		■									
RF003	■	■									
RF004			■								
RF005				■	■	■	■	■			
RF006									■		
RF007										■	
RF008											■

Quadro 9 – Matriz de relacionamento entre os casos de uso e os requisitos funcionais

3.2.2 Diagramas de atividades

Nesta seção são listados os diagramas de atividades que demonstram visualmente o funcionamento dos casos de uso do sistema, no que tange à interação do usuário com o aplicativo.

A Figura 8 tem o diagrama de atividades relacionado ao caso de uso UC001 – Cadastrar usuários.

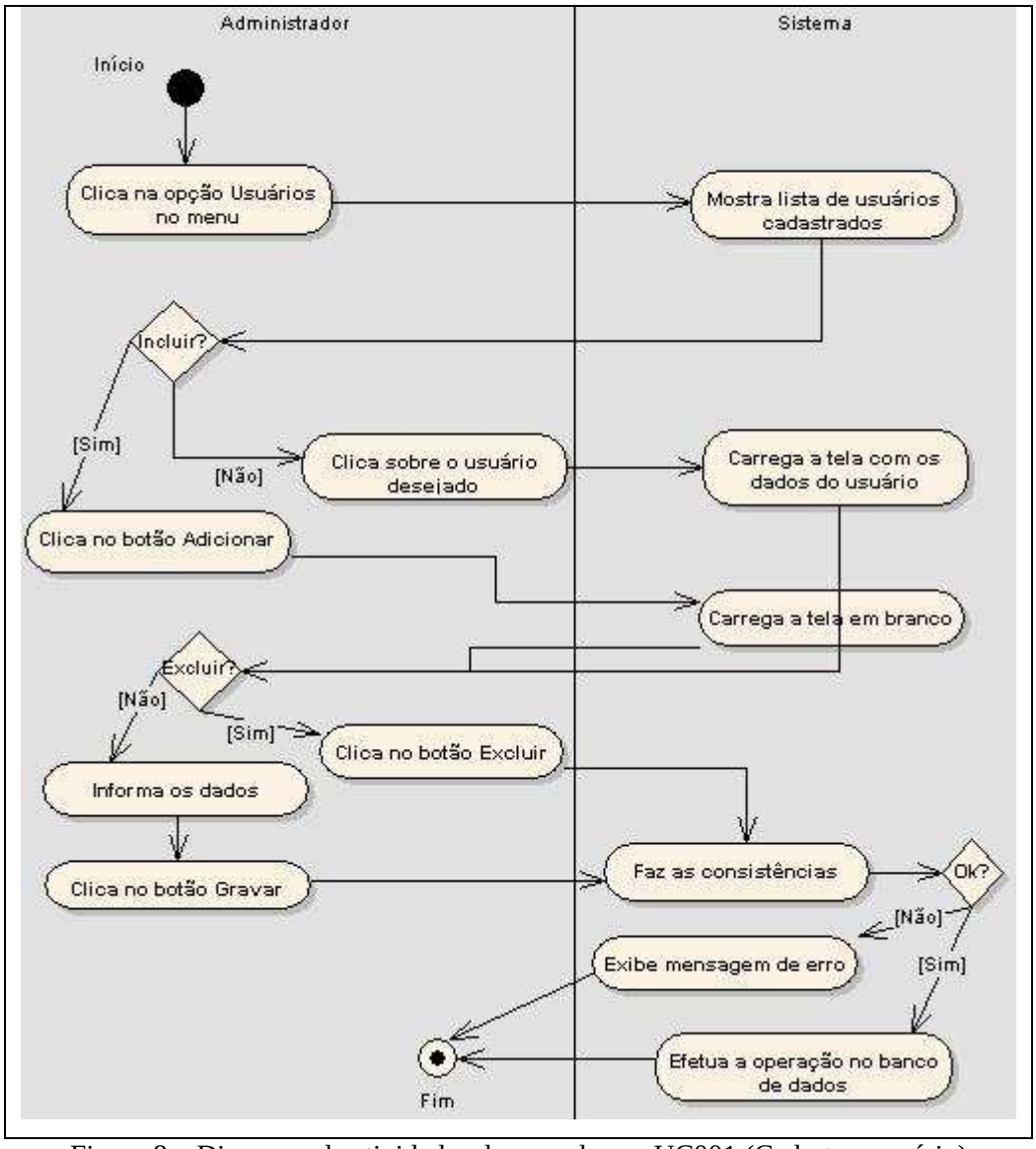


Figura 8 – Diagrama de atividades do caso de uso UC001 (Cadastrar usuário)

A figura 9 refere-se ao caso de uso UC002 – Cadastrar projetos.

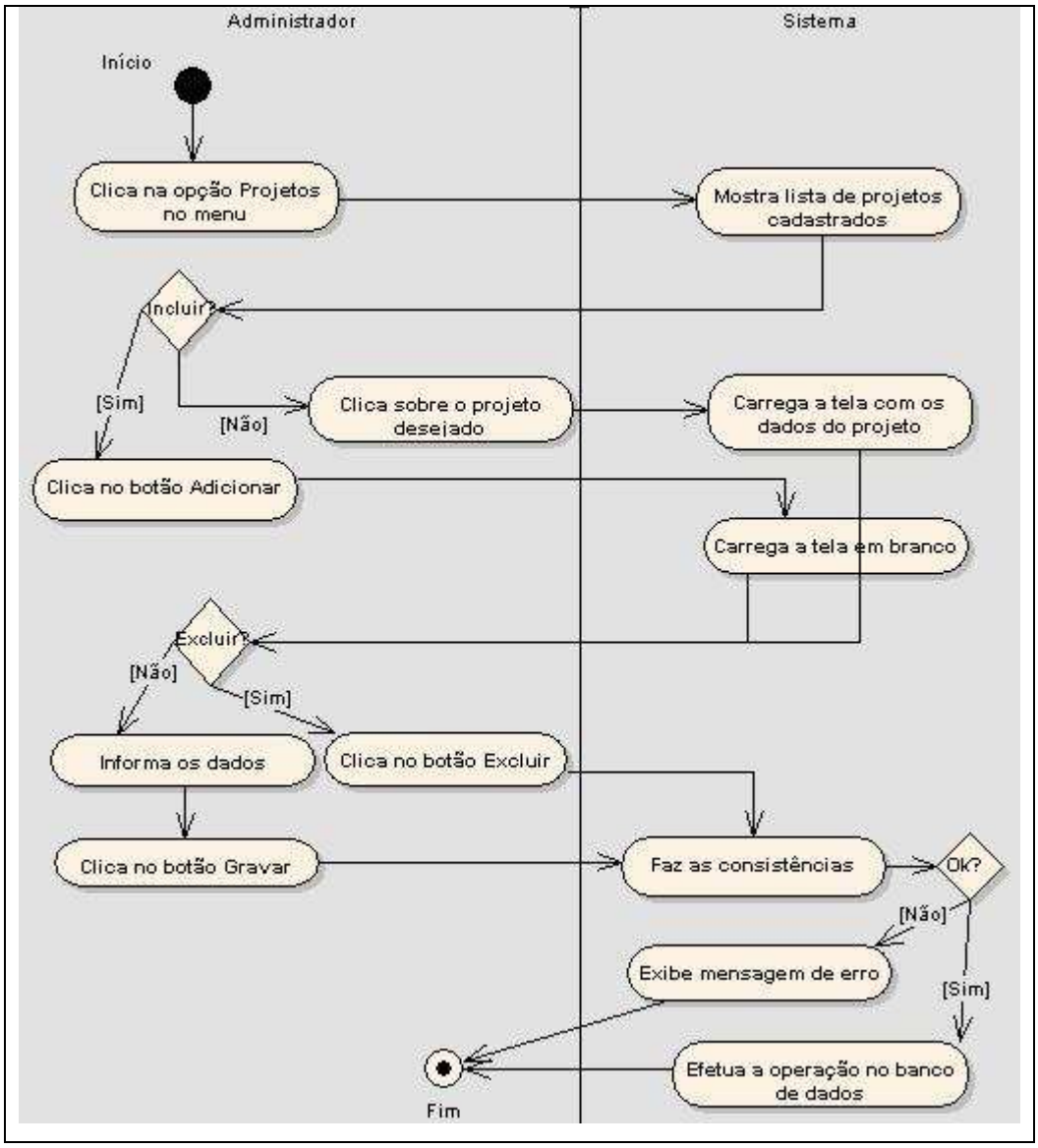


Figura 9 – Diagrama de atividades do caso de uso UC002 (Cadastrar projeto)

O último caso de uso exclusivo do ator Administrador (UC003 – Cadastrar versões de projetos) é detalhado na figura 10.

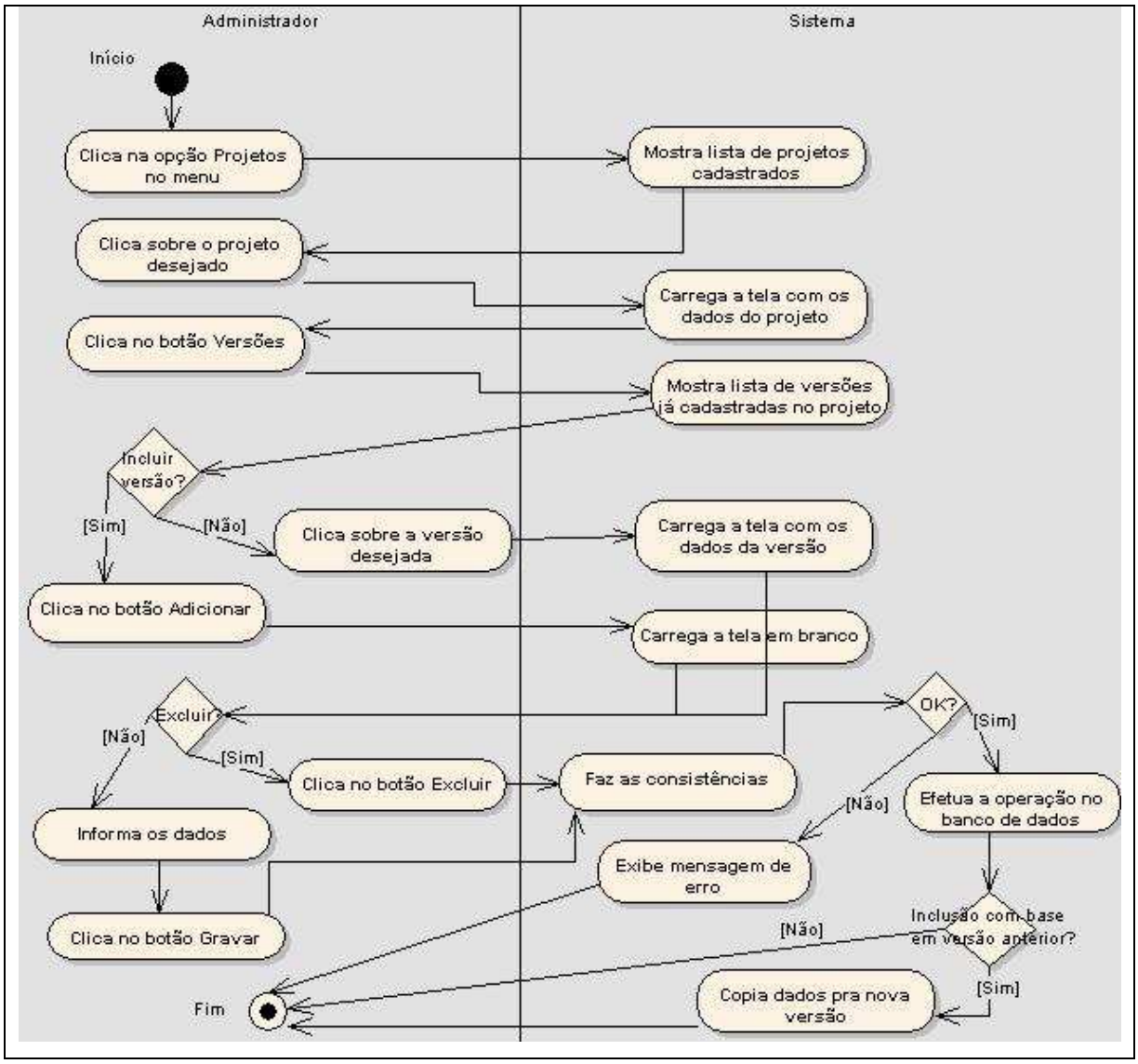


Figura 10 – Diagrama de atividades do UC003 (Cadastrar versões de projetos)

Os diagramas de atividades das figuras 11, 12, 13, 14, 15, 16, 17 e 18 descrevem, respectivamente, os casos de uso UC004, UC005, UC006, UC007, UC008, UC009, UC010 e UC011.

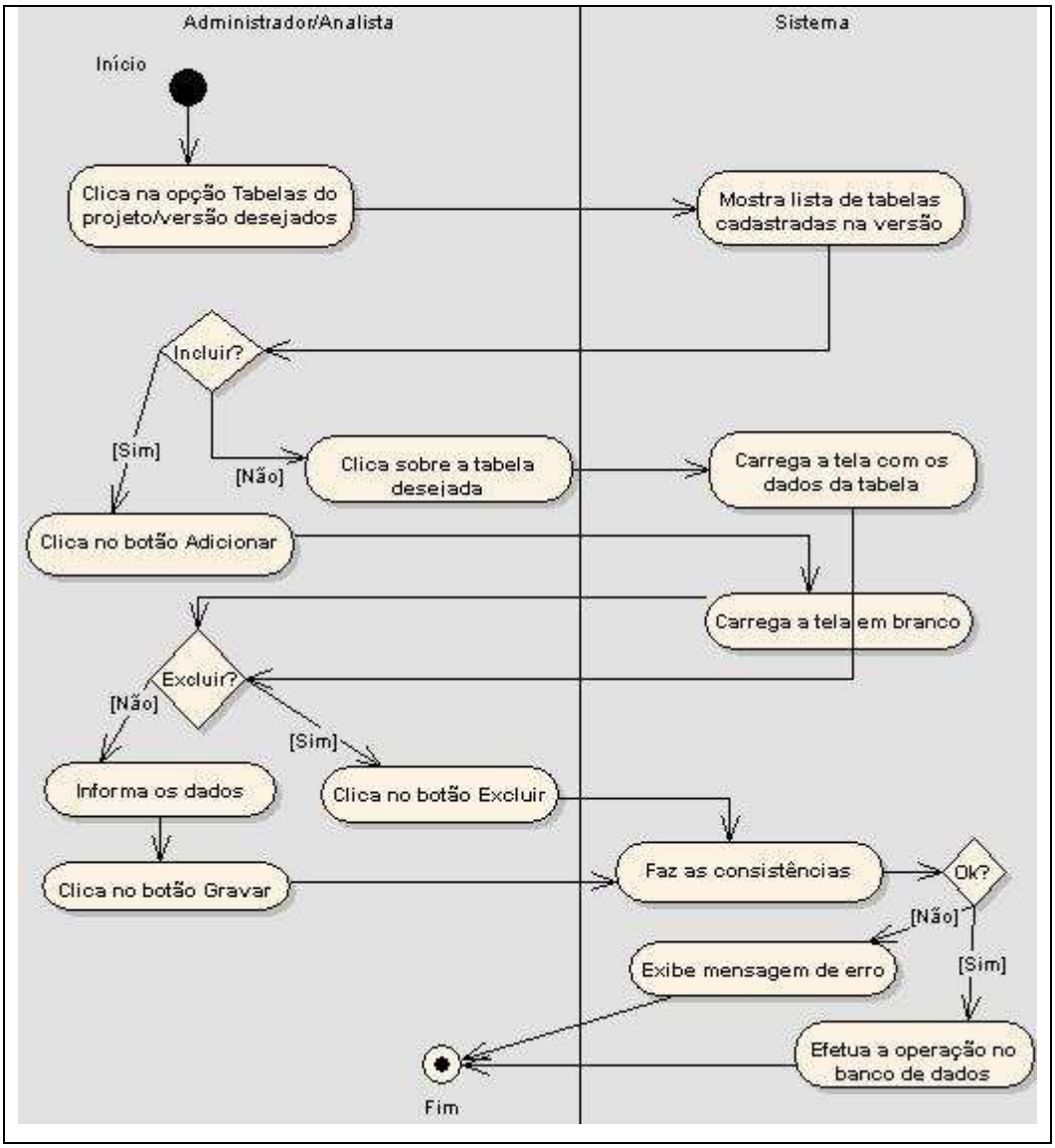


Figura 11 – Diagrama de atividades do UC004 (Cadastrar Tabelas)

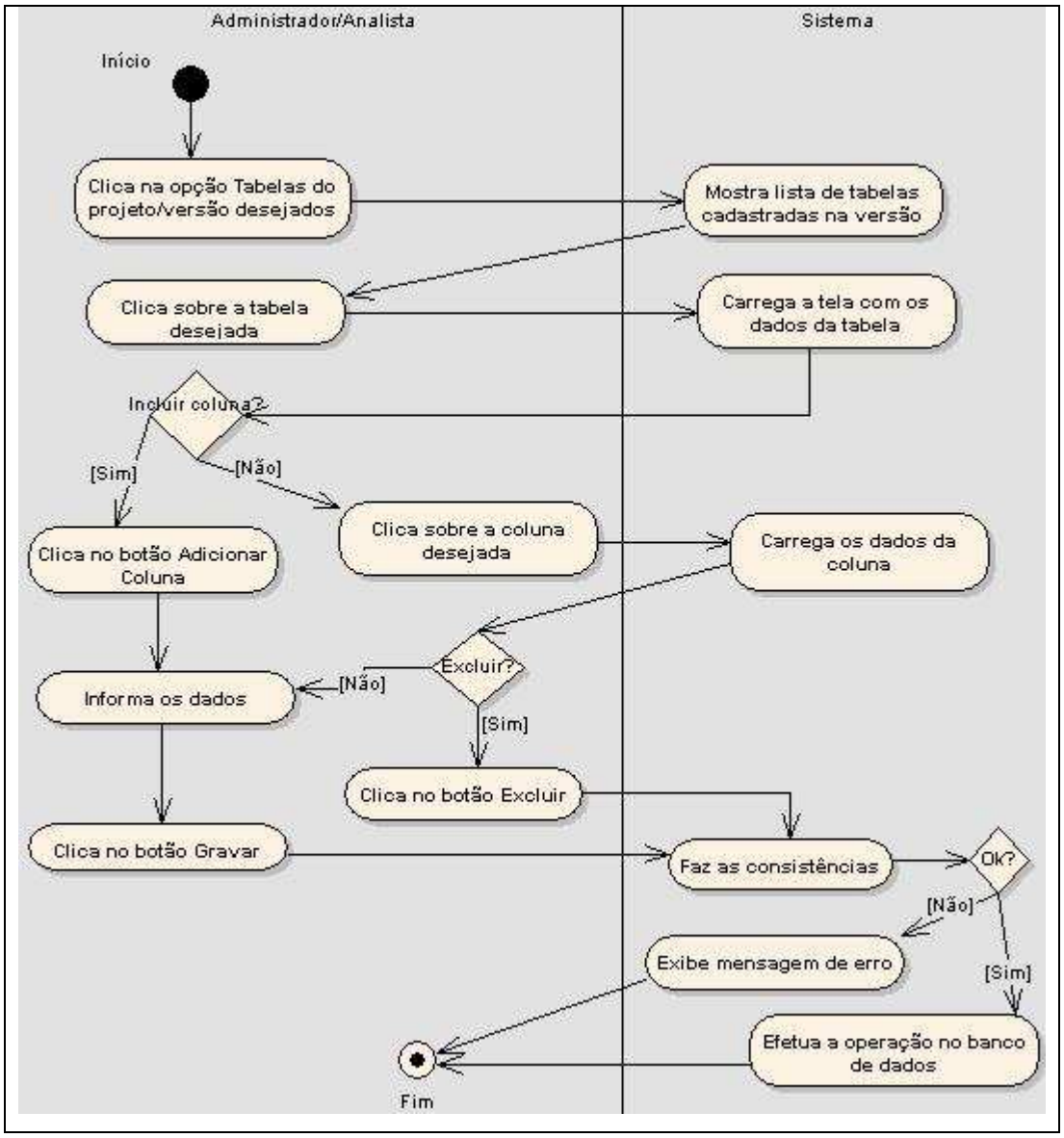


Figura 12 – Diagrama de atividades do UC005 (Cadastrar Campos das Tabelas)

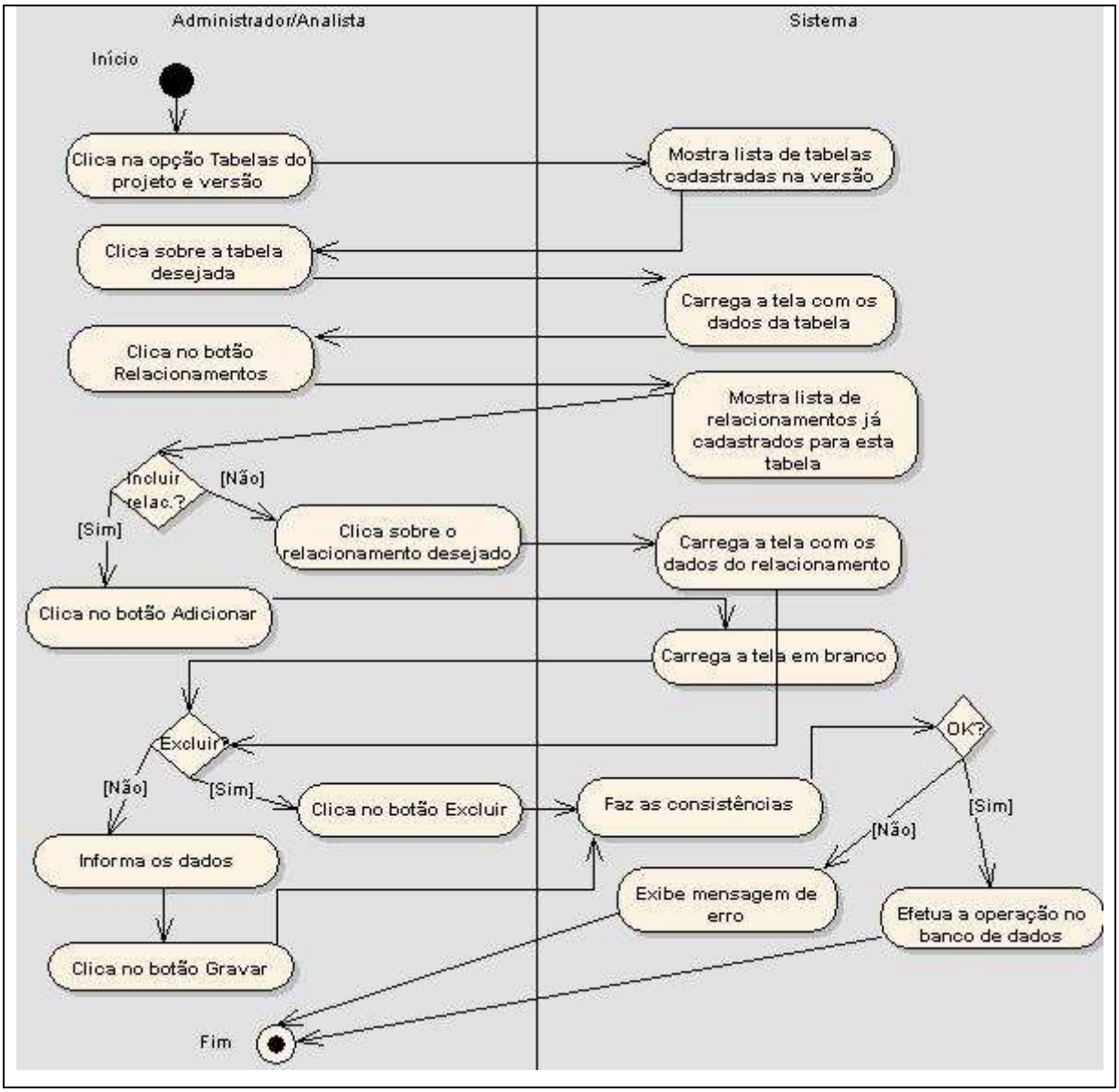


Figura 13 – Diagrama de atividades do UC006 (Cadastrar Relacionamentos)

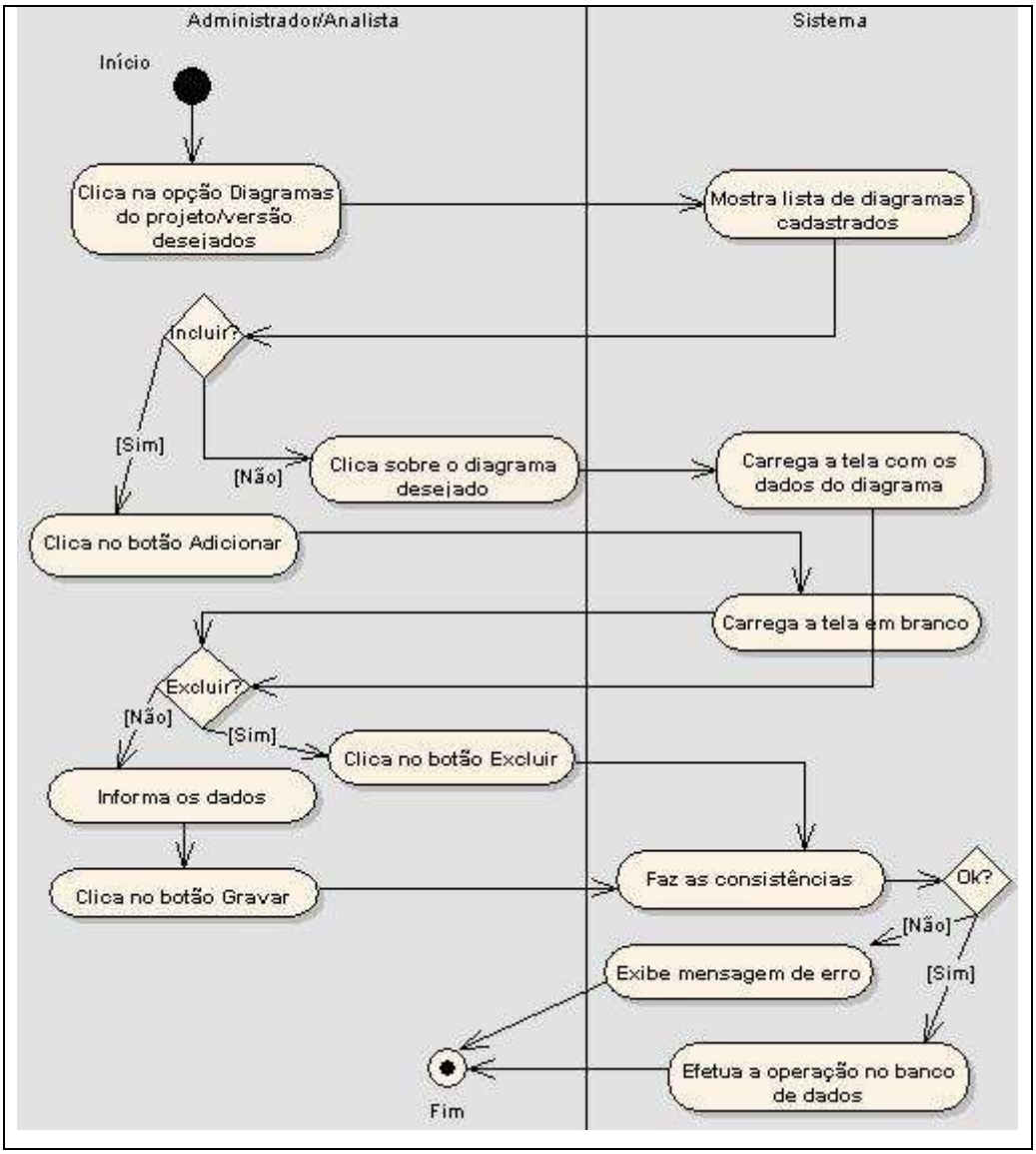


Figura 14 – Diagrama de atividades do UC007 (Cadastrar Diagramas)

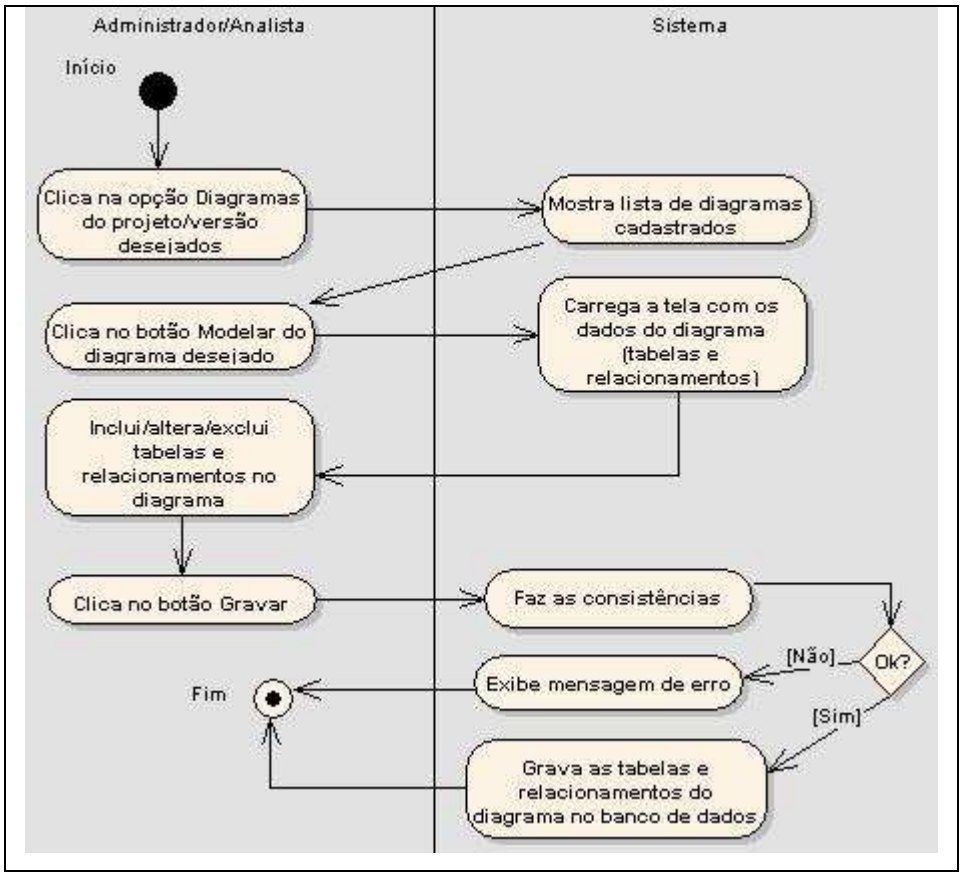


Figura 15 – Diagrama de atividades do UC008 (Modelar Diagramas)

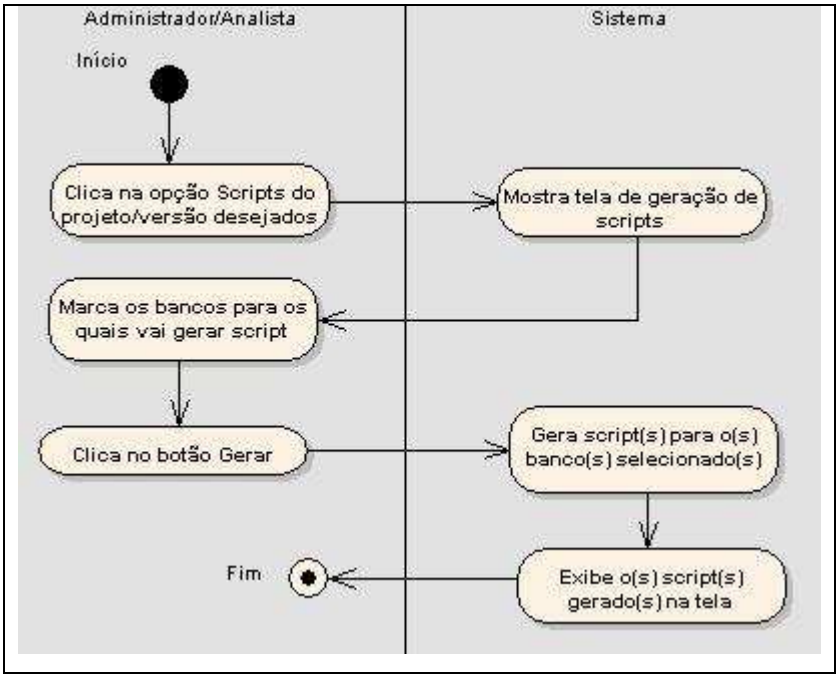


Figura 16 – Diagrama de atividades do UC009 (Gerar script de criação)

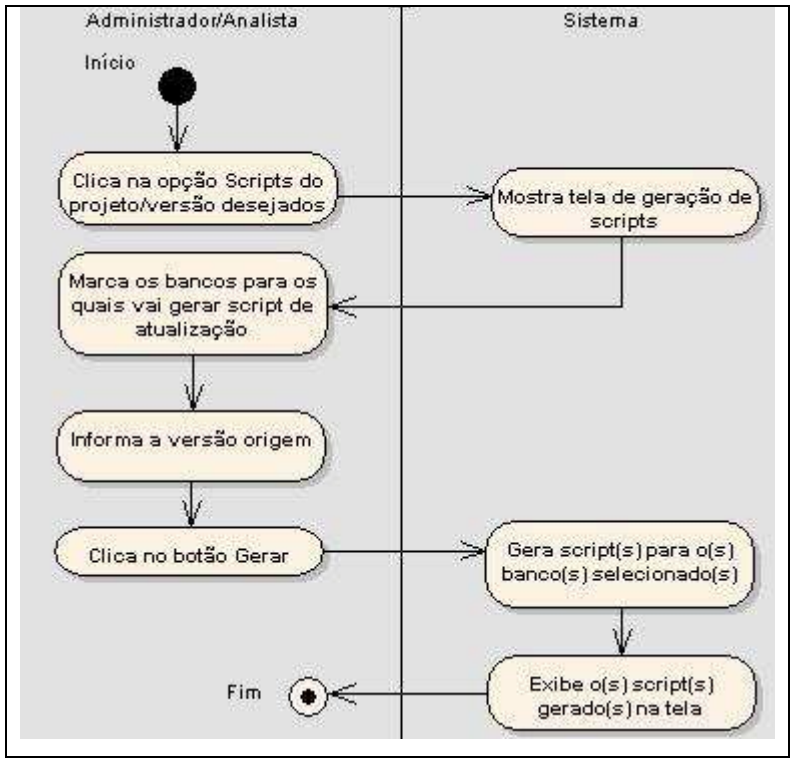


Figura 17 – Diagrama de atividades do UC010 (Gerar script de atualização)

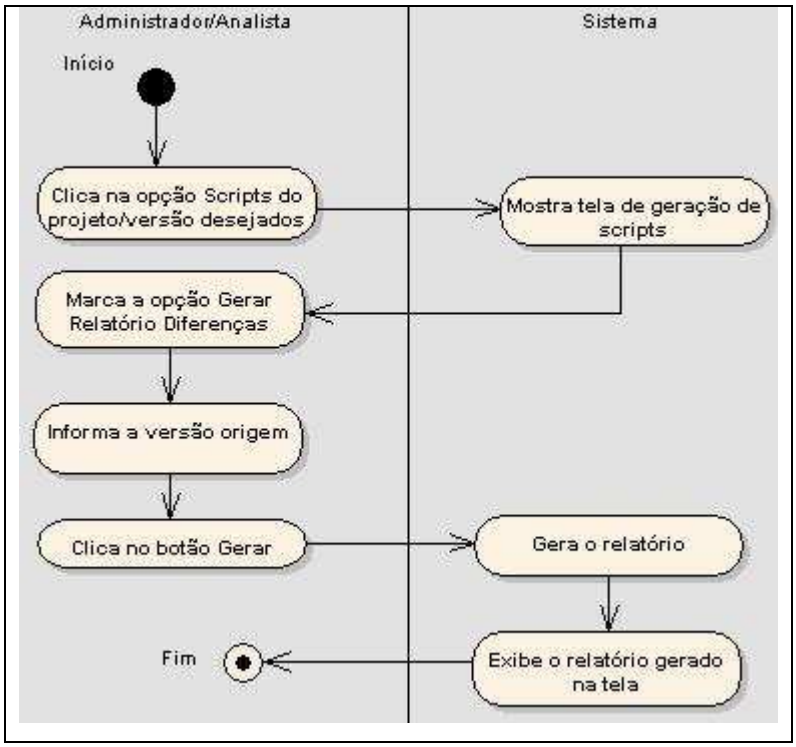


Figura 18 – Diagrama de atividades do UC011 (Gerar relatório de diferenças)

3.2.3 Diagramas de pacotes e de classes

O sistema foi desenvolvido utilizando o conceito de orientação a objetos e, portanto,

foram desenvolvidas inúmeras classes, as quais foram agrupadas em pacotes para facilitar a organização e manutenção do código-fonte.

A figura 19 ilustra os principais pacotes do sistema, organizados de forma a demonstrar sua posição dentro da arquitetura MVC.

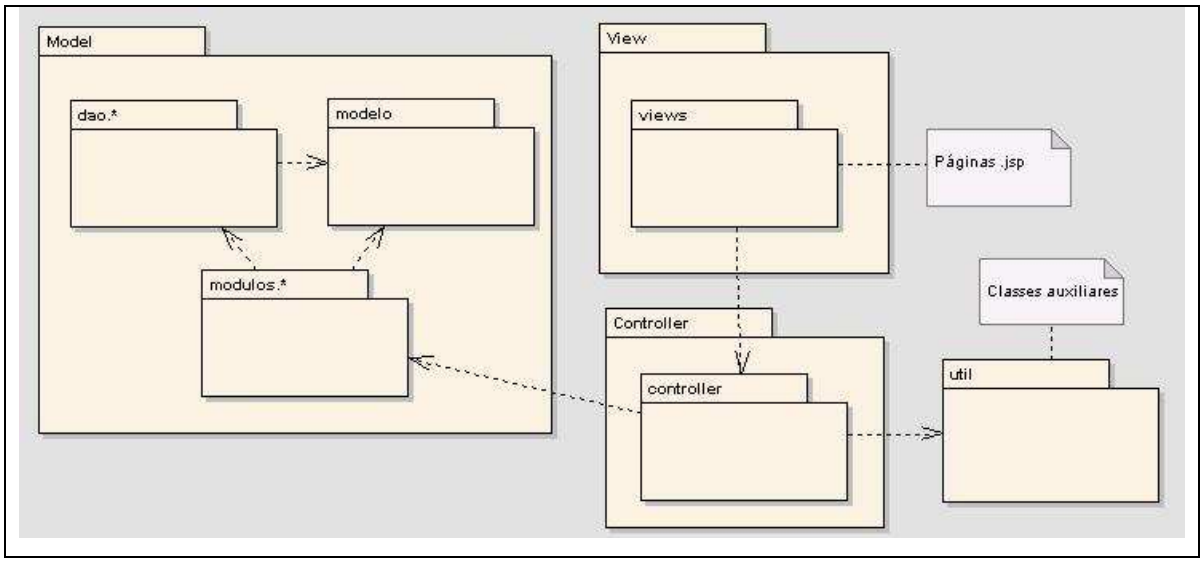


Figura 19 – Diagrama de pacotes do sistema na arquitetura MVC

A figura 20 detalha o pacote `dao.*` ilustrado na figura anterior.

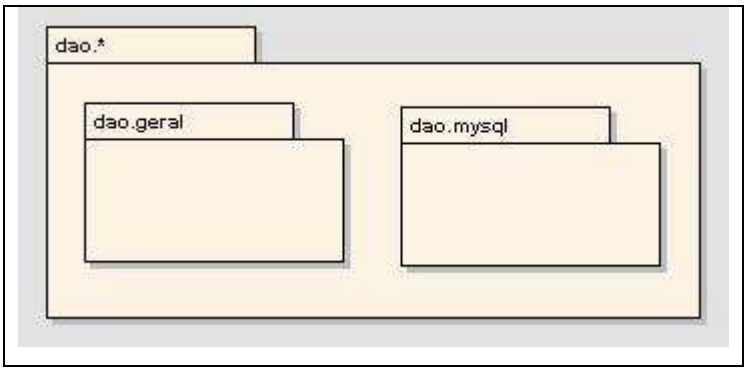


Figura 20 – Diagrama de pacotes detalhando o pacote `dao.*`

Por fim, a figura 21 detalha o pacote `modulos.*` ilustrado na figura 19.

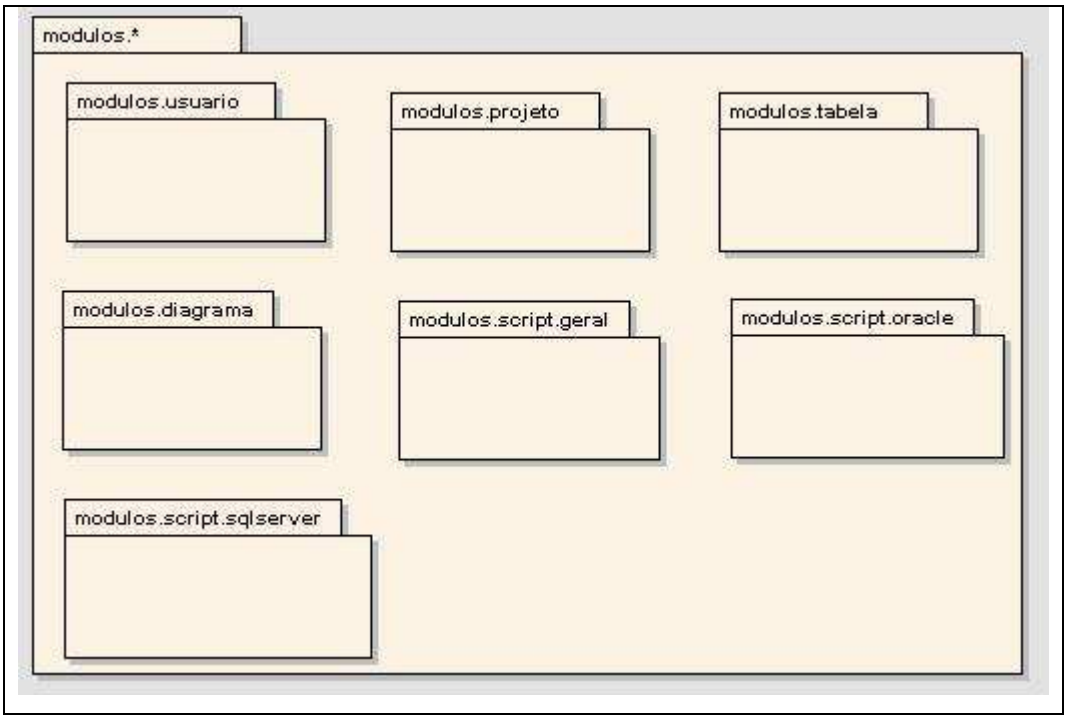


Figura 21 – Diagrama de pacotes detalhando o pacote `modulos.*`

O quadro 10 descreve a função de cada pacote dentro do sistema.

Pacote	Descrição
`views`	Pacote onde estão as páginas .jsp, ou seja, a interface do sistema.
`controller`	Pacote com as classes onde está implementada a camada de Controle da aplicação, ou seja, as classes que ligam o Modelo à Visão.
`util`	Classes diversas utilizadas para tratamento de exceções, criptografia, etc.
`modelo`	Neste pacote estão as classes do domínio da aplicação.
`dao.geral`	Classes abstratas para implementação do padrão DAO, responsáveis pelo acesso ao banco de dados.
`dao.mysql`	Contêm as subclasses DAO para acesso ao banco de dados MySql.
`modulos.usuario`	Possui as classes responsáveis pelas rotinas relacionadas ao cadastramento de usuários do sistema.
`modulos.projeto`	Neste pacote estão as classes responsáveis pelas rotinas relacionadas ao cadastramento e controle dos projetos.
`modulos.tabela`	Classes responsáveis pelas rotinas relacionadas ao cadastro de tabelas, colunas e relacionamentos.
`modulos.diagrama`	Pacote onde estão as classes relacionadas ao cadastro e modelagem dos diagramas.
`modulos.script.geral`	Neste pacote estão as classes genéricas utilizadas nas rotinas de geração dos *scripts* de criação e atualização de bancos de dados. Além disso, possui a

	classe que gera o relatório de diferenças entre as versões.
`modulos.script.oracle`	Classes específicas para geração dos *scripts* de atualização e criação do banco de dados para Oracle.
`modulos.script.sqlserver`	Classes específicas para geração dos *scripts* de atualização e criação do banco de dados para MSSqlServer.

Quadro 10 – Funções dos pacotes no sistema

O quadro seguinte, 11, traz as classes do pacote `controller`.

Classe	Descrição
`FrontController`	Classe que implementa o padrão de mesmo nome, representando a camada de Controle da arquitetura MVC. Analisa as solicitações recebidas (endereços enviadas pelo *browser*), instancia e executa os `Commands` responsáveis por atender as solicitações.

Quadro 11 – Classes do pacote `controller`

No pacote `util` encontram-se as classes descritas no quadro 12.

Classe	Descrição
`Command`	Classe básica para implementação do padrão de projeto de mesmo nome. É estendida nos pacotes dos módulos.
`Criptografia`	Classe utilizada para criptografar a senha dos usuários do sistema.
`RequestHelper`	Classe abstrata com a interface padrão para controle dos atributos que deverão ser passados como parâmetros para a execução dos `Commands`.
`HttpRequestHelper`	Implementação da classe anterior, voltada para aplicações web.
`ViewFlow`	Classe para armazenamento das relações entre as solicitações, os `Commands` que devem ser chamados para atendê-las e as páginas .jsp que devem ser exibidas após a execução do respectivo `Command`.
`WmException`	Classe de exceção estendida de `Exception`.

Quadro 12 – Classes do pacote `util`

As classes do pacote `modelo` são descritas no quadro 13.

Classe	Descrição
`ChavePrimaria`	Classe do domínio da aplicação, com os atributos referentes às chaves primárias das tabelas.
`Coluna`	Classe do domínio da aplicação, com os atributos referentes às colunas das tabelas.
`ColunaRelacionamento`	Classe do domínio da aplicação, com os atributos referentes às colunas pertencentes a um relacionamento.
`Diagrama`	Classe do domínio da aplicação, com os atributos referentes aos diagramas.
`Projeto`	Classe do domínio da aplicação, com os atributos referentes aos projetos.
`Relacionamento`	Classe do domínio da aplicação, com os atributos

	referentes aos relacionamentos entre tabelas.
`Tabela`	Classe do domínio da aplicação, com os atributos referentes às tabelas.
`TabelaDiagrama`	Classe do domínio da aplicação, com os atributos referentes às tabelas que integram os diagramas.
`Usuario`	Classe do domínio da aplicação, com os atributos referentes aos usuários do sistema.
`UsuarioProjeto`	Classe do domínio da aplicação, com os atributos referentes às relações entre usuários e projetos.
`Versao`	Classe do domínio da aplicação, com os atributos referentes às versões dos projetos.

Quadro 13 – Classes do pacote `modelo`.

O quadro 14 descreve as classes dos pacotes `dao.geral`, que são estendidas no pacote `dao.mysql`.

Classe	Descrição
`ColunaDAO`	Classe com métodos para gravação, exclusão e recuperação de dados referentes às colunas das tabelas.
`ColunaRelacionamentoDAO`	Classe com métodos para gravação, exclusão e recuperação de dados referentes às colunas pertencentes aos relacionamentos entre as tabelas.
`ComparaVersoesDAO`	Classe com métodos para recuperação de dados referentes às diferenças existentes entre duas versões de um projeto (tabelas, colunas e relacionamento incluídos, excluídos e/ou alterados).
`DBDAOFactory`	Classe com métodos de criação dos objetos DAO que serão utilizados pelo sistema. Classe-fábrica.
`DiagramaDAO`	Classe com métodos para gravação, exclusão e recuperação de dados referentes aos diagramas.
`ProjetoDAO`	Classe com métodos para gravação, exclusão e recuperação de dados referentes aos projetos.
`RelacionamentoDAO`	Classe com métodos para gravação, exclusão e recuperação de dados referentes aos relacionamentos entre as tabelas.
`TabelaDAO`	Classe com métodos para gravação, exclusão e recuperação de dados referentes às tabelas.
`UsuarioDAO`	Classe com métodos para gravação, exclusão e recuperação de dados referentes aos usuários.
`UsuarioProjetoDAO`	Classe com métodos para gravação, exclusão e recuperação de dados referentes aos relacionamentos entre usuários e projetos.
`VersãoDAO`	Classe com métodos para gravação, exclusão e recuperação de dados referentes às versões dos projetos.

Quadro 14 – Classes do pacote `dao.geral`, estendidas no pacote `dao.mysql`

Nessa etapa do desenvolvimento do trabalho, também foi criado o diagrama de classes relativo ao núcleo de processamento das requisições no sistema, representado na figura 22.

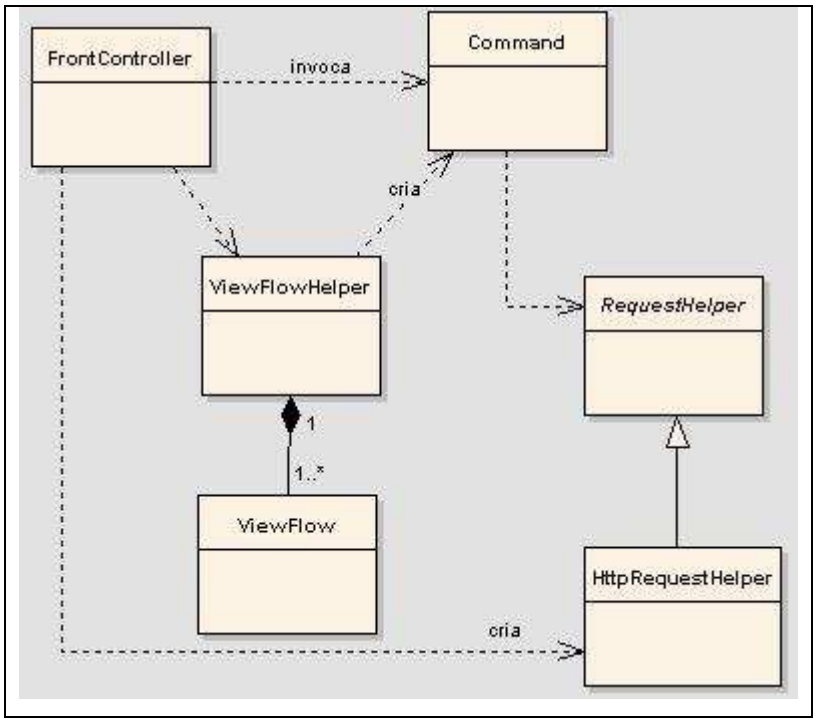


Figura 22 – Classes do núcleo de processamento de requisições do sistema

3.2.4 Modelo lógico do banco de dados do sistema

A modelagem do banco de dados do sistema foi feita utilizando o software PowerDesigner, da Sybase. A figura 23 ilustra o modelo do banco de dados.

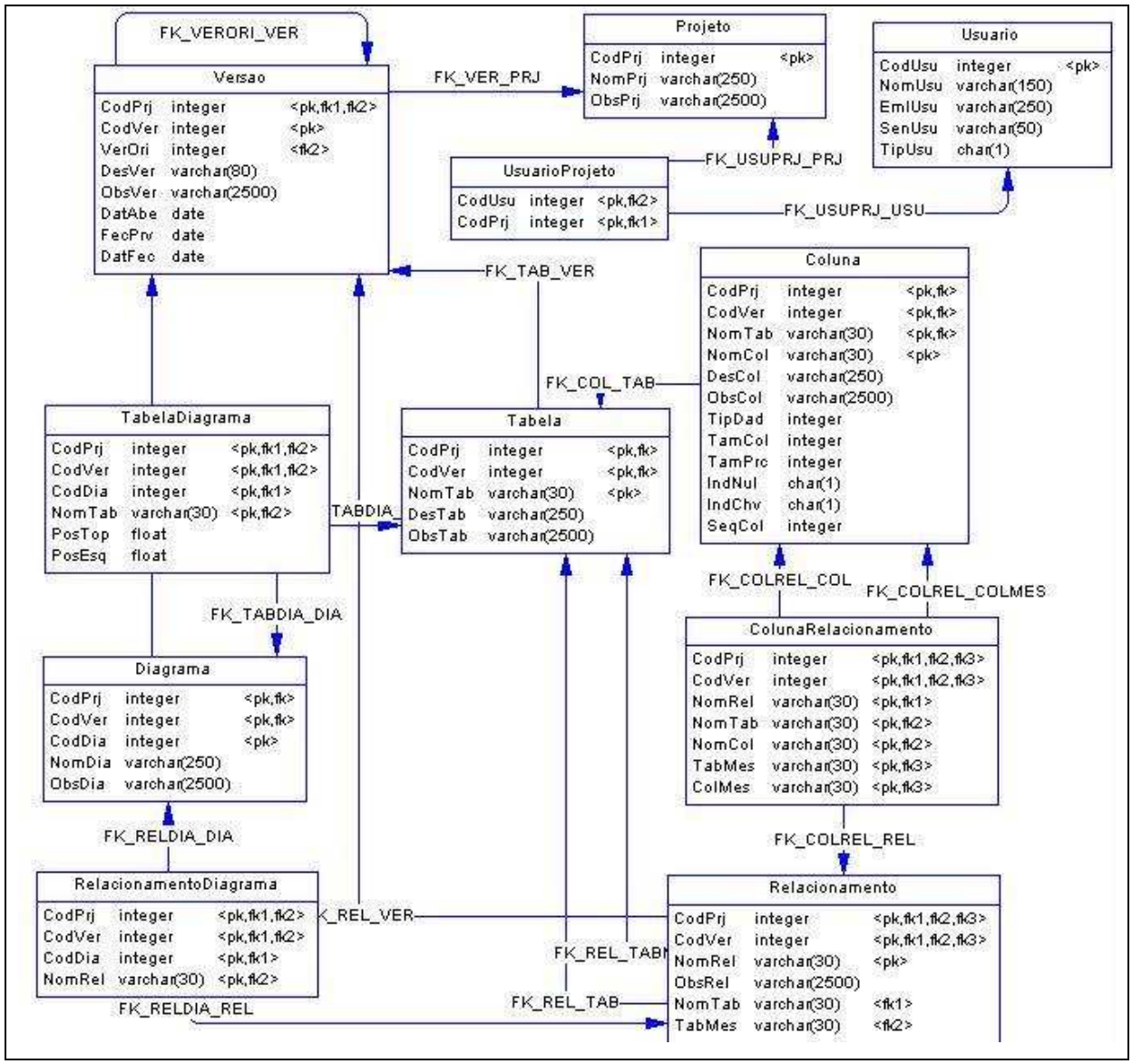


Figura 23 – Modelo do banco de dados do sistema

3.3 IMPLEMENTAÇÃO

Este tópico versa sobre a fase de implementação do software especificado no tópico anterior (batizado como WebModeler), descrevendo as técnicas e ferramentas utilizadas para tanto, além de conter um exemplo da operacionalidade da aplicação.

3.3.1 Técnicas e ferramentas utilizadas

O software foi desenvolvido na linguagem Java, de acordo com a especificação JEE, sendo que o ambiente de desenvolvimento utilizado foi o Eclipse, na versão 3.2.0. O Eclipse foi utilizado para desenvolver todas as camadas (MVC) do aplicativo, incluindo a interface (arquivos JSP) e seus códigos JavaScript. O servidor de aplicações utilizado para executar o software foi o Apache Tomcat (versão 5.5) e o SGBD utilizado foi o MySql.

3.3.2 Processo de implementação

A implementação do sistema ocorreu conforme os seguintes passos:

a) desenvolvimento da camada de persistência e recuperação dos dados;
b) desenvolvimento do núcleo de processamento de requisições do sistema;
c) implementação dos cadastros básicos do sistema;
d) implementação do módulo de modelagem gráfica;
e) implementação do módulo de geração de *scripts*;
f) definição dos estilos (cores, fontes) da interface da aplicação.

Cada um desses passos será detalhado a seguir.

3.3.2.1 Desenvolvimento da camada de persistência e recuperação dos dados

Nessa etapa, foram desenvolvidas as classes que implementam o padrão de projeto DAO, ou seja, as classes responsáveis por gravar os dados e recuperá-los no banco de dados.

Foram desenvolvidos dois pacotes de classes: o `dao.geral` e o `dao.mysql`. O primeiro contêm as super-classes com a definição dos seus métodos e atributos, enquanto o segundo consiste numa extensão do anterior, onde as classes foram adaptadas para funcionamento no banco de dados MySql.

Para cada tabela do banco de dados da aplicação, foi criada uma classe DAO correspondente, com os métodos pertinentes. O quadro 15 demonstra os principais métodos criados na classe `UsuarioDAO`.

Método	**Objetivo**
`carregar(int)`	Retorna um objeto do tipo `Usuario`, com os dados do usuário cujo código foi passado como parâmetro.
`excluir(int)`	Exclui do banco de dados o usuário cujo código foi passado como parâmetro.
`gravar(Usuario)`	Recebe um objeto do tipo `Usuario` como parâmetro e insere/altera os dados do mesmo no banco.
`listar()`	Retorna uma lista de objetos do tipo `Usuario` com todos os usuários cadastrados no sistema.

Quadro 15 – Principais métodos da classe `UsuarioDAO`

Conforme descrito anteriormente, a classe `UsuarioDAO` apenas define os métodos, os quais foram implementados na classe `MySqlUsuarioDAO`, do pacote `dao.mysql`, sendo que o mesmo aconteceu para as demais classes DAO criadas. O quadro 16 ilustra a implementação dos principais métodos da classe `MySqlUsuarioDAO`.

```
package dao.mysql;
[...]
public class MySqlUsuarioDAO extends UsuarioDAO {
[...]
	public Usuario carregar(int codigo) throws WmException {
		Usuario usu = null;
		PreparedStatement ps;
		try {
			ps   =   this.getConexao().prepareStatement("select   *   from
USUARIO where CODUSU = ?");
			ps.setInt(1, codigo);
			ResultSet rs = ps.executeQuery();
			while (rs.next()){
				usu = new Usuario();
				usu.setCodigo(rs.getInt("CODUSU"));
				usu.setNome(rs.getString("NOMUSU"));
				usu.setEmail(rs.getString("EMLUSU"));
				usu.setSenha(rs.getString("SENUSU"));
				usu.setTipo(rs.getString("TIPUSU"));
			}
			rs.close();
			ps.close();
		} catch (SQLException e) {
			throw new WmException(e.getMessage());
		}
		return usu;
	}
[...]
	public void excluir(int codigo) throws WmException {
		PreparedStatement ps;
		try {
			ps  =  this.getConexao().prepareStatement("delete  from  USUARIO
where CODUSU = ?");
			ps.setInt(1, codigo);
			ps.executeUpdate();
			ps.close();
		} catch (SQLException e) {
			throw new WmException(e.getMessage());
		}
```

```
	}
[...]
	public int gravar(Usuario usu, boolean atualizaSenha) throws WmException {
		int retorno = usu.getCodigo();
		PreparedStatement ps = null;
		if	(usu.getCodigo()  ==  0){  //código  igual  a  zero  significa
Inclusão
			try {
				ps  =  this.getConexao().prepareStatement("insert  into
USUARIO (CODUSU,NOMUSU,EMLUSU,SENUSU,TIPUSU) values (?,?,?,?,?)");
				usu.setCodigo(this.proximoCodigo());
				retorno = usu.getCodigo();
				ps.setInt(1, usu.getCodigo());
				ps.setString(2, usu.getNome());
				ps.setString(3, usu.getEmail());
				ps.setString(4, usu.getSenha());
				ps.setString(5, usu.getTipo());
			} catch (SQLException e) {
				throw new WmException(e.getMessage());
			}
		} else { //é alteração
			try {
				if	(atualizaSenha) {
					ps  =  this.getConexao().prepareStatement("update
USUARIO set NOMUSU = ?,EMLUSU = ?,SENUSU = ?,TIPUSU = ? where CODUSU = ?");
					ps.setString(1, usu.getNome());
					ps.setString(2, usu.getEmail());
					ps.setString(3, usu.getSenha());
					ps.setString(4, usu.getTipo());
					ps.setInt(5, usu.getCodigo());
				} else {
					ps  =  this.getConexao().prepareStatement("update
USUARIO set NOMUSU = ?,EMLUSU = ?,TIPUSU = ? where CODUSU = ?");
					ps.setString(1, usu.getNome());
					ps.setString(2, usu.getEmail());
					ps.setString(3, usu.getTipo());
					ps.setInt(4, usu.getCodigo());
				}

			} catch (SQLException e) {
				throw new WmException(e.getMessage());
			}
		}
		//executa o SQL no banco
		try {
			ps.executeUpdate();
			ps.close();
		} catch (SQLException e) {
			throw new WmException(e.getMessage());
		}
		return retorno;
	}
[...]
	public List<Usuario> listar() throws WmException {
		List<Usuario> usuarios = new ArrayList<Usuario>();
		PreparedStatement ps;
		try {
			ps = this.getConexao().prepareStatement("select * from
USUARIO");
			ResultSet rs = ps.executeQuery();
			while (rs.next()){
				Usuario usu = new Usuario();
				usu.setCodigo(rs.getInt("CODUSU"));
				usu.setNome(rs.getString("NOMUSU"));
				usu.setEmail(rs.getString("EMLUSU"));
				usu.setSenha(rs.getString("SENUSU"));
				usu.setTipo(rs.getString("TIPUSU"));
				usuarios.add(usu);
```

```
                    }
                    rs.close();
                    ps.close();
             } catch (SQLException e) {
                    throw new WmException(e.getMessage());
             }
             return usuarios;
      }
[...]
}
```

Quadro 16 – Implementação dos principais métodos na classe `MySqlUsuarioDAO`

Após criar todas as classes DAO necessárias à aplicação, foi desenvolvida uma classe cuja funcionalidade é servir como fábrica de objetos DAO. No pacote `dao.geral` ela foi criada com o nome `DBDAOFactory`, sendo estendida na classe `MySqlDAOFactory` do pacote `dao.mysql`. O objetivo da utilização desta "classe-fábrica" é permitir que em tempo de execução seja definido qual fábrica será usada pela aplicação e, consequentemente, quais classes DAO serão instanciadas. O quadro 17 contêm o código-fonte da classe `MySqlDAOFactory`.

```
package dao.mysql;
[...]
public class MySqlDAOFactory extends DBDAOFactory {

      public ProjetoDAO getProjetoDAO() {
             return new MySqlProjetoDAO(this.getConexao());
      }
      public UsuarioDAO getUsuarioDAO(){
             return new MySqlUsuarioDAO(this.getConexao());
      }
      public UsuarioProjetoDAO getUsuarioProjetoDAO(){
             return new MySqlUsuarioProjetoDAO(this.getConexao());
      }
      public VersaoDAO getVersaoDAO(){
             return new MySqlVersaoDAO(this.getConexao());
      }
      public TabelaDAO getTabelaDAO(){
             return new MySqlTabelaDAO(this.getConexao());
      }
      public ColunaDAO getColunaDAO(){
             return new MySqlColunaDAO(this.getConexao());
      }
      public RelacionamentoDAO getRelacionamentoDAO(){
             return new MySqlRelacionamentoDAO(this.getConexao());
      }
      public ColunaRelacionamentoDAO getColunaRelacionamentoDAO(){
             return new MySqlColunaRelacionamentoDAO(this.getConexao());
      }
      public DiagramaDAO getDiagramaDAO(){
             return new MySqlDiagramaDAO(this.getConexao());
      }
      public ComparaVersoesDAO getComparaVersoesDAO(){
             return new MySqlComparaVersoesDAO(this.getConexao());
      }
}
```

Quadro 17 – Código-fonte da classe `MySqlDAOFactory`

3.3.2.2 Desenvolvimento do núcleo de processamento de requisições do sistema

A parte do sistema aqui nomeada como “núcleo de processamento de requisições” nada mais é do que a implementação do padrão *Front Controller* em conjunto com o padrão *Command*, além de ser a camada *Controller* da arquitetura MVC.

Seguindo o que foi exposto no capítulo 2 acerca do padrão *Front Controller*, criou-se uma classe de mesmo nome, a qual consiste num *servlet* Java que, segundo o que foi definido no arquivo de configuração web.xml, será responsável por atender todas as requisições cuja *url* termine com .do, conforme pode ser visto no quadro 18.

```
<servlet>
      <servlet-name>FrontCtrl</servlet-name>
      <servlet-class>controller.FrontController</servlet-class>
</servlet>
<servlet-mapping>
      <servlet-name>FrontCtrl</servlet-name>
      <url-pattern>*.do</url-pattern>
</servlet-mapping>
```

Quadro 18 – Mapeamento da classe `FrontController` no arquivo web.xml

A classe `FrontController` possui como objetivos receber as requisições do cliente (*browser*), encaminhá-las para serem tratadas pelas respectivas classes e, em seguida, devolver para o *browser* a página que deve ser exibida após o processamento ter sido concluído.

Para o tratamento das requisições, foi empregado o padrão de projeto *Command*. Com isto, foi implementada uma super-classe com o mesmo nome deste padrão, a qual possui apenas um atributo do tipo `DBDAOFactory` (fábrica de objetos DAO) e um método denominado `executar()`. Esta classe foi estendida para implementar cada tipo de requisição existente em cada módulo do sistema, como por exemplo, cadastro dos usuários (`GravarUsuarioCommand`, `ExcluirUsuarioCommand`, etc.) e projetos (`GravarProjetoCommand`, `ListarProjetosCommand`, etc.). O método `executar()` é onde a lógica de cada `Command` foi escrita, ou seja, é neste método que ocorre o processamento das requisições e é onde se encontram as regras de negócio do software. A grande maioria das requisições enviadas ao `FrontController` vêm acompanhada de um ou mais parâmetros que precisam ser considerados no processamento dentro dos `Commands` e, por esse motivo, no método `executar()` da classe `Command` foi incluído um parâmetro que é um objeto da classe `RequestHelper`.

A classe abstrata `RequestHelper` foi criada para servir como uma lista que contém

todos os parâmetros que precisam ser passados para o `Command` para que este possa atender a requisição. Por exemplo, a requisição de gravação de um novo usuário vem acompanhada de inúmeros parâmetros, tais como o nome, e-mail e senha, os quais precisam ser passados para o `Command` para que este possa efetuar a gravação. Esta classe foi estendida, dando origem à `HttpRequestHelper`, a qual foi implementada para tratar os parâmetros que são passados na *url* de uma requisição do ambiente web, recuperando os valores dos parâmetros de um objeto do tipo `HttpServletRequest` (classe nativa JEE). O código-fonte da classe `HttpRequestHelper` pode ser visto no quadro 19.

```
package util;
[..]
public class HttpRequestHelper extends RequestHelper {
	private HashMap<String,String> parametros = new HashMap<String,String>();
	private HttpServletRequest request;
	public HttpRequestHelper(HttpServletRequest request) {
		this.request = request;
	}
	public String getParametro(String nomeParametro) {
		String retorno = request.getParameter(nomeParametro);
		if	((retorno == null) || (retorno.equals(""))){
			retorno = parametros.get(nomeParametro);
		}
		if	(retorno == null){
			retorno = "";
		}
		return retorno;
	}
	public void setParametro(String nomeParametro, Object valor){
		parametros.put(nomeParametro, valor.toString());
	}
	public String[] getParametroLista(String nomeParametro) {
		return request.getParameterValues(nomeParametro);
	}
	public void setAtributo(String atributo, Object valor) {
		request.setAttribute(atributo, valor);
	}
	public Object getAtributo(String atributo) {
		Object retorno = null;
		retorno = this.request.getAttribute(atributo);
		if (retorno == null) {
			retorno = request.getSession().getAttribute(atributo);
		}
		return retorno;
	}
	public void setAtributoSessao(String atributo, Object valor){
		request.getSession().setAttribute(atributo, valor);
	}
	public Object getAtributoSessao(String atributo){
		return request.getSession().getAttribute(atributo);
	}
	public void removeAtributoSessao(String atributo){
		request.getSession().removeAttribute(atributo);
	}
}
```

Quadro 19 – Código-fonte da classe `HttpRequestHelper`

Como exemplo, cita-se a requisição de exclusão de um usuário no banco de dados, a qual possui a seguinte *url*: “/usu_excluir.do?codigo=999”. Neste exemplo, ao receber a

solicitação vinda do *browser*, o `FrontController` cria o `Command` que irá processar a requisição (neste caso, o nome da classe é `ExcluirUsuarioCommand`) e dispara o seu método `executar()`, passando também para o `Command` um objeto `HttpRequestHelper` onde há o parâmetro de nome "codigo" com o valor 999.

Cada requisição possui uma classe `Command` responsável por seu processamento e, para saber qual `Command` deve ser executado, foram desenvolvidas outras duas classes para auxiliar a `FrontController`: `ViewFlow` e `ViewFlowHelper`. A primeira possui simplesmente três atributos: um com a *url* da requisição, outro com o nome da classe `Command` que atenderá esta solicitação e mais um com o nome da página (arquivo JSP) que deverá ser exibida no *browser* após a execução do `Command`. Já a classe `ViewFlowHelper` consiste numa lista de objetos do tipo `ViewFlow`, tendo um método que, a partir da *url* de uma solicitação, percorre a lista e retorna o objeto `Command` relativo a esta solicitação.

O `FrontController` possui como atributo um objeto `ViewFlowHelper` com o mapeamento de todas as requisições possíveis do sistema, sendo que este mapeamento foi feito de forma estática no método de inicialização do `FrontController`, mas poderia, por exemplo, ter sido feito de forma dinâmica, através da leitura de um arquivo XML. O quadro 20 mostra o trecho do código-fonte do `FrontController` onde é feito o mapeamento das solicitações relativas ao cadastro de projetos e versões.

```
[…]
public void init() throws ServletException {
 super.init();

 vfh.addComando("/prj_listar.do","modulos.projeto.ListarProjetosCommand","prj_lista.jsp");
 vfh.addComando("/prj_tela.do", "modulos.projeto.TelaProjetoCommand", "prj_form.jsp");
 vfh.addComando("/prj_gravar.do","modulos.projeto.GravarProjetoCommand", "prj_listar.do");
 vfh.addComando("/prj_excluir.do","modulos.projeto.ExcluirProjetoCommand","");
 vfh.addComando("/prj_carregar.do", "modulos.projeto.CarregaProjetoCommand", "");
 vfh.addComando("/vrs_tela.do", "modulos.projeto.TelaVersaoCommand", "vrs_form.jsp");
 vfh.addComando("/vrs_carregar.do", "modulos.projeto.CarregaVersaoCommand", "");
 vfh.addComando("/vrs_gravar.do", "modulos.projeto.GravarVersaoCommand", "prj_listar.do");
 vfh.addComando("/vrs_excluir.do","modulos.projeto.ExcluirVersaoCommand","prj_listar.do");
 vfh.addComando("/vrs_listar.do","modulos.projeto.ListarVersoesCommand", "vrs_lista.jsp");
 vfh.addComando("/vrs_copiar.do", "modulos.projeto.CopiarVersaoCommand", "");
 […]
}
[…]
```

Quadro 20 – Mapeamento dos `Commands` na classe `FrontController`

Observando o quadro 20, pode-se compreender que quando o `FrontController` receber a requisição com *url* igual a "/prj_listar.do", disparará o método `executar()` da classe `modulos.projeto.ListarProjetosCommand` e, após a conclusão do processamento, enviará para o *browser* a página prj_lista.jsp.

A classe `FrontController` também é responsável por controlar qual fábrica de

objetos DAO será utilizada na aplicação. Para tanto, possui um atributo do tipo `DBDAOFactory`, o qual é instanciado de acordo com o parâmetro de nome “factory” definido no arquivo de configuração web.xml. No desenvolvimento do software, foi utilizada sempre a classe `MySqlDAOFactory`, a qual, conforme descrito no tópico 3.3.2.1, instancia objetos DAO implementados para funcionamento no SGBD MySql. A fábrica instanciada pelo `FrontController` também é passada como parâmetro para os `Commands` para que eles tenham acesso ao banco de dados da aplicação.

Para facilitar a compreensão do exposto acima a cerca do funcionamento deste núcleo do sistema, encontra-se um diagrama de seqüência na figura 24.

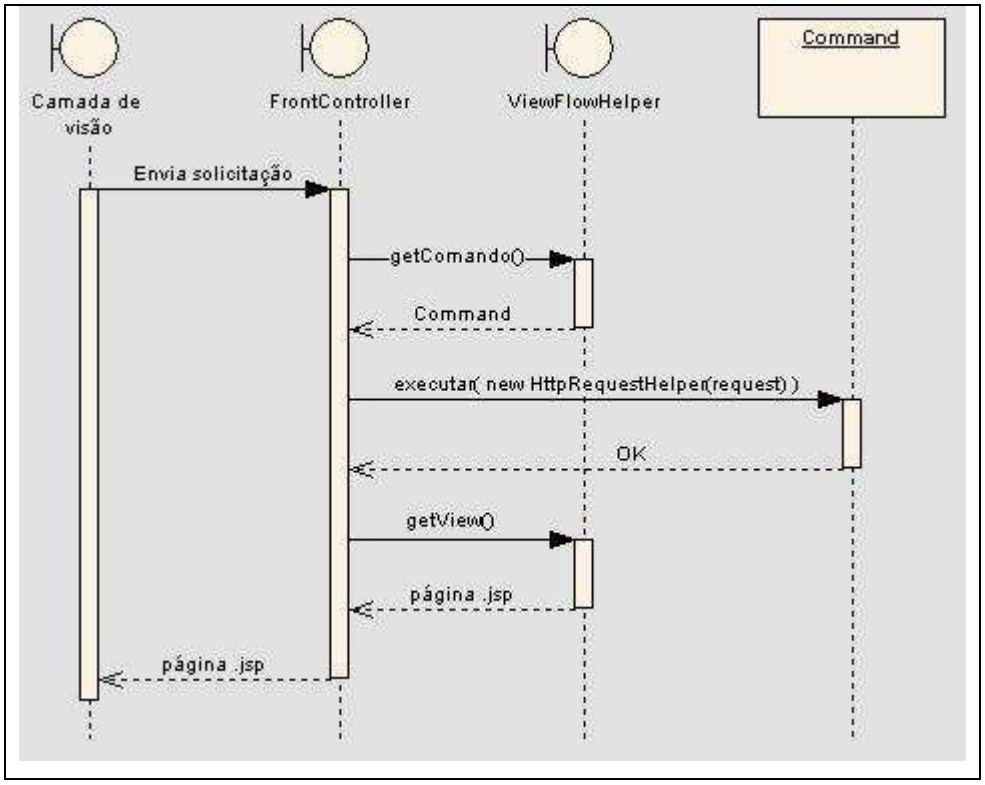


Figura 24 – Diagrama de seqüência do núcleo do sistema

3.3.2.3 Implementação dos cadastros básicos do sistema

Concluído o desenvolvimento da camada de persistência e do núcleo de processamento de requisições, foram implementadas as rotinas relativas aos cadastros de projetos, versões, usuários, tabelas, colunas, relacionamentos e diagramas.

Nesta fase, o esforço foi concentrado no desenvolvimento das telas JSP com os campos necessários para cada cadastro e também das respectivas classes `Command`

responsáveis pela recuperação, inclusão, alteração e exclusão dos registros. Sendo assim, para o cadastro de projetos, por exemplo, foram criadas as páginas prj_form.jsp (tela para inclusão, alteração e exclusão dos projetos) e prj_lista.jsp (tela com a listagem de todos os projetos cadastrados no sistema), além das classes `CarregaProjetoCommand` (responsável por carregar as informações de um determinado projeto), `GravarProjetoCommand` (para gravar os dados de um projeto no banco de dados), `ExcluirProjetoCommand` (apagar um projeto do banco de dados) e `ListarProjetosCommand` (carregar uma lista com todos os projetos cadastrados), sendo que esta estrutura é bastante semelhante para os demais cadastros citados anteriormente.

No caso das telas de listagem, as mesmas são apresentadas no *browser* após o processamento de um `Command` no qual houve o carregamento de uma lista com os respectivos registros, lista esta que é enviada para a página JSP pelo `FrontController`. O quadro 21 exibe o código-fonte da classe `ListarProjetosCommand`.

```
package modulos.projeto;

import util.Command;
import util.WmException;
import util.RequestHelper;

public class ListarProjetosCommand extends Command {

	public String executar(RequestHelper rh) throws WmException {
		//joga para o atributo "lista" todos os projetos cadastrados
		rh.setAtributo("lista",getDbFactory().getProjetoDAO().listar());
		return "OK";
	}

}
```

Quadro 21 – Código-fonte da classe `ListarProjetosCommand`

Como pode ser visto no quadro 21, o `Command` não possui sequer uma linha de comandos SQL para buscar a listagem dos projetos cadastrados no banco de dados. Ao invés disso, pede para a sua fábrica de objetos DAO instanciar um `ProjetoDAO`, cujo método `listar()` é executado para acessar o banco de dados e carregar uma lista de objetos do tipo `Projeto`, com os dados de todos os projetos cadastrados no sistema. Esta lista, por sua vez, é inserida no `RequestHelper` como um atributo, sendo que assim será enviada pelo `FrontController` para a página responsável por sua exibição. Neste ponto pode-se observar a clara divisão entre a interface e as regras de negócio (proporcionada pela arquitetura MVC), já que o `Command` apenas retorna uma lista sem que tenha conhecimento da forma como os dados da mesma serão apresentados para o usuário.

No desenvolvimento dos JSP, foram utilizadas as *tags* da *JavaServer Pages Standard Tag Library* (JSTL) para evitar a existência de código Java puro nestes arquivos. Segundo a

Sun Microsystems (2007a), a JSTL é uma biblioteca para desenvolvimento de páginas JSP que encapsula os principais métodos para desenvolvimento web em *tags* simples, possuindo *tags* para testes condicionais, iterações, formatação de texto, execução de comandos SQL e manipulação de documentos XML. O quadro 22 traz as principais *tags* disponibilizadas pela JSTL e que foram utilizadas neste trabalho.

Nome da *tag*	**Objetivo**
`c:if`	Usada para testes condicionais simples.
`c:forEach`	*Tag* básica de iteração, usada para percorrer listas.
`c:import`	Usada para importar uma *url* em determinada parte da página jsp
`c:out`	Exibe um texto na página.
`c:choose`	Usada para testes condicionais, tendo o mesmo funcionamento do comando `switch` em Java.
`c:when`	Utilizada em conjunto com a *tag* `c:choose`, para testar se determinada expressão é verdadeira ou não.

Fonte: adaptado de Sun Microsystems (2007b)

Quadro 22 – Principais *tags* da JSTL utilizadas no trabalho

Neste trabalho, foi utilizada a implementação da JSTL distribuída pelo Projeto Jakarta (2004), sendo esta implementação descrita como de código aberto (*open-source*). O quadro 23 demonstra o código-fonte da página prj_lista.jsp, onde a JSTL foi usada.

```
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<html>
   <head>
      <link rel="stylesheet" href="css/wm.css" type="text/css"/>
   </head>
<body>

<a href='<c:url value="prj_tela.do?codigo=0"/>'>Adicionar</a>
<br><br>
<table>
  <tr>
    <th>Abrir</th>
    <th>Código</th>
    <th width="500">Descrição</th>
  </tr>
  <c:forEach items="${lista}" var="projeto" varStatus="linha">
  <tr>
     <td align="center" class="linha_lista">
       <a href='<c:url  value="prj_tela.do?codigo=${projeto.codigo}"/>'>
         <img border=0 src="img/abrir.jpg" width="20" height="20">
       </a>
     </td>
     <td class="linha_lista"><c:out value="${projeto.codigo}"/></td>
     <td class="linha_lista"><c:out value="${projeto.nome}" /></td>
  </tr>
  </c:forEach>
</table>
<br>
</body>
</html>
```

Quadro 23 – Código-fonte da página prj_lista.jsp

Conforme exibido no quadro anterior, não há código Java na página, sendo que o mesmo foi substituído pelas *tags* próprias da JSTL, como a `forEach`, utilizada para

percorrer a lista de projetos.

As telas de listagem foram desenvolvidas de forma bem simples, não utilizando nenhum recurso ou técnica mais avançada como o Ajax, por exemplo. Nestas páginas, a única tecnologia diferenciada empregada na implementação foi a biblioteca de *tags* da JSTL.

Nas telas de cadastro, por sua vez, as técnicas do Ajax foram amplamente utilizadas, especialmente para fazer a comunicação com o servidor de forma assíncrona e sem a atualização de toda a página. Estas telas foram desenvolvidas de forma a ter uma interface semelhante a de aplicações *desktop*, como mostra a figura 25. No topo, foram colocados os seguintes botões:

a) botão Primeiro: carrega os dados do primeiro registro cadastrado;
b) botão Anterior: carrega os dados do registro anterior ao posicionado;
c) botão Próximo: carrega os dados do registro posterior ao posicionado;
d) botão Último: carrega os dados do último registro;
e) botão Pesquisar: abre a tela de listagem dos registros cadastrados;
f) botão Gravar: utilizado para inserir e/ou alterar registros;
g) botão Cancelar: cancela as alterações que tenham sido feitas em algum registro;
h) botão Novo: coloca a tela em estado de inserção;
i) botão Excluir: utilizado para excluir o registro do banco de dados.

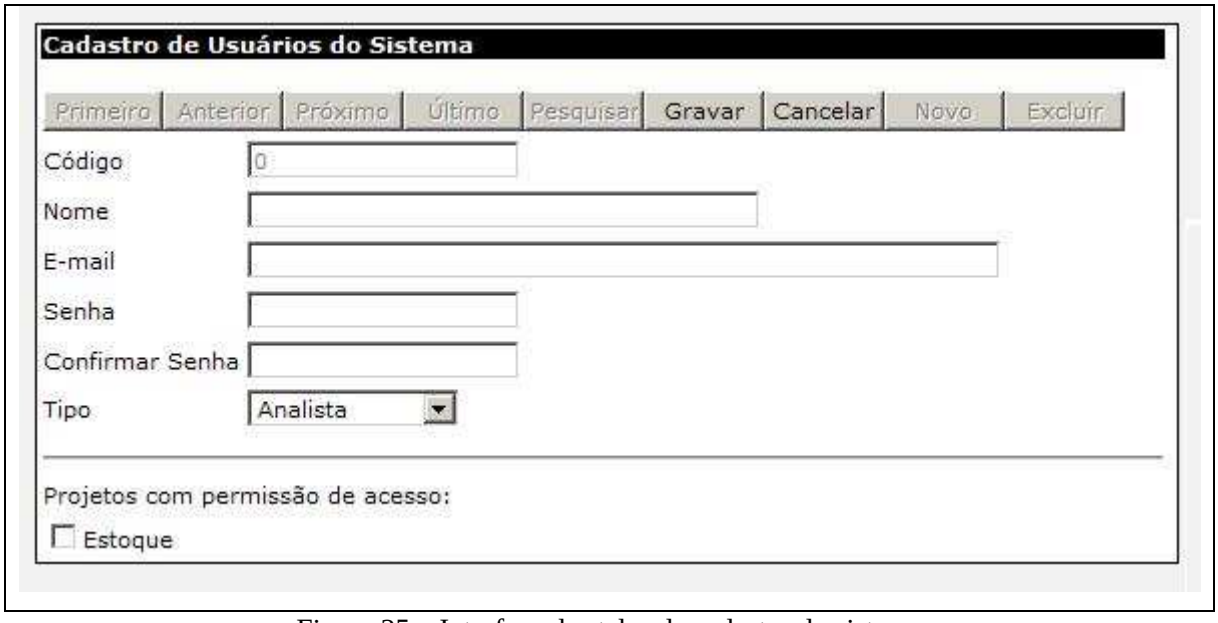


Figura 25 – Interface das telas de cadastro do sistema

Com exceção do botão Pesquisar, os demais utilizam Ajax e, quando acionados, executam suas funções e em seguida atualizam apenas os valores dos campos, sem a necessidade de atualizar a página inteira. O botão Novo utiliza apenas código JavaScript para limpar todos os campos da tela, como pode ser visto no quadro 24.

```
function novo1(){
  document.getElementById("codigo").value = 0;
  document.getElementById("nome").value = "";
  document.getElementById("observacao").value = "";
  document.getElementById("nome").focus();
}
```

Quadro 24 – Código-fonte do botão Novo do cadastro de diagramas

Os demais botões se comunicam assincronamente com o servidor através do objeto `XMLHttpRequest`, isto é, sempre que um botão é acionado, um código JavaScript é executado para criar o objeto `XMLHttpRequest` e enviar a requisição ao servidor. Após o servidor processar a solicitação, há um outro código JavaScript para tratar o retorno.

Para facilitar a implementação das chamadas assíncronas nas telas, foi criado um arquivo de código JavaScript contendo as funções básicas necessárias para envio das solicitações ao servidor e tratamento do seu retorno, sendo este arquivo importado pelas páginas onde será feito o uso desse tipo de comunicação. O código-fonte desse arquivo pode ser visto no quadro 25.

```
var xmlHttp;

function createXMLHttpRequest(){
    if (window.XMLHttpRequest) { // Usado para Mozilla, Safari
       xmlHttp = new XMLHttpRequest();
    } else
       if (window.ActiveXObject) { // Usado pro IE
              try {
                  xmlHttp = new ActiveXObject("Msxml2.XMLHTTP");
             } catch (e) {
              try {
                       xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
                  } catch (e) {}
              }
         }
}

function handleStateChange(){
    if (xmlHttp.readyState == 4){
       if  (xmlHttp.status == 200){
            parseResults();
       }  else
            alert(xmlHttp.statusText);
    }
}

function executeXMLHttpRequest(metodo,url,assincrono,parametros){
    createXMLHttpRequest();
    xmlHttp.open(metodo,url,assincrono);
    xmlHttp.onreadystatechange = handleStateChange;
    xmlHttp.setRequestHeader("Content-Type","application/x-www-form-
urlencoded");
    xmlHttp.send(parametros);
}
```

Quadro 25 – Código-fonte do arquivo ajax.js

A função `createXMLHttpRequest()`, como o próprio nome dá a entender, cria

um objeto `XMLHttpRequest`, o qual é atribuído para a variável `xmlHttp`. Já a função `executeXMLHttpRequest()` é usada para enviar a requisição ao servidor. Pode-se observar que nesta função o atributo `onreadystatechange` da variável `xmlHttp` recebe o valor `handleStateChange`, que nada mais é do que outra função constante no mesmo arquivo, a qual é responsável por receber o retorno do servidor. Quando este retorno chega ao objeto, é disparada a função `parseResults()`, que foi implementada em cada tela para tratar o retorno esperado em cada situação. O quadro 26 mostra como foi implementada a chamada assíncrona para carregar os dados do cadastro de diagramas.

```
function carrega1(){
  var url = "dia_carregar.do?";
  var parametros = "projeto=" + document.getElementById("projeto").value;
  parametros = parametros + "&versao="+document.getElementById("versao").value;
  parametros = parametros + "&tipo=atual";
  parametros = parametros + "&atual="+document.getElementById("codigo").value;
  parametros = parametros + "&ajax=sim";
  executeXMLHttpRequest("POST",url,true,parametros);
}
```

Quadro 26 – Chamada assíncrona para carregar os dados do cadastro de um diagrama

Já o quadro 27 mostra a implementação da função que trata o retorno da chamada assíncrona ilustrada no quadro anterior.

```
function parseResults(){
  var resultado = "";
  resultado = mlHttp.responseXML.getElementsByTagName("resultado")[0].firstChild.nodeValue;
  if   (resultado == "OK"){ //processou sem erros
    var codigo = 0;
    if (xmlHttp.responseXML.getElementsByTagName("codigo")[0].firstChild != null){
       codigo = xmlHttp.responseXML.getElementsByTagName("codigo")[0].firstChild.nodeValue;
    }
    var nome = "";
    if (xmlHttp.responseXML.getElementsByTagName("nome")[0].firstChild != null){
       nome = xmlHttp.responseXML.getElementsByTagName("nome")[0].firstChild.nodeValue;
    }
    var observacao = "";
    if (xmlHttp.responseXML.getElementsByTagName("observacao")[0].firstChild != null){
      observacao =
xmlHttp.responseXML.getElementsByTagName("observacao")[0].firstChild.nodeValue;
    }
    document.getElementById("codigo").value = codigo;
    document.getElementById("nome").value = nome;
    document.getElementById("observacao").value = observacao;
    trataEstado("C");
  } else if (resultado == "ERRO") {
  //erro
  var msg = xmlHttp.responseXML.getElementsByTagName("mensagem")[0].firstChild.nodeValue;
  alert(msg);
  }
escondeDivMsg();
}
```

Quadro 27 – Código-fonte da função `parseResults()` do cadastro de diagramas

Para os casos de chamadas assíncronas, as classes `Command` desenvolvidas para atender as requisições no servidor foram implementadas de forma que retornem um arquivo XML para a página, para que esta última possa interpretar mais facilmente o retorno. Um exemplo da montagem deste XML no servidor pode ser visto no quadro 28, que contem o

código-fonte da classe `Command` responsável por obter os dados de um diagrama no banco de dados.

```
package modulos.diagrama;
[...]
public class CarregaDiagramaCommand extends Command {

	public String executar(RequestHelper rh) throws WmException {

		[...]
		//montar o xml
		StringBuffer registro = new StringBuffer("<diagrama>");
		registro.append("<codigo>");
		registro.append(codigo);
		registro.append("</codigo>");
		registro.append("<nome>");
		registro.append(nome);
		registro.append("</nome>");
		registro.append("<observacao>");
		registro.append(observacao);
		registro.append("</observacao>");
		registro.append("<resultado>");
		registro.append("OK");
		registro.append("</resultado>");
		registro.append("</diagrama>");
		rh.setAtributo("xml", registro);
		return "OK";
	}
}
```

Quadro 28 – Montagem do XML que será retornado para a tela de cadastro de diagramas

No caso específico do cadastro de usuários, foi necessário desenvolver uma classe específica para possibilitar que a senha do usuário fosse gravada no banco de dados de forma criptografada, atendendo assim ao requisito não funcional RNF006. Esta classe, chamada `Criptografia`, implementa o algoritmo *Message-Digest algorithm 5* (MD5), o qual, segundo Vale (2007), é um algoritmo de *hash* unidirecional de 128 bits, desenvolvido pela RSA Data Security e de domínio público, descrito na RFC 1321, sendo utilizado por softwares com protocolo ponto-a-ponto (P2P), verificação de integridade de arquivos e *logins*. Ainda segundo Vale (2007), a principal característica deste algoritmo é que, uma vez convertido, um *hash* MD5 não pode ser transformado novamente no texto que lhe deu origem, o que faz com que o método de verificação seja a comparação das duas *hash* (uma da base de dados e a outra da conversão do texto informado pelo usuário). O código-fonte da classe `Criptografia` pode ser visto no quadro 29.

```
package util;

import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

public class Criptografia {

  private static MessageDigest md = null;
  static {
       try {
              md = MessageDigest.getInstance("MD5");
```

```
       } catch (NoSuchAlgorithmException ex) {
             ex.printStackTrace();
       }
       }

  private static char[] hexCodes(byte[] text) {
    char[] hexOutput = new char[text.length * 2];
    String hexString;

    for (int i = 0; i < text.length; i++) {
      hexString = "00" + Integer.toHexString(text[i]);
      hexString.toUpperCase().getChars(hexString.length() - 2,
                                 hexString.length(), hexOutput, i * 2);
    }
    return hexOutput;
  }

  public static String criptografar(String pwd) {
       if (md != null) {
         return new String(hexCodes(md.digest(pwd.getBytes())));
       }
       return null;
  }
}
```

Quadro 29 – Código-fonte da classe `Criptografia`

3.3.2.4 Implementação do módulo de modelagem gráfica

A modelagem de um banco de dados no sistema de forma gráfica visa disponibilizar ao analista uma interface amigável, na qual seja possível visualizar facilmente a estrutura do banco que está sendo projetado. Essa interface exibe, basicamente, as tabelas cadastradas (com suas colunas) e os relacionamentos entre elas, sendo as tabelas representadas por meio de retângulos em cujo interior aparecem o nome das colunas e seus respectivos tipos de dados. Já os relacionamentos são representados por meio de linhas conectadas às duas tabelas relacionadas, com uma seta na extremidade ligada à tabela-mestre.

No desenvolvimento desta parte do sistema, o Ajax foi utilizado tanto ou mais do que nas telas de cadastro. Utilizando JavaScript, DOM, CSS e o objeto `XMLHttpRequest` foi possível criar recursos para exibir as tabelas e relacionamentos, criar menus de contexto (botão direito do *mouse*) e arrastar e soltar (*drag-and-drop*) as tabelas.

O primeiro passo foi o desenvolvimento da rotina que adiciona as tabelas no diagrama. Esta rotina consiste em um código JavaScript que desenha um retângulo na tela quando o usuário aciona o botão responsável por adicionar tabelas. O código dessa função está no quadro 30.

```
function novaTabela(){
  countTb ++;
  var nomeTabela = "tabela"+countTb;
  var d; var t;
  d = document.createElement("div");
  d.name = nomeTabela;
  d.id = nomeTabela;
  d.setAttribute("id",nomeTabela);
  d.ondblclick = montaTabela;
  d.onmousedown = clickTabela;
  d.novo = "sim";
  d.style.cssText = "background-color: #FFFFFF";
  d.setAttribute("class","tb_diagrama");
  d.setAttribute("className","tb_diagrama");
  var iH = "<table width=100% border=1><tr><td style='font-family: Tahoma;
font-size: 8pt'><p align=center>" + nomeTabela + "</td></tr></table>";
  d.innerHTML = iH;
  document.getElementById("camada1").appendChild(d);
  dragdrop(nomeTabela,nomeTabela);
}
```

Quadro 30– Código JavaScript que adiciona uma tabela ao diagrama

As tabelas são “desenhadas” na tela através da criação de um objeto HTML do tipo DIV. Para que seja possível arrastar e soltar as tabelas, são associadas funções aos eventos `onmouseup` e `onmousemove` da página, funções estas que tratam, respectivamente, o momento em que o usuário deixa de pressionar o *mouse* e o momento em que o usuário move o *mouse*. Além disso, cada DIV (tabela) criado tem uma função associada a seu evento `onmousedown`, para tratar o momento em que o usuário clica sobre o mesmo. Sendo assim, o funcionamento básico do processo de *drag-and-drop* das tabelas é o seguinte:

a) quando o usuário clica sobre a tabela, é disparada a função JavaScript associada ao evento `onmousedown` do DIV. Essa função alimenta uma variável global indicando qual foi a tabela sobre a qual o usuário clicou.
b) ao mover o *mouse* com o botão esquerdo pressionado, é executada a função definida para o evento `onmousemove` da página, função esta que reposiciona na tela o DIV sobre o qual o usuário clicou, de acordo com a posição do *mouse*.
c) ao soltar o botão do *mouse*, executa-se a função associada ao evento `onmouseup` do documento, liberando a variável inicializada no primeiro passo.

Para que o usuário possa informar os dados da tabela e das suas colunas, foram disponibilizadas duas formas de acesso à tela de cadastro de tabelas: através de um duplo-clique sobre o DIV ou através do menu de contexto que aparece ao clicar com o botão direito do *mouse* sobre o mesmo. Ambas as opções levam o usuário à tela de cadastro de tabelas (a mesma que já havia sido desenvolvida) que é aberta em uma nova página do *browser*. Após informar os dados desejados, o usuário aciona um botão que fecha esta tela e faz com que, através de uma outra função JavaScript, o DIV relativo à tabela em questão seja redesenhado

para exibir corretamente o nome da tabela e os dados de suas colunas. Esta função que atualiza o DIV na tela utiliza um objeto `XMLHttpRequest` para obter as informações necessárias do servidor assincronamente, sem a necessidade de atualizar a página inteira, ou seja, apenas o DIV é atualizado. O quadro 31 mostra o código da função que faz a solicitação ao servidor utilizando o objeto `XMLHttpRequest`.

```
function recarregaTabela(idTab){
  divAtivo = document.getElementById(idTab);
  //execução da requisiçao Ajax pra carregar os dados da tabela
  var url = "tab_carregar.do?";
  var param = "tipo=atual";
  param = param + "&atual="+divAtivo.name;
  param = param + "&projeto="+document.getElementById("i_projeto").value;
  param = param +"&versao="+document.getElementById("i_versao").value;
  param = param + "&ajax=sim";
  executeXMLHttpRequest("POST",url,false,param);
}
```

Quadro 31 – Código-fonte da solicitação assíncrona de dados da tabela

Após concluir a etapa de desenvolvimento do cadastro das tabelas no diagrama, deu-se a implementação da rotina de inclusão de relacionamentos entre tabelas. Para cadastrar um relacionamento, foi incluída uma opção no menu de contexto das tabelas, a qual abre a tela de cadastro de relacionamentos em uma nova página do *browser*. Depois de feito o cadastro, ao pressionar o botão para adicionar o relacionamento no diagrama, a tela de cadastro é fechada e o relacionamento é desenhado na tela. Para formar a linha e a seta que representam um relacionamento, são criados inúmeros pequenos DIVs, cada um ocupando 2 *pixels* de altura e outros 2 de largura, sendo que foi implementado um algoritmo que, a partir das posições das tabelas mestre e filha no diagrama, cria estes inúmeros DIVs até conectar uma tabela à outra. Parte do código deste algoritmo está no quadro 32.

```
function desenhaPonto(top, left, desc){
  var d = document.createElement("div");
  d.name = "rel";
  d.id = "rel";
  d.nomeRel = desc;
  d.style.cssText = "position:absolute;background:black;width:2px;height:2px;font-
size:2px";
  d.style.cssText = d.style.cssText + ";top:"+top+"px;left:"+left+"px";
  d.ondblclick = function (){
              editarRel(desc);
                  }
  d.onmousedown = function (){
                      [...]
                   }
  document.getElementById("camada1").appendChild(d);
}

function desenhaRelacionamento(mestre,filha,nome){
  var tabMestre = null;
  var tabFilha = null;
  var tabelas = document.getElementsByTagName("div");
  for (var x = 0; x < tabelas.length; x ++) {
       if     (tabelas[x].getAttribute("class") == "tb_diagrama") {
              if     (tabelas[x].name == mestre) {
                     tabMestre = tabelas[x].id;
              }
              if     (tabelas[x].name == filha) {
```

```
                    tabFilha = tabelas[x].id;
                }
        }
  }
  if   ((tabMestre != null) && (tabFilha != null)){
   recarregaTabela(tabFilha); //pode ter sido alterada com a criação do relacionamento
   var mL = document.getElementById(tabMestre).offsetLeft; //lado esquerdo da mestre
   var mW = document.getElementById(tabMestre).offsetWidth; //largura da mestre
   var mH = document.getElementById(tabMestre).offsetHeight; //altura da mestre
    var mT = document.getElementById(tabMestre).offsetTop; //topo da mestre
    var fL = document.getElementById(tabFilha).offsetLeft; //lado esquerdo da filha
    var fW = document.getElementById(tabFilha).offsetWidth; //largura da filha
    var fH = document.getElementById(tabFilha).offsetHeight; //altura da filha
    var fT = document.getElementById(tabFilha).offsetTop; //topo da filha
    var mR = mL + mW; //lado direito da mestre
    var fR = fL + fW; //lado direito da filha
    var mB = mT + mH; //base da mestre
    var fB = fT + fH; //base da filha
    […]
    if (mR < fL) { //se a tabela mestre estiver à esquerda da filha
                            […]
        var x = b;
        //desenhar verticalmente até a metade da filha
        for (i = tM; i <= tF; i = (i + 2)){  //se a filha estiver pra baixo
                desenhaPonto(i,x,nome);
        }
        for (i = tM; i > tF; i = (i - 2)){  //se a filha estiver pra cima
                desenhaPonto(i, x,nome);
        }
        //desenhar horizontalmente até a filha
        for (i = b; i <= fL; i = (i + 2)){
                desenhaPonto(tF,i,nome);
        }

        //desenhar a seta
        i = mR + 1;
        t = tM + 1;
        desenhaPonto(t, i,nome);
                            […]
    }
[…]
}
```

Quadro 32 – Trecho do código-fonte da rotina que desenha os relacionamentos

Por fim, foi implementada a rotina de gravação da modelagem gráfica, para que os dados das tabelas e relacionamentos adicionados no diagrama sejam salvos no banco de dados. No caso das tabelas, são gravadas as coordenadas (topo e esquerda) onde as mesmas se encontravam no momento em que o usuário optou por salvar o diagrama, para que ao reabri-lo, os objetos estejam nas mesmas posições. A gravação dos dados do diagrama também é feita através do objeto `XMLHttpRequest`, conforme ilustrado no quadro 33.

```
function gravar(){
  var url = "dia_gravar_modelo.do?";
  var parametros = "projeto="+document.getElementById("i_projeto").value;
  parametros = parametros + "&versao="+document.getElementById("i_versao").value;
  parametros = parametros + "&diagrama="+document.getElementById("i_codigo").value;
  parametros = parametros + "&ajax=sim";
  var tabelas = document.getElementsByTagName("div");
  var x;
  for (x = 0; x < tabelas.length; x ++) {
       if     (tabelas[x].getAttribute("class") == "tb_diagrama") {
              if     (tabelas[x].novo != "sim") {
                     parametros = parametros + "&tabela=" + tabelas[x].name;
                     parametros = parametros + "&posEsq=" + tabelas[x].offsetLeft;
                     parametros = parametros + "&posTop=" + tabelas[x].offsetTop;
              }
       }
  }
  //relacionamentos
```

```
  var r;
  for (x = 0;x<aRel.length;x++){
       r = aRel[x];
       parametros = parametros + "&rel=" + r.nome;
  }
  mostraDivMsg();
  executeXMLGravacao("POST",url,true,parametros);
}
```

Quadro 33 – Código-fonte da solicitação assíncrona para gravação da modelagem

3.3.2.5 Implementação do módulo de geração de *scripts*

O desenvolvimento deste módulo foi dividido nas seguintes etapas:

a) implementação das classes e rotinas responsáveis pela geração dos *scripts* de criação do banco de dados para os SGBD Oracle e MSSqlServer;
b) implementação das classes e rotinas para geração dos *scripts* de atualização do banco de dados para os mesmos SGBD supra citados;
c) criação das classes e rotinas para geração do relatório de diferenças entre versões.

As rotinas foram desenvolvidas de forma a gerar *scripts* compatíveis com a versão 10g do Oracle e com a versão 2000 do MSSqlServer, sendo que os testes durante a implementação foram feitos nessas duas versões destes SGBD.

A tela do sistema onde são cadastradas tabelas e suas colunas foi desenvolvida de forma que o usuário informe tipos de dados próprios do sistema, com nomenclaturas em Língua Portuguesa (Numérico, Texto Variável, Data, etc.), ao invés de tipos próprios de algum SGBD. Sendo assim, o primeiro passo foi relacionar os tipos do sistema com os tipos nativos do Oracle e do MSSqlServer, relacionamento este feito com base nas informações de Oracle (2005) e Battisti (2001, p.155-157) e descrito no quadro 34.

WebModeler	**Oracle**	**MSSqlServer**
Texto Variável	Varchar2	Varchar
Texto Fixo	Char	Char
Inteiro	Integer	Int
Inteiro Curto	Smallint	Smallint
Inteiro Longo	Integer	Bigint
Numérico	Decimal	Decimal
Data	Date	DateTime
Data/Hora	DateTime	DateTime
Binário	Blob	Binary

Quadro 34 – Relacionamento entre os tipos de dados do sistema e dos SGBD

Uma vez definido o relacionamento entre os tipos de dados, foram desenvolvidas as classes e rotinas para geração dos *scripts* de criação do banco de dados. Foram criadas duas classes, `ScriptsOracle` e `ScriptsSqlServer`, as quais possuem métodos que retornam o texto adequado para criação, alteração e exclusão das tabelas, relacionamentos e colunas. Ambas possuem implementação bastante parecida e retornam comandos SQL idênticos, porém, decidiu-se criar duas classes separadas prevendo uma futura melhoria no sistema, onde poderiam ser disponibilizados recursos mais avançados e específicos de cada banco de dados.

O quadro 35 exibe o trecho do código-fonte da classe `ScriptsSqlServer` onde é gerado o *script* de criação de uma tabela para o banco de dados MSSqlServer.

```
public String scriptGeraTabela(Tabela tab) throws WmException{

  String retorno = "";
  String create = "create table ";
  create +=  tab.getNome();
  create += " (";
  retorno += create;

  //carregar as colunas
  ColunaDAO colDAO = this.factory.getColunaDAO();
  List<Coluna> listaColunas = colDAO.listar(tab.getProjeto(), tab.getVersao(),
tab.getNome());
  Coluna col;
  String coluna;
  String pk = "";

  /* formato p/ colunas:
   * NomeColuna tipoDado(tamanho,precisao) nulo,
   * Ex: Codigo integer not null,
   */
  for (int i = 0; i < listaColunas.size(); i ++){
       col = listaColunas.get(i);
       if     (col.isPertenceChave()){
              pk += col.getNome();
              pk += ",";
       }
       coluna = "  \n  ";
       coluna += col.getNome();
       coluna += " ";

       coluna += this.tipoDado(col.getTipoDado(), col.getTamanho(),
col.getPrecisao());
       if     (!col.isPermiteNulo())
              coluna += " not null,";
       else
              coluna += ",";
       retorno += coluna;
  }

  if   (!pk.equals("")){
       pk = pk.substring(0, pk.length() - 1); //tira a última vírgula
       coluna = "\n constraint CP_";
       coluna += tab.getNome();
       coluna += " primary key (";
       coluna += pk;
       coluna += ")";
       retorno += coluna;
  } else
```

```
        retorno = retorno.substring(0, retorno.length() - 1); //tira vírgula

   retorno += ") ;";
   return retorno;
}
```

Quadro 35 – Código-fonte da geração de *script* de criação de tabela para o MSSqlServer

A lógica implementada para geração destes *scripts* é bem simples: primeiro são inseridos os comandos para criar todas as tabelas e em seguida os comandos para criar todos os relacionamentos. Como pode ser visto no quadro anterior, também neste caso foram utilizadas as classes DAO para ter acesso às informações do banco de dados. Primeiro a classe busca todas as colunas da tabela e para cada uma delas monta uma linha na variável de retorno, com o nome da coluna, seu tipo e, se ela tiver sido cadastrada indicando que não poderá ter valores nulos, concatenar-se-á o texto *not null*. Além disso, a cada coluna da lista, a classe verifica se a mesma pertence à chave primária da tabela e, se pertencer, adiciona o seu nome na variável que será utilizada para montar o texto relativo à criação desta chave.

A sintaxe utilizada pelo sistema pare gerar os comandos de criação das tabelas e dos relacionamentos pode ser vista nos quadros 36 e 37.

```
create table NOME_TABELA(
  NOME_COLUNA TIPO_DADO [not null],
  constraint CP_NOME_TABELA primary key (COLUNAS_PK)
)
```

Quadro 36 – Sintaxe para criação das tabelas

```
alter table NOME_TABELA
  add constraint NOME_RELACIONAMENTO foreign key (COLUNAS)
     references TABELA_MESTRE (COLUNAS_MESTRE)
```

Quadro 37 – Sintaxe para criação dos relacionamentos

A rotina de geração de *scripts* de atualização foi desenvolvida para gerar um arquivo com todos os comandos SQL necessários para atualizar a estrutura de um banco de dados que esteja de acordo com uma versão mais antiga do projeto, igualando esta estrutura à da versão mais recente. A lógica implementada para este fim foi a seguinte:

a) inicialmente são inseridos no arquivo os comandos necessários para excluir relacionamentos entre tabelas que existiam na versão origem e que foram removidos na versão destino;
b) em seguida, os comandos para excluir relacionamentos que foram alterados de uma versão para outra;
c) na seqüência, comandos para excluir chaves primárias removidas ou alteradas;
d) o passo seguinte é inserir os comandos de criação das tabelas que foram incluídas

na versão destino;

e) a seguir, os comandos para excluir tabelas removidas na nova versão do projeto;

f) depois vêm os comandos para incluir as novas colunas, excluir as colunas removidas e modificar as colunas alteradas;

g) os comandos para recriar as chaves primárias incluídas ou alteradas são gerados na seqüência;

h) por fim, os comandos para recriar os relacionamentos incluídos ou alterados.

Para gerar estes *scripts*, foram desenvolvidas outras duas classes: `ScriptAtualizacaoOracle` e `ScriptAtualizacaoSqlServer`, que utilizam as classes `ScriptsOracle` e `ScriptsSqlServer` e, assim como estas últimas, também possuem implementação semelhante mas foram criadas separadas para facilitar a futura implementação de recursos próprios de cada banco. O quadro 38 mostra parte do código-fonte da classe `ScriptAtualizacaOracle`.

```
package modulos.script.oracle;
[...]
public class ScriptAtualizacaoOracle {
[...]
  public ScriptAtualizacaoOracle(DBDAOFactory factory) {
    super();
    this.factory = factory;
    this.compara = this.factory.getComparaVersoesDAO();
    this.geraScript = new ScriptsOracle(this.factory);
  }

  public String gerar(int projeto,int vrsOrigem, int vrsDestino) throws WmException{
    String retorno = "";
    List<Relacionamento>relExc = this.compara.getRelExcl(projeto,vrsOrigem,vrsDestino);
    [...]
    List<Tabela> tabNov = this.compara.getTabelasNovas(projeto,vrsOrigem, vrsDestino);
    [...]
    //Primeiro, excluir relacionamentos removidos na versao destino
    for(i = 0; i < relExc.size(); i ++){
       retorno += geraScript.scriptExcluiRelacionamento(relExc.get(i));
       retorno += "\n\n";
    }
    [...]
    //Terceiro, criar tabelas incluídas
    for(i = 0; i < tabNov.size(); i ++){
       retorno += geraScript.scriptGeraTabela(tabNov.get(i));
       retorno += "\n\n";
    }
    //Quarto, excluir tabelas removidas
    for(i = 0; i < tabExc.size(); i ++){
       retorno += geraScript.scriptExcluiTabela(tabExc.get(i));
       retorno += "\n\n";
    }
    [...]
    return retorno;
  }
}
```

Quadro 38 – Código-fonte da classe `ScriptAtualizacaoOracle`

É visível no quadro anterior a utilização da classe `ComparaVersoesDAO`, que foi desenvolvida para acessar o banco de dados do sistema, comparar as versões que estão sendo consideradas na geração do *script* e carregar listas contendo as tabelas, relacionamentos e

colunas que foram incluídas, modificadas ou excluídas na versão destino.

Esta mesma classe DAO também foi utilizada para implementar a rotina que gera o relatório de diferenças entre duas versões, o qual exibe uma listagem das tabelas que foram adicionadas, alteradas ou excluídas na versão destino. Para o caso das tabelas alteradas, são listados os nomes das colunas que foram incluídas ou removidas e, no caso das colunas alteradas, além do nome exibe o tipo de dado e o indicativo se permite nulo nas duas versões. O quadro 39 traz um exemplo da forma como as informações são apresentadas neste relatório.

```
/*
Relatório de diferenças entre versões
Projeto: Estoque
Versão anterior: 1.0
Nova versão: 2.0
Data: 16/05/2007 20:30:23
*/

* Tabelas adicionadas:
Tabela1 – Nome da tabela
Tabela2 – Nome da tabela

* Tabelas Excluídas:
TabelaX – Cadastro

* Tabelas Alteradas:
Colunas adicionadas:
Tabela3.Coluna1 – Texto Variável (3) – não permite nulo

Colunas excluídas:
Tabela3.Codigo – Inteiro

Colunas alteradas:
Tabela3.Nome - Texto Variável(15) - não permite nulo /  Texto Variável(20) - não permite nulo
```

Quadro 39 – Exemplo do relatório de diferenças entre versões

Os *scripts* de criação e atualização de base e também o relatório de diferenças entre versões são apresentados ao usuário na forma de um arquivo PDF. Para facilitar a geração deste arquivo, utilizou-se a biblioteca iText, a qual, segundo Lowagie (2007), permite gerar dinamicamente arquivos PDF contendo textos, tabelas e imagens. O quadro 40 demonstra a utilização dessa biblioteca na implementação do sistema.

```
[…]
//criar e abrir o documento
Document document = new Document();
document.addTitle("/* Projeto: Teste / Versão 1 */");
try {
    response.setContentType("application/pdf");
    PdfWriter.getInstance(document, response.getOutputStream());
    document.open();
} catch (DocumentException e) {
       e.printStackTrace();
}
[…]
document.add(new Paragraph("/*"));
document.add(new Paragraph("   Script de criação do banco de dados para SQLServer "));
document.add(new Paragraph("   Projeto: " + prj.getNome()));
```

```
document.add(new Paragraph("   Versão: " + vrsDes.getDescricao()));
document.add(new Paragraph("*/"));
[…]
document.close();
[…]
```

Quadro 40 – Utilização da biblioteca iText

Conforme apresentado no quadro 40, a utilização desta biblioteca pode ser considerada bastante simples: basta criar um objeto da classe `Document`, abrí-lo usando o método `open()` e em seguida adicionar o texto através do método `add()`. Após adicionar todo o texto, basta fechar o documento com o método `close()`. Além destes métodos, essa biblioteca disponibiliza inúmeros outros, como, por exemplo, para iniciar uma nova página no documento.

3.3.2.6 Definição dos estilos (cores, fontes) da interface da aplicação

Finalizada a implementação de todas as rotinas do software, foi feita a definição dos estilos da interface. Foram feitos inúmeros testes, com várias cores e tipos de fontes, até se chegar ao layout final, que foi definido no arquivo CSS listado no quadro 41.

```
body{
	font-family:Verdana;
	font-size:8pt;
	background-color:#f1f1f1;
}

button{
	font-family:Verdana;
	font-size:8pt;
}

input{
	font-family:Verdana;
	font-size:8pt;
}

select{
	font-family:Verdana;
	font-size:8pt;
}

textarea{
	font-family:Verdana;
	font-size:8pt;
}

table{
	font-family:Verdana;
	font-size:8pt;
}
```

```
th{
	background-color:#000000;
	color:#FFFFFF;
	text-align:left;
}

.linha_lista{
	font-family:Verdana;
	font-size:8pt;
	color:#000000;
	background-color:#FFFFFF;
	border:1px solid #000000;
}

.body_desenho{
	font-family:Verdana;
	background-color:#FFFFFF;
}

.form{
	font-family:Verdana;
	font-size:8pt;
	background-color:#FFFFFF;
	border:1px solid #000000;
}

.td_coluna_comum{
	font-family:Verdana;
	font-size:8pt;
	color:#000000;
	background-color:#FFFFFF;
	border:1px solid #000000;
}

.td_coluna_selecionada{
	font-family:Verdana;
	font-size:8pt;
	color:#FFFFFF;
	background-color:#000080;
	border:1px solid #000000;
}

.tb_diagrama{
	position:absolute;
	font-size:50%;
	z-index:3;
	top:1;
	left:1;
	display:block;
	width:150px;
	height:90px;
	border:1px solid #0000FF;
	background-color:#FFFFFF";
}

.td_diagrama{
	font-size: 24pt;
	font-family: Tahoma;
}
```

Quadro 41 – Código do arquivo de estilos wm.css

Este arquivo define o tipo da fonte, tamanho, cor, bordas e alinhamento dos objetos HTML como botões, campos de texto, listas de seleção, etc. e também define estilos que foram usados em implementações específicas, como o estilo .tb_diagrama, usado para definir o *layout* das tabelas no módulo de modelagem gráfica.

3.3.3 Operacionalidade da implementação

Aqui será demonstrado o funcionamento da implementação através de um estudo de caso de uma situação hipotética, no qual uma empresa de software contratou um novo analista para desenvolver um sistema básico de *Customer Relationship Management* (CRM).

O primeiro passo é o cadastramento do projeto e do analista, por parte do administrador do sistema, o qual acessará o software informando seu código e senha na tela de entrada exibida na figura 26.

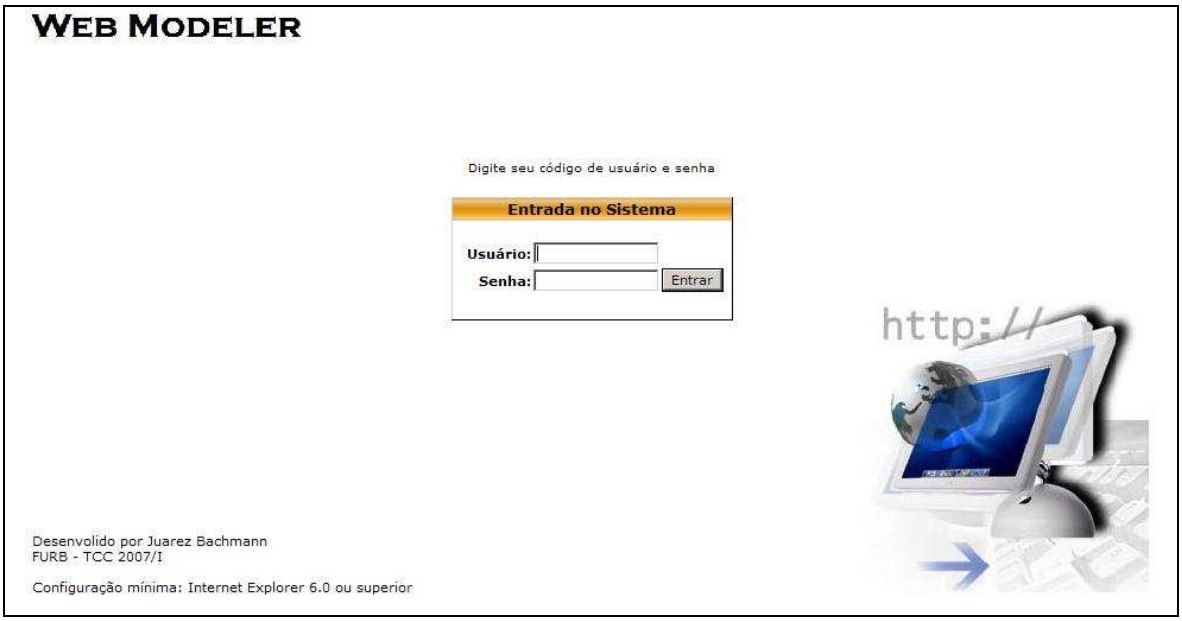


Figura 26 – Tela de entrada do sistema

Se o código e a senha informados estiverem corretos, será apresentada a tela principal do sistema, conforme a figura 27.

Figura 27 – Tela principal do sistema para o administrador

Através do item de menu Administrador, serão acessadas as telas de cadastro de projetos e usuários. Já no item Analista, estão todos os modelos de todos os bancos de dados projetados através do sistema.

Para cadastrar o novo analista, o administrador acessa o item de menu Usuários e o sistema exibe uma listagem dos usuários já cadastrados, como pode ser visto na figura 28.

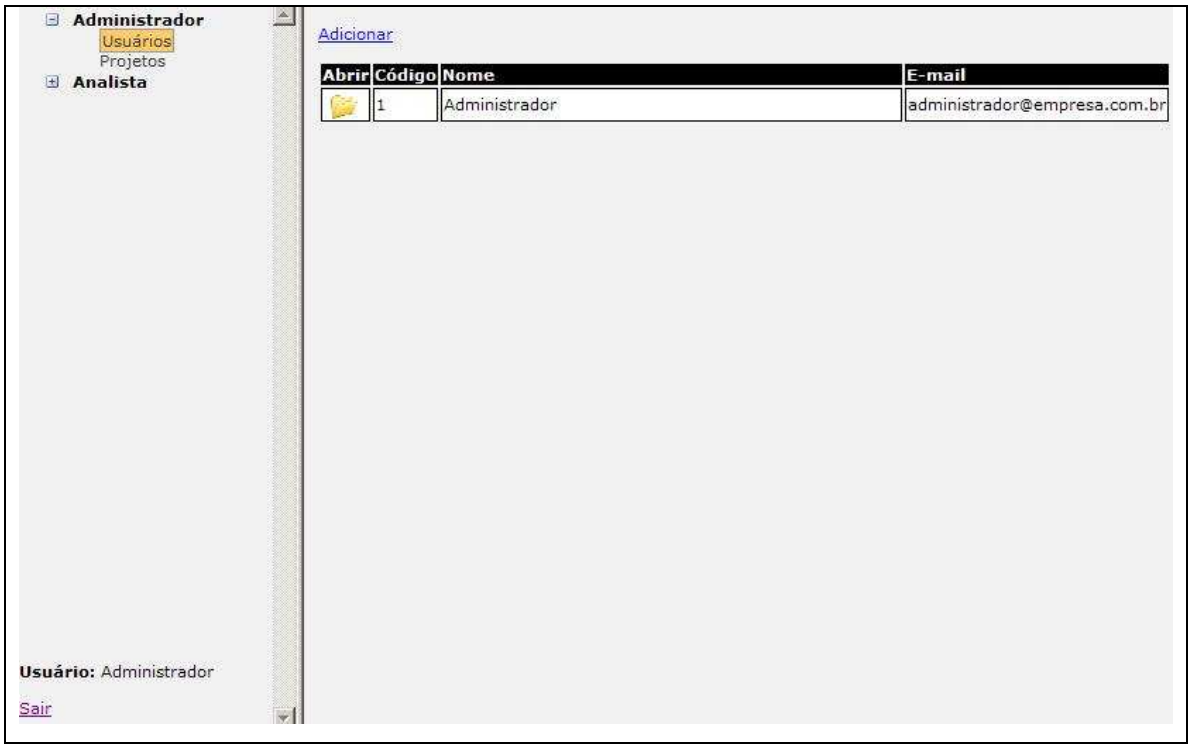


Figura 28 – Tela com a listagem dos usuários

Caso o administrador queira alterar algum dado de algum usuário já cadastrado, basta clicar sobre o ícone em forma de pasta na coluna Abrir do respectivo registro. Clicando no *link* adicionar que aparece acima da listagem, é aberta a tela de cadastro de usuários, onde o administrador informa os dados do analista, conforme exibido na figura 29.

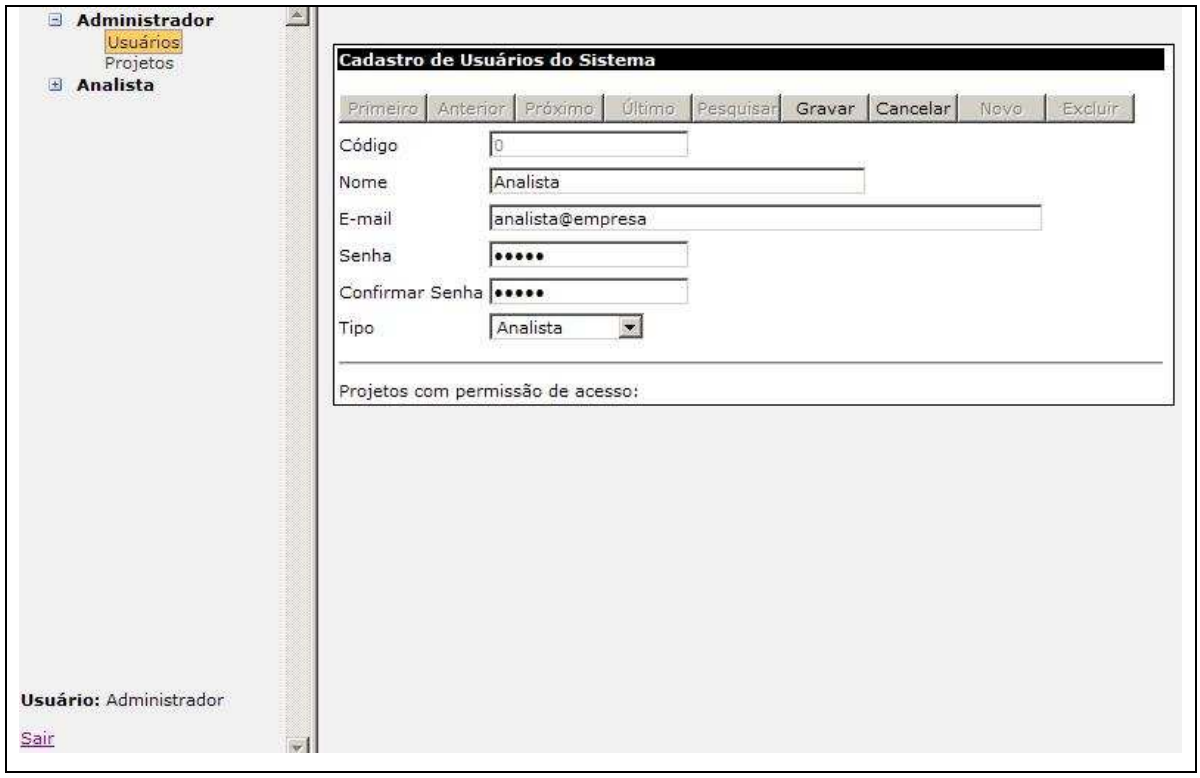


Figura 29 – Tela de cadastro de usuários

Após informar os dados na tela de cadastro, basta clicar no botão Gravar para que o novo usuário seja inserido no banco de dados. Dois campos merecem destaque nesta tela. O primeiro é o da senha, onde será definida a senha de acesso do usuário ao sistema. O outro é o campo Tipo, onde será definido se o usuário é um administrador (acesso total ao sistema) ou um analista (acesso às versões em aberto dos projetos aos quais esteja relacionado).

Em seguida, o administrador clica no item de menu Projetos para cadastrar o novo projeto. Ao clicar neste item de menu, também é exibida uma listagem dos projetos já cadastrados e, para incluir outro, o administrador clica sobre o *link* Adicionar para que seja exibida a tela de cadastro de projetos, onde os dados do novo projeto são inseridos, sendo possível, inclusive, definir neste momento quais usuários terão acesso ao mesmo, bastando para tanto marcar os nomes dos usuários desejados. Essa tela de cadastro é exibida na figura 30.

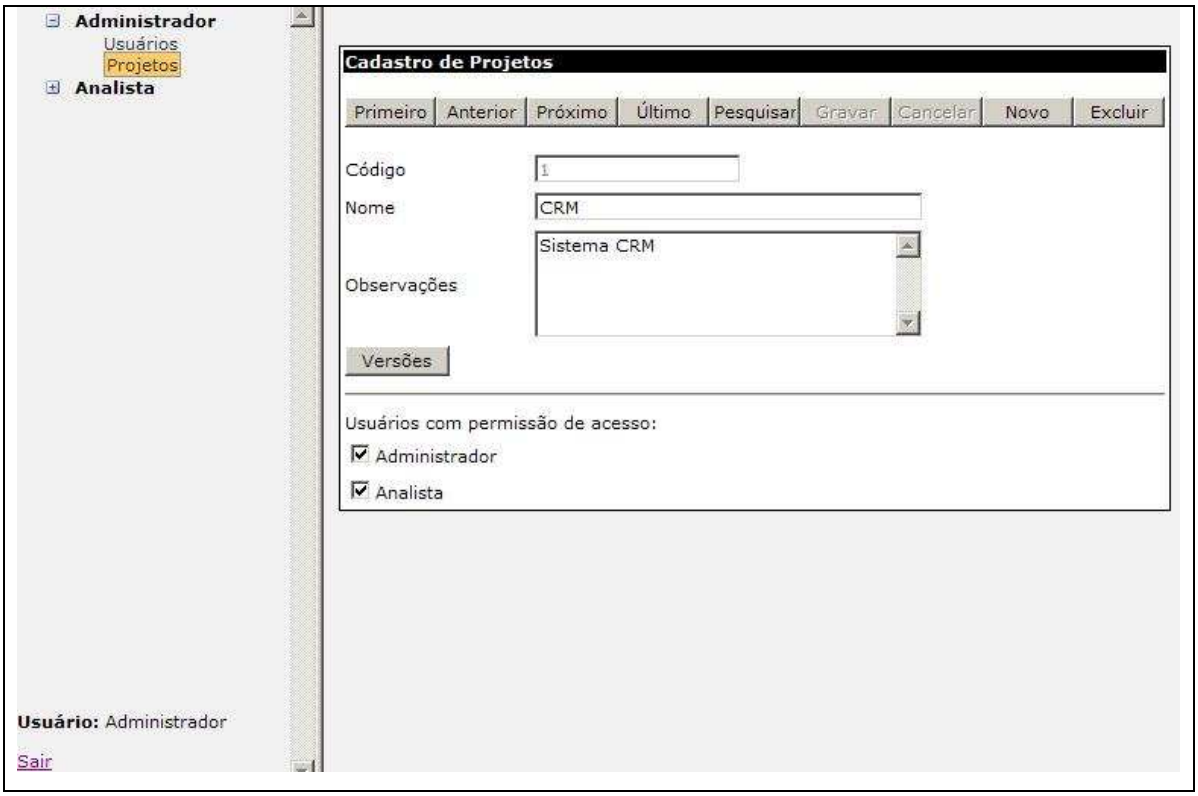


Figura 30 – Tela de cadastro de projetos

Como é bastante comum que com o passar do tempo um projeto de banco de dados seja alterado inúmeras vezes, seja para modificar requisitos e funcionalidades existentes ou para adicionar funcionalidades ao software modelado, o sistema permite que sejam cadastradas versões de cada projeto, a fim de que o histórico de alterações fique registrado. Sendo assim, todas as informações relativas às tabelas, colunas, relacionamentos e diagramas são armazenadas pelo sistema no nível de versões. Para cadastrar uma versão do projeto para que o analista possa trabalhar nela, o administrador clica no botão Versões que aparece na tela de cadastro do projeto. Novamente o sistema apresentará uma tela de listagem, dessa vez exibindo as versões cadastradas para o projeto em questão, sendo que para incluir uma versão basta clicar no *link* Adicionar, o que fará com que o sistema exiba a tela de cadastro de versões, ilustrada na figura 31.

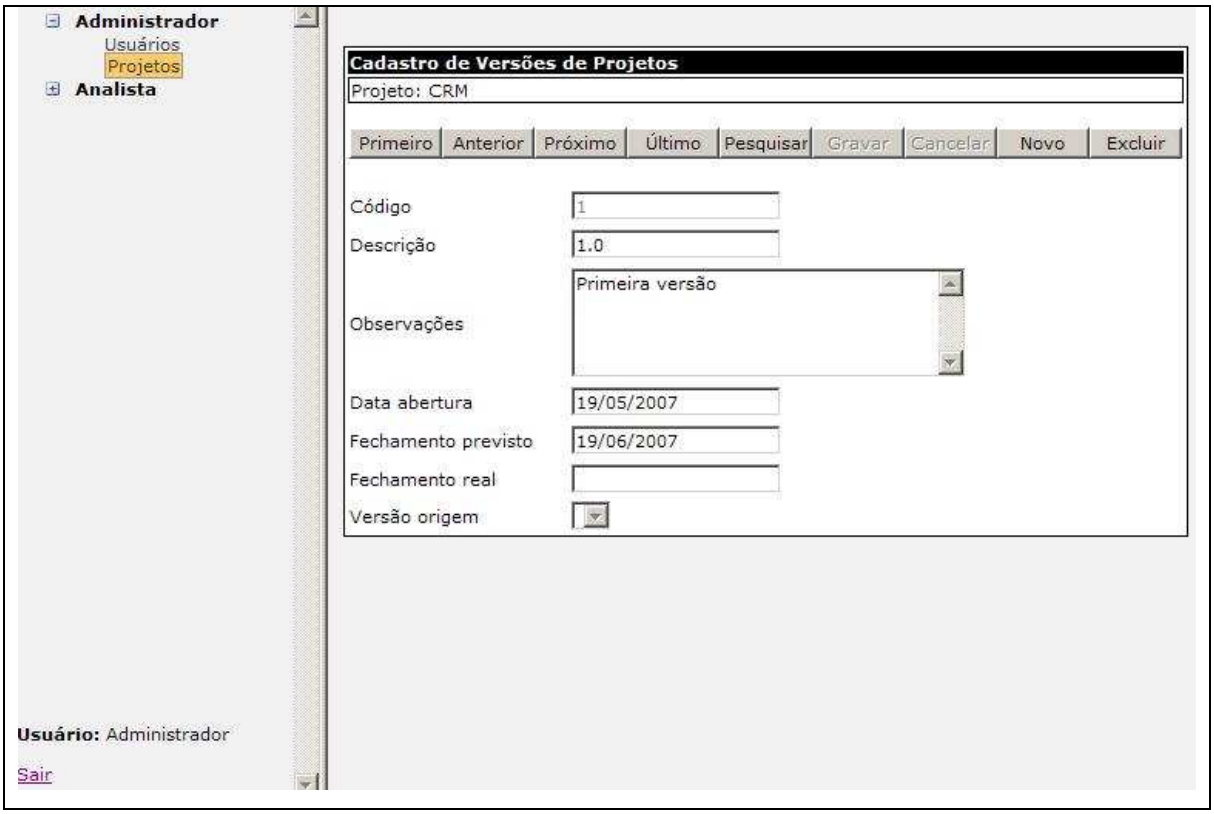


Figura 31 – Tela de cadastro de versões

Ao cadastrar uma versão, o administrador precisa informar as data de abertura e fechamento previsto da mesma, pois, com isso, os analistas só terão acesso à versão a partir da data de abertura estipulada. A data de fechamento previsto é apenas informativa, não há nenhuma consistência a ser feita com ela. Quando o administrador desejar que nenhum analista possa acessar a versão, basta informar a data de fechamento real. Ao criar uma versão, há ainda a opção de informar uma versão de origem, sendo que nesse caso a nova versão é criada como uma cópia da anterior.

Com isto, o administrador efetuou todos os cadastramentos necessários para que o analista possa começar a trabalhar, bastando apenas que ele acesse o sistema informando o seu código de usuário e sua senha. A tela principal para o analista, a qual aparece na figura 32, é ligeiramente diferente da tela principal do administrador, já que o analista não tem acesso ao cadastro de projetos e de usuários.

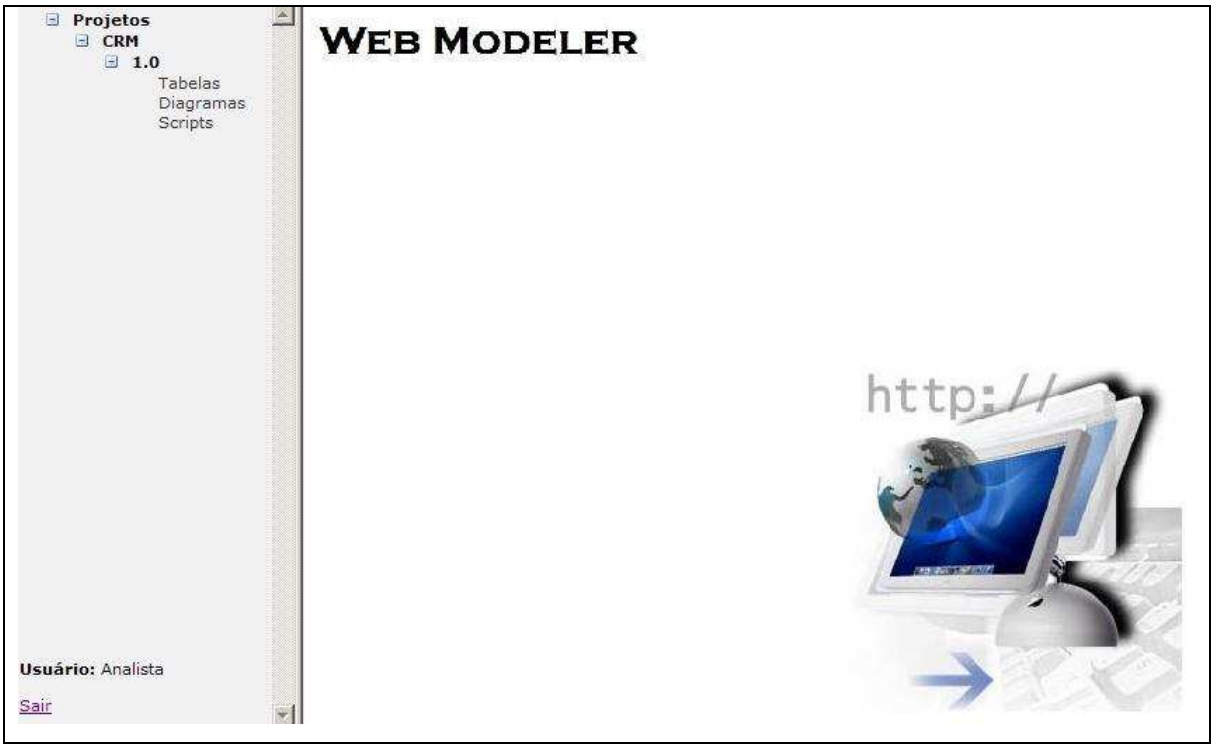


Figura 32 – Tela principal para o analista

O menu que aparece para os analistas traz todos os projetos aos quais os mesmos têm acesso e, para cada projeto, todas as versões que estão em aberto, isto é, as versões que ele pode alterar.

O item de menu Tabelas dá acesso à listagem das tabelas existentes em cada versão de cada projeto, tendo esta listagem o mesmo padrão das demais, isto é, pode-se alterar uma tabela clicando no ícone da coluna Abrir ou pode-se criar uma tabela acionando o *link* Adicionar. A tela de cadastro de tabelas pode ser vista na figura 33.

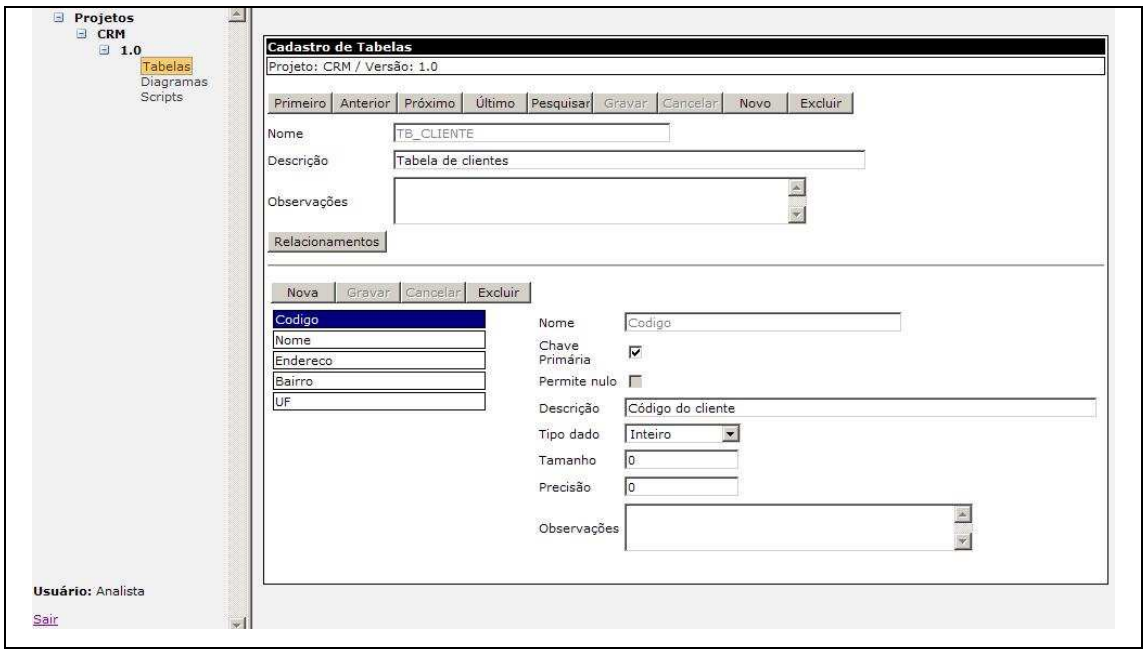


Figura 33 – Tela de cadastro de tabelas

Na parte superior desta tela devem ser informados o nome da tabela e a sua descrição (um detalhamento do seu objetivo). Abaixo destes dados aparece, na parte esquerda da tela, a lista das colunas desta tabela e, ao lado desta lista, os dados da coluna sobre a qual tiver sido acionado o botão esquerdo do *mouse* na lista. Além destas informações, esta tela também apresenta o botão Relacionamentos, o qual abre a tela de cadastro de chaves estrangeiras, exibida na figura 34.

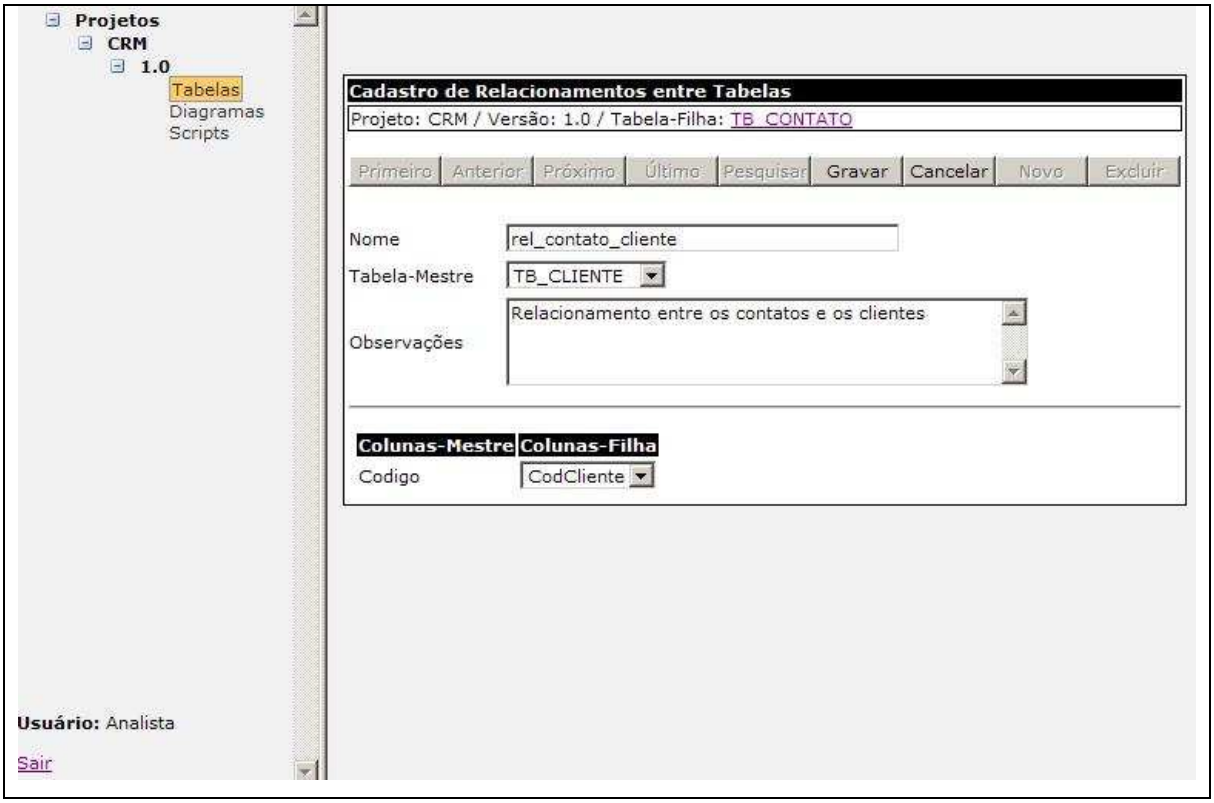


Figura 34 – Cadastro de relacionamentos entre tabelas

A tabela na qual a tela estava posicionada ao acionar o botão Relacionamentos é tratada como a tabela-filha no cadastro do relacionamento. Ao informar a tabela-mestre, as colunas da sua chave primária aparecem automaticamente na parte inferior da tela, sob o título Colunas-Mestre, sendo que para cada coluna da chave primária da tabela-mestre deve ser informada uma coluna da tabela-filha, coluna esta que deve ser selecionada no campo Colunas-Filha.

Além de cadastrar tabelas e relacionamentos da forma exposta anteriormente, o analista pode optar por fazê-lo de forma gráfica, modelando diagramas. O primeiro passo, neste caso, é cadastrar os diagramas que serão modelados, utilizando para tanto a opção de menu Diagramas da respectiva versão do projeto. Esta opção exibirá a listagem de diagramas já cadastrados, permitindo incluir novos diagramas ao clicar sobre o *link* Adicionar, o que abrirá a tela de cadastro de diagramas que é ilustrada na figura 35.

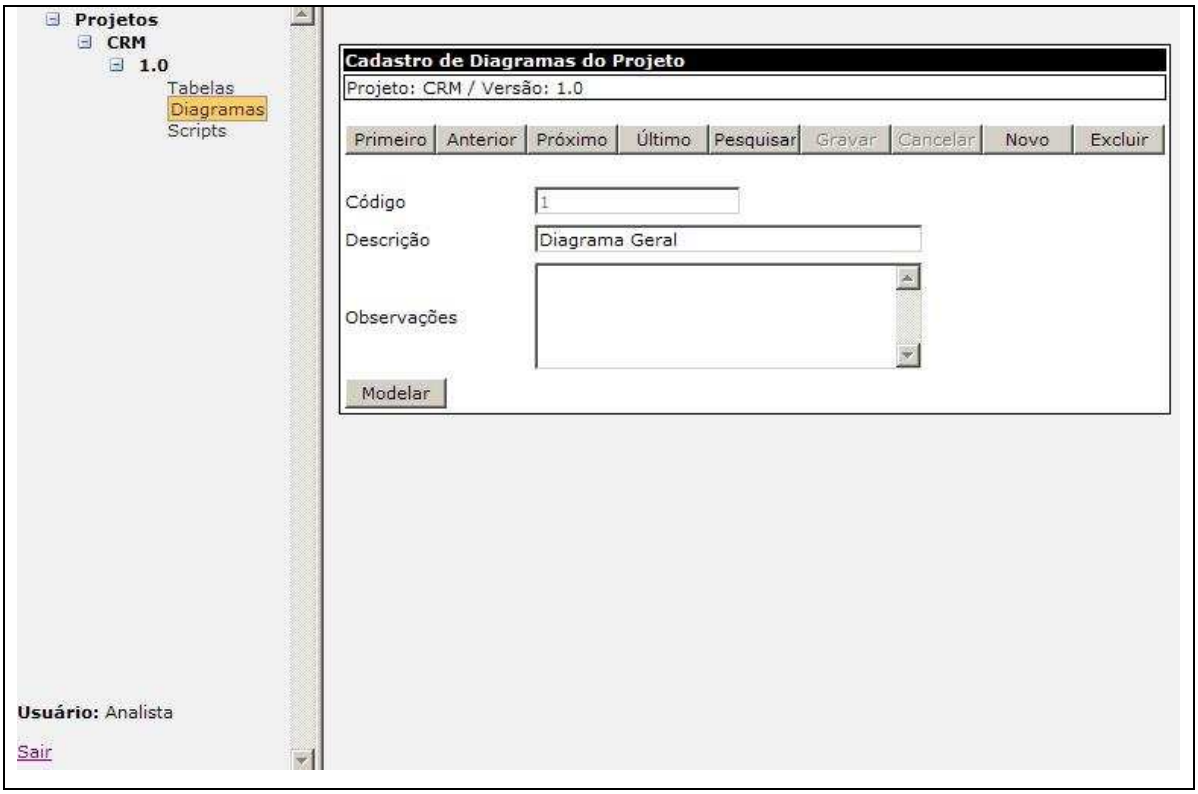


Figura 35 – Tela de cadastro de diagramas

Após gravar as informações do diagrama, ao acionar o botão Modelar o analista terá acesso à tela onde é feita a modelagem gráfica do banco de dados. Esta tela possui um quadro chamado Ferramentas, o qual possui três botões: um para adicionar tabelas ao diagrama, outro para salvar o diagrama e outro para descartar as alterações feitas após o último salvamento. A figura 36 mostra a tela de modelagem gráfica com uma tabela adicionada a partir do acionamento do primeiro botão da caixa de ferramentas.

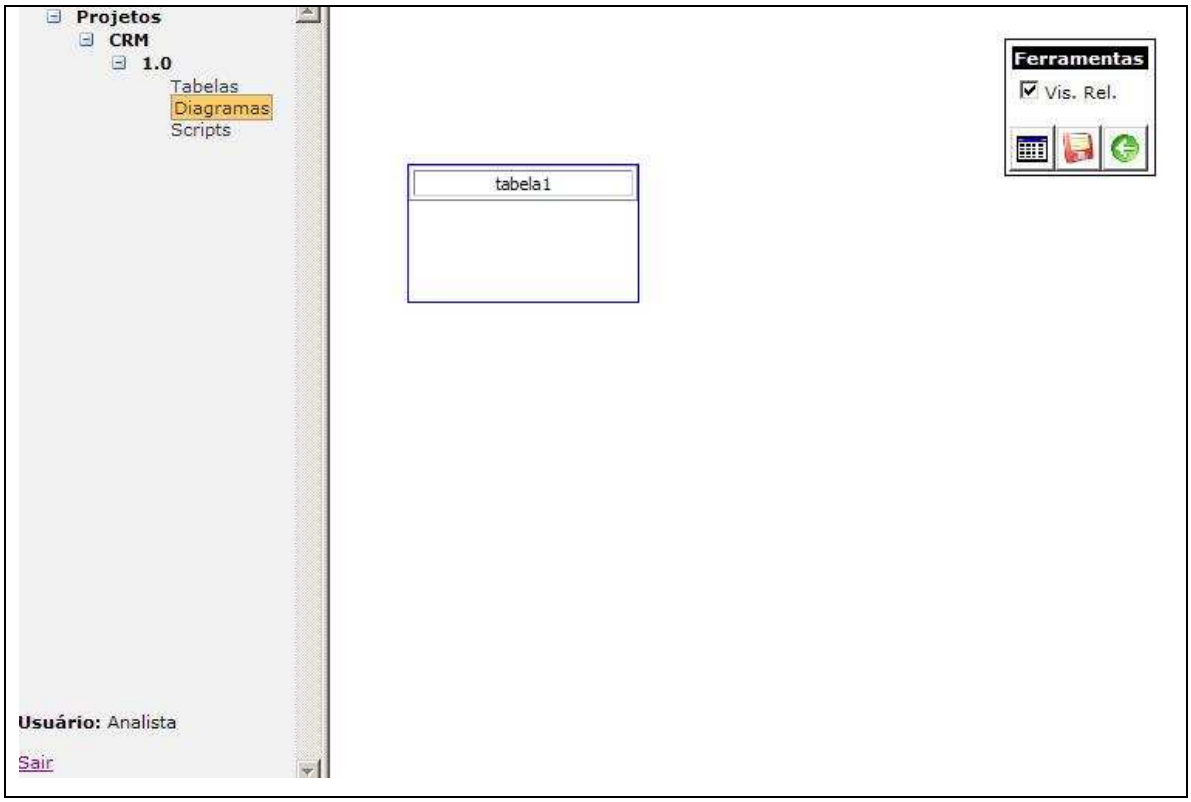


Figura 36 – Tela de modelagem gráfica

Após adicionar a tabela, sua estrutura deve ser definida pelo analista, o qual tem duas opções para acessar a tela de cadastro de tabelas para fazê-lo: dando um duplo clique sobre a tabela ou então através da opção Editar Tabela no menu de contexto que aparece ao acionar o botão direito do *mouse* sobre a tabela. Ambas opções fazem com que seja aberta a tela de cadastro de tabelas, onde o analista pode cadastrar uma nova tabela ou então abrir uma já existente e incluí-la no diagrama acionando o botão Adicionar. A figura 37 mostra o mesmo diagrama após a definição dos dados da tabela antes chamada tabela1.

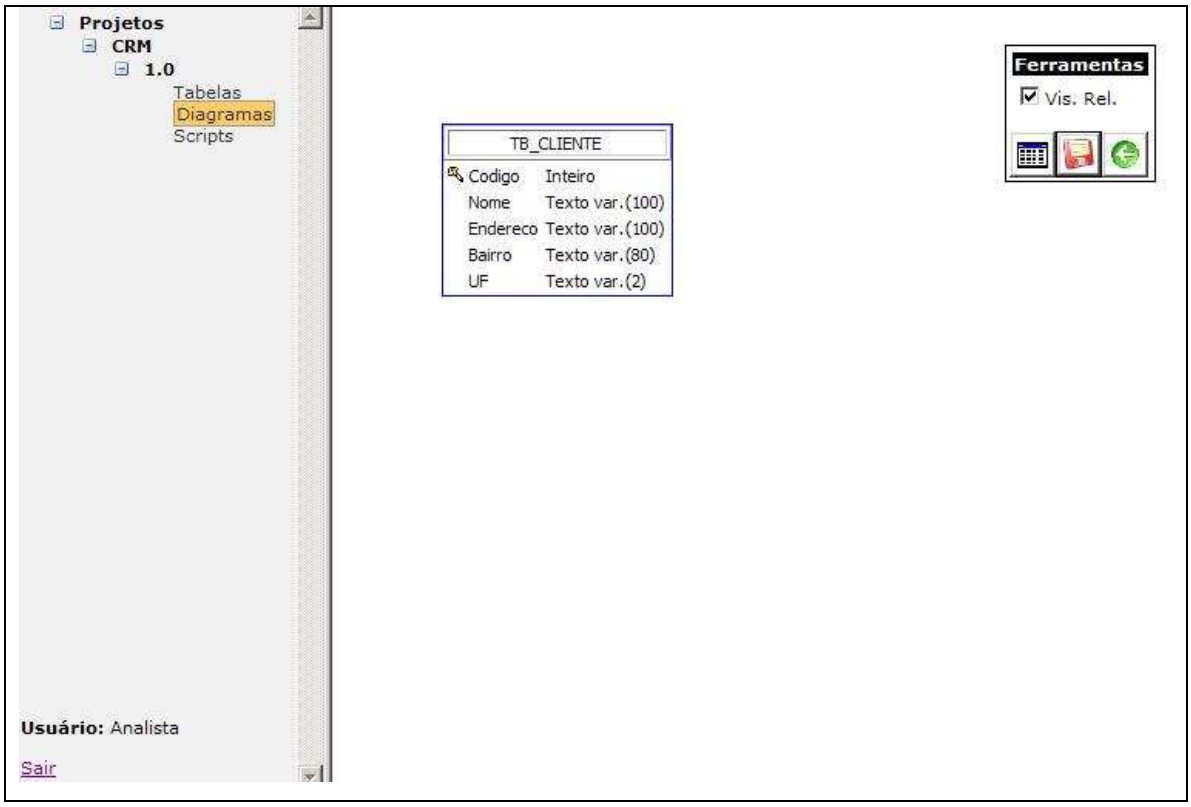


Figura 37 – Tela de modelagem gráfica com a tabela definida

Para adicionar relacionamentos entre duas tabelas, basta acionar o botão direito do *mouse* sobre a tabela que será a filha e clicar sobre a opção Relacionar. A tela de cadastro de relacionamentos será aberta e, após informar os dados do relacionamento e adicioná-lo ao diagrama, o mesmo ficará conforme exibido na figura 38.

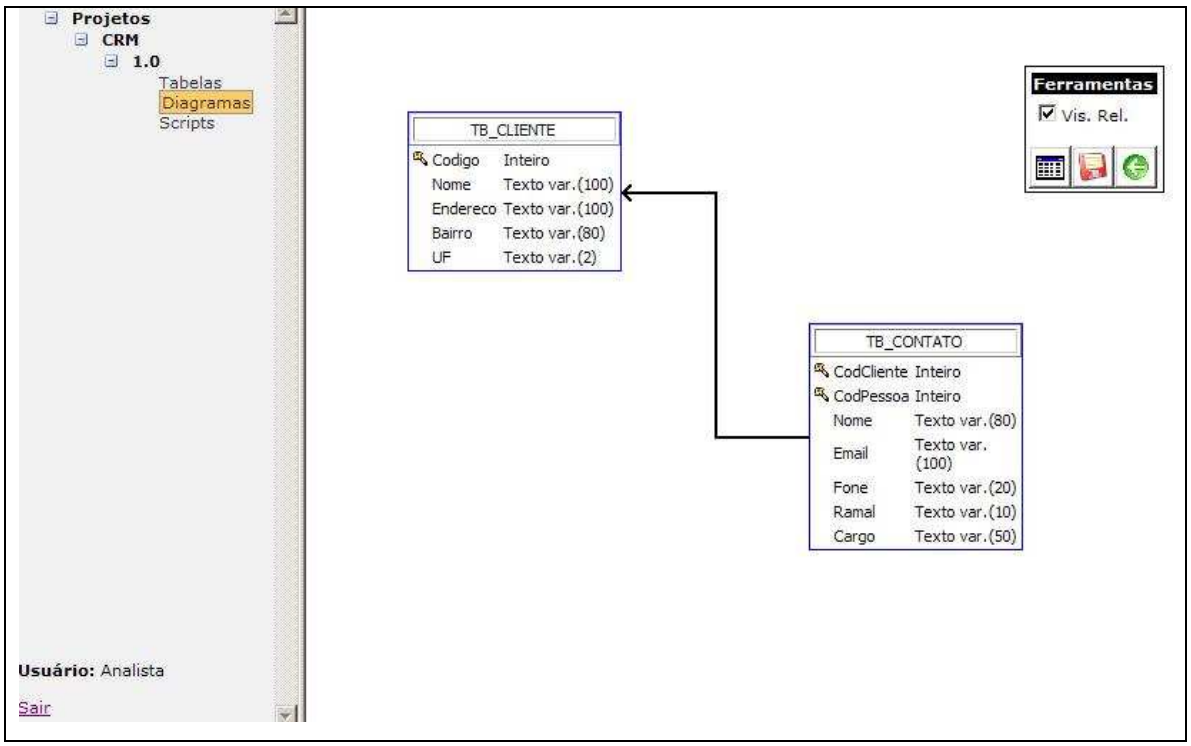


Figura 38 – Relacionamento entre tabelas na modelagem gráfica

Após concluir a modelagem da versão do projeto de banco de dados, o analista pode gerar os arquivos com os comandos SQL necessários à criação do banco de dados, através da opção de menu Scripts. Esta opção faz com que o sistema exiba a tela que pode ser vista na figura 39.

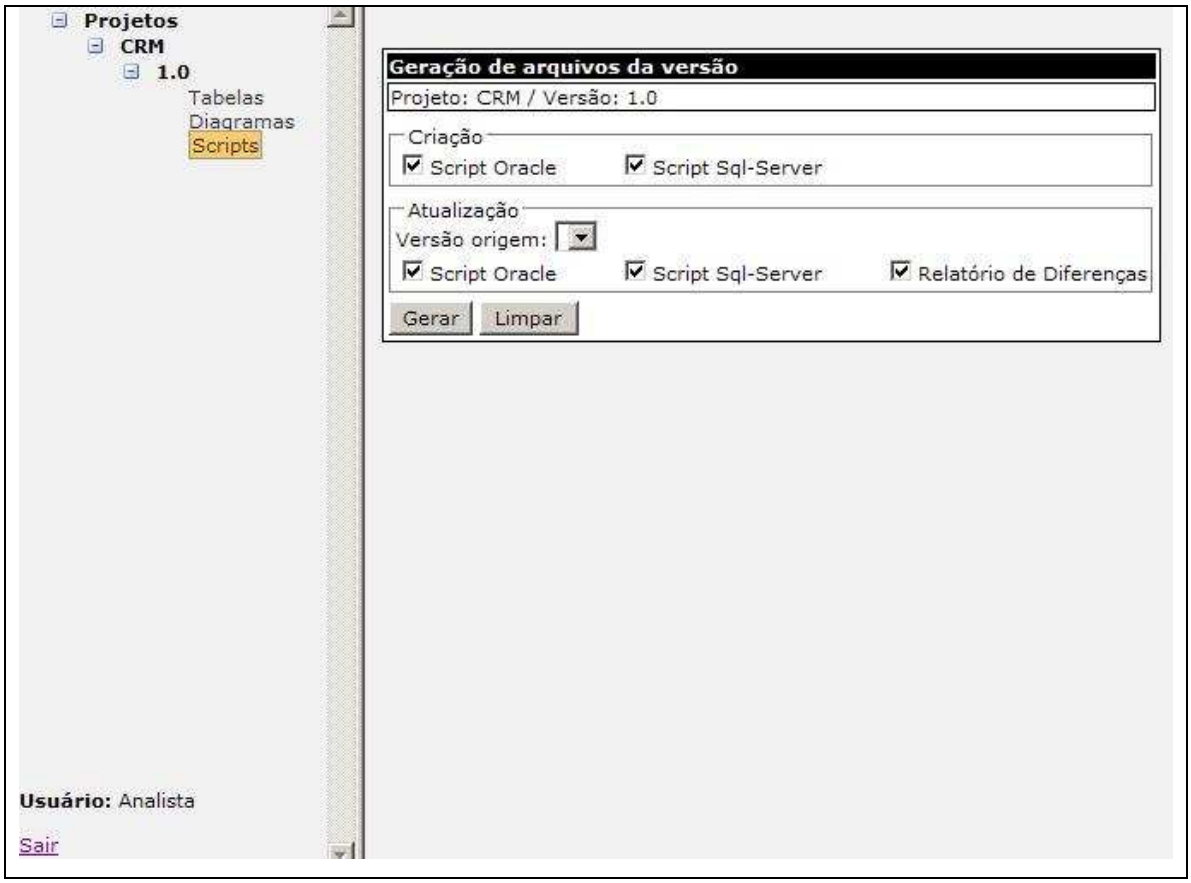


Figura 39 – Tela de geração de *scripts*

Nesta tela, o analista define os SGBD para os quais deverá ser feita a geração dos *scripts* de criação do banco de dados e, se for o caso, também os *scripts* de atualização a partir de uma versão anterior, bem como o relatório de diferenças entre estas versões. Tanto os *scripts* quanto o relatório são apresentados ao usuário no *browser* na forma de um arquivo PDF único. Em princípio, a geração deste arquivo encerra o processo de criação de uma versão, sendo que se o analista estiver de acordo, o passo seguinte é o fechamento da versão por parte do administrador do sistema, que deve fazê-lo na tela de cadastro da respectiva versão.

Nos apêndices A e B constam, respectivamente, os *scripts* gerados para a criação do banco de dados nos SGBD MSSqlServer e Oracle. Já os apêndices C e D trazem os *scripts* de atualização para a versão 2.0 e o apêndice E traz o relatório de diferenças entre as versões 1.0 (cujos passos da modelagem forem exibidos até aqui) e 2.0, sendo que esta última foi criada posteriormente com base na anterior, mas com as seguintes alterações:

a) inclusão da coluna Nascimento (do tipo Data) na tabela TB_CLIENTE;
b) exclusão da coluna Ramal da tabela TB_CONTATO;
c) inclusão da tabela TB_PRODUTO, tendo as colunas Código (do tipo Inteiro Longo), Nome (do tipo Texto Variável com 40 posições), Preco (do tipo Numérico, com 11 posições e precisão igual a 2) e Foto (do tipo Binário).

Desta forma, encerra-se a apresentação da operacionalidade da aplicação.

3.4 RESULTADOS E DISCUSSÃO

O software apresentado neste trabalho permite a modelagem de bancos de dados relacionais no ambiente web, possibilitando o cadastramento de projetos e suas versões, bem como as tabelas e relacionamentos existentes em cada versão. Essa modelagem pode ser feita, inclusive, de forma gráfica, como ocorre em diversas ferramentas CASE disponíveis no mercado. É possível ainda gerar *scripts* para a criação do banco de dados nos SGBD Oracle e MSSqlServer e, além disso, graças ao controle de versões, também é possível gerar *scripts* de atualização de bancos de dados.

Com relação às ferramentas CASE existentes no mercado, o software implementado tem algumas limitações, principalmente na questão da modelagem gráfica, pois não há recursos como impressão e *zoom* do diagrama, além de recursos para formatação das tabelas e relacionamentos. Muito da limitação da modelagem gráfica do WebModeler deve-se a atual falta de recursos dos *browsers* e do JavaScript em se tratando de criação de elementos gráficos (retas, figuras geométricas, etc.). Além disso, a falta de experiência com desenvolvimento web também impediu que fossem criados recursos mais avançados no aplicativo. Em contrapartida, o WebModeler oferece vantagens como o acesso remoto (por tratar-se de um aplicativo web), controle de projetos e versões, cadastro de usuários e restrições de acesso por meio do relacionamento entre projetos e usuários.

As figuras 40 e 41 foram incluídas a seguir para ilustrar a semelhança na modelagem de um banco de dados no software ora apresentado e numa ferramenta CASE já existente (no caso, o PowerDesigner). O mesmo banco de dados foi modelado em ambos os aplicativos.

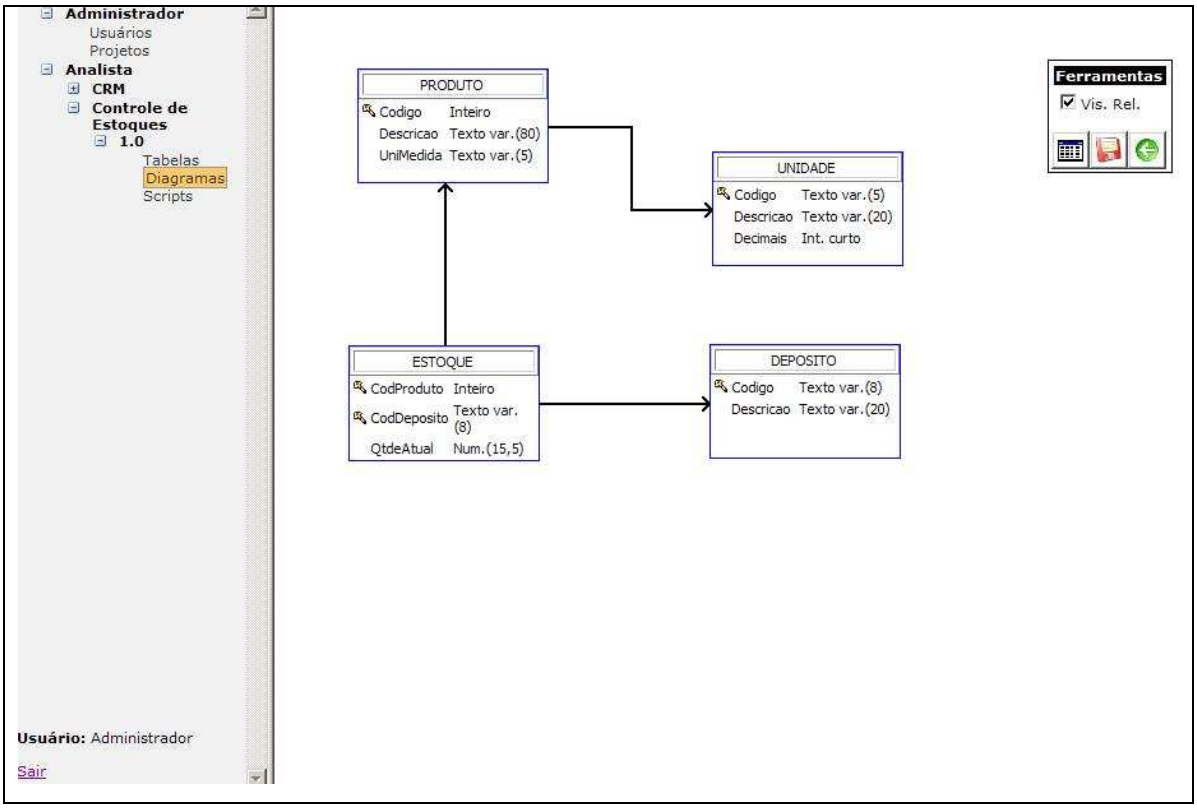


Figura 40 – Modelagem no WebModeler

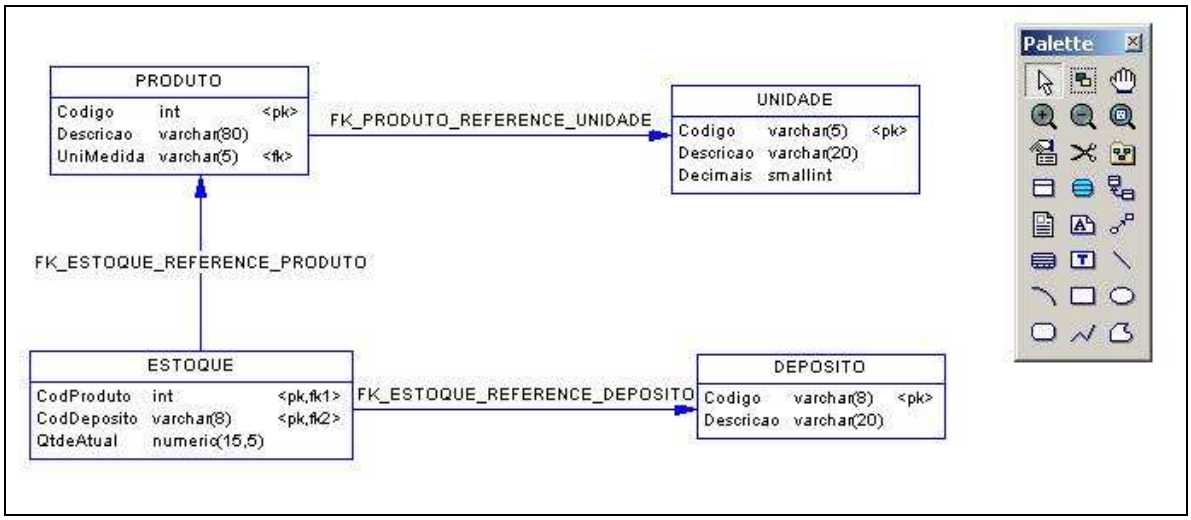


Figura 41 – Modelagem no PowerDesigner

Já com relação ao trabalho de Bernardi (1997), o software atual apresenta inúmeros novos recursos. Além dos que diferenciam o WebModeler do PowerDesigner, pode-se citar também que o trabalho atual permite salvar os diagramas desenvolvidos, algo que não era possível no trabalho anterior, além de ser um aplicativo web, ao invés de *desktop*.

Em termos de desenvolvimento do aplicativo, as tecnologias, técnicas e ferramentas empregadas mostraram-se bastante eficientes. A utilização do Ajax mostrou-se um pouco complicada no início do desenvolvimento, mas com o passar do tempo e com a prática essa dificuldade diminuiu consideravelmente. Gois (2006) utilizou o *framework Google Web*

Toolkit (GWT) para disponibilizar os recursos do Ajax, porém, no trabalho atual, nenhum *framework* foi utilizado, isto é, todos os recursos foram criados utilizando única e exclusivamente JavaScript, DOM, CSS e o objeto `XMLHttpRequest`, o que, por um lado tornou a implementação mais demorada (por não ter classes e funções prontas, como no *framework*), mas em contrapartida oferece maior domínio sobre a lógica do funcionamento das rotinas. Além disso, em seu trabalho, Gois (2006) também utilizou gráficos para representar os mapas dos ambientes, sendo que para tanto utilizou a linguagem *Scalable Vector Graphics* (SVG). Porém, como o Internet Explorer não dá suporte nativamente a esta linguagem (que é suportada de forma nativa pelo Firefox, por exemplo), no presente trabalho os gráficos do módulo de modelagem de diagramas foram desenhados somente com elementos HTML padrão e JavaScript, algo que mais uma vez despendeu um pouco mais de tempo de desenvolvimento.

4 CONCLUSÕES

Ao findar deste trabalho, pode-se concluir que os seus objetivos iniciais foram alcançados com sucesso. O aplicativo web desenvolvido permite que um banco de dados relacional seja modelado, inclusive através de diagramas, de forma independente do SGBD onde será criado (Oracle ou MSSqlServer), pois os tipos de dados para as colunas são próprias do sistema e não nativas de algum SGBD específico. Essa independência de algum SGBD específico pode ser considerada importante principalmente porque diversas linguagens de programação modernas, como Delphi e Java, por exemplo, permitem a geração de programas cujo código-fonte é totalmente compatível com vários SGBDs relacionais, ou seja, não é necessária uma preocupação com o SGBD que será utilizado posteriormente e, com a ferramenta desenvolvida neste trabalho, também é possível criar o modelo do banco de dados sem esta preocupação.

Com relação à utilização conjunta das técnicas Ajax e da plataforma JEE não foi identificada nenhuma limitação que impedisse ou restringisse esta combinação. Foi possível comprovar que ambas são totalmente compatíveis, pois tanto as rotinas com utilização de Ajax para disponibilizar recursos de interface mais avançados quanto para comunicação assíncrona com o servidor foram desenvolvidas sem maiores problemas e tiveram funcionamento extremamente satisfatório. Dessa forma, foi possível disponibilizar na interface do sistema recursos como arrastar e soltar as tabelas na modelagem gráfica, exibir menus de contexto ao clicar com o botão direito do *mouse* e atualizar somente parte das páginas, principalmente nas telas de cadastro, onde somente os dados dos campos são atualizados através de comunicação assíncrona com o servidor ao invés de atualizar a página inteira sempre que novas informações são solicitadas.

De forma semelhante, a utilização da arquitetura MVC e dos demais padrões de projeto também mostrou ser bastante eficiente para separar claramente a interface das regras de negócio. Graças a esta arquitetura, por exemplo, é possível migrar este software para o ambiente *desktop* apenas criando uma nova camada de visão e alterando algumas classes da camada de controle, não sendo necessário reescrever toda a camada de modelo (regras de negócio), ou seja, além de facilitar a manutenção do sistema, esta arquitetura também possibilita uma grande reutilização de código.

As vantagens para os usuários do software desenvolvido em relação a outras ferramentas CASE são inúmeras. Em primeiro lugar, trata-se de um aplicativo web que utiliza

um banco de dados, ao invés de ser uma aplicação *desktop* onde os modelos são salvos em arquivos binários. Com isso, além de uma maior segurança no armazenamento dos modelos, também é facilitado o acesso de inúmeros usuários ao mesmo projeto, pois todos podem acessá-lo a qualquer hora e de qualquer lugar e trabalhar simultaneamente no mesmo, sem a necessidade de, por exemplo, enviar o arquivo binário com os modelos para o e-mail de outro usuário para que este proceda as suas alterações. É bem verdade que neste ponto o software possui uma pequena limitação, pois não impede que dois ou mais usuários alterem os dados da mesma coluna, da mesma tabela ou do mesmo relacionamento ao mesmo tempo, o que pode gerar alguns problemas.

Ainda com relação aos usuários, o software permite, além do cadastro destes, o relacionamento entre projetos e usuários, ou seja, a definição dos analistas que poderão acessar cada projeto, lembrando que o acesso ao sistema só é possível mediante informação de senha.

Outra vantagem deste software é o controle das versões dos projetos, com a possibilidade de se gerar *scripts* de criação e atualização de bases de dados para os SGBD Oracle e MSSqlServer, dois dos mais comuns SGBD relacionais do mercado atualmente.

Quanto ao mercado para utilização da ferramenta ora apresentada, percebeu-se que além dos fins comerciais (empresas desenvolvedoras de software) a mesma pode ser utilizada para fins didáticos, em especial em disciplinas que visam prover conhecimentos básicos a cerca de bancos de dados relacionas, projeto de bancos de dados e modelagem de sistemas.

4.1 EXTENSÕES

Várias sugestões podem ser dadas para trabalhos futuros baseados no atual. Inicialmente, podem ser melhorados os recursos já existentes, disponibilizando novas ferramentas para modelagem gráfica, como recursos de impressão de diagramas, aplicação de *zoom* na visualização e possibilidade de personalizar os relacionamentos, alterando suas posições e o seu formato. O módulo de geração de *scripts* também pode sofrer ajustes, principalmente para permitir a geração de arquivos compatíveis com outros SGBD relacionais disponíveis no mercado, tais como Interbase, MySql, PostgreeSql, entre outros.

Com relação a novos recursos, possivelmente haja duas linhas diferenciadas a seguir. Na linha exclusiva de modelagem de bancos de dados, pode ser desenvolvida uma rotina de

engenharia reversa, a qual se conectaria a um banco de dados já existente e importaria a sua estrutura para o sistema, inclusive gerando um diagrama inicial com as tabelas e relacionamentos deste banco. Pode ser implementado também o cadastro de índices nas tabelas, bem como a possibilidade de criação de tipos de dados para as colunas próprios do usuário (domínios). Uma integração entre o sistema e outras ferramentas CASE também poderia ser desenvolvida, onde o sistema geraria arquivos compatíveis com estas ferramentas e importaria arquivos gerados pelas mesmas.

A outra linha que poderia ser seguida diz respeito à área de modelagem de sistemas, onde o sistema poderia ser expandido para que além do modelo do banco de dados pudessem ser desenvolvidos diagramas da UML, como diagramas de caso de uso, de classes, atividades, etc.

REFERÊNCIAS BIBLIOGRÁFICAS

ASLESON, R.; SCHUTTA, N. T. **Fundamentos do Ajax**. Rio de Janeiro: Alta Books, 2006.

BATTISTI, J. **SQL Server 2000:** administração e desenvolvimento: curso completo. Rio de Janeiro: Acxel Books, 2001.

BERNARDI, Y. A. **Protótipo de uma ferramenta de entidade relacionamento (ER) para geração de código para banco de dados (Interbase).** 1997. 86 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

BOND, M. et al. **Aprenda J2EE em 21 dias**: com EJB, JSP, Servlets, JNDI, JDBC e XML. Tradução João Eduardo Nóbrega Tortello. São Paulo: Pearson Education do Brasil, 2003.

COUGO, P. **Modelagem conceitual e projeto de bancos de dados**. Rio de Janeiro: Campus, 1997.

CRANE, D.; PASCARELLO, E.; JAMES, D. **Ajax em ação.** Tradução Edson Furmankiewicz e Carlos Schafranski. São Paulo: Pearson Prentice Hall, 2007.

DATE, C. J. **Introdução a sistemas de bancos de dados**. Rio de Janeiro: Campus, 2000.

FORTES, D. Web 2.0. **Info Exame**, São Paulo, n. 243, p. 44-49, Jun. 2006.

GAMMA, E. et al. **Padrões de projeto:** soluções reutilizáveis de software orientado a objetos. Tradução Luiz A. Meirelles Salgado. Porto Alegre: Bookman, 2000.

GOIS, M. V. S. **Ajax na construção de uma aplicação web para monitoramento de ambientes em plantas 2D.** 2006. 72 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

HEUSER, C. A. **Projeto de banco de dados**. 3. ed. Porto Alegre: Sagra Luzzato, 2000.

JAKARTA PROJECT. **Taglibs.** [S.l.], 2004. Disponível em: <http://jakarta.apache.org/taglibs/index.html>. Acesso em: 14 maio 2007.

KURNIAWAN, B. **Java para a web com servlets, JSP e EJB**: um guia do programador para soluções escalonáveis em J2EE. Rio de Janeiro: Ciência Moderna Ltda., 2002.

LOWAGIE. **iText Homepage.** [S.l.], 2007. Disponível em: <http://www.lowagie.com/iText/>. Acesso em: 16 maio 2007.

ORACLE. **Oracle Database SQL Reference 10g Release 2:** Datatypes. [S.l.], 2005. Disponível em: <http://download-east.oracle.com/docs/cd/B19306_01/server.102/b14200/sql_elements001.htm#i54330/>. Acesso em: 24 maio 2007.

SULLIVAN, Sean. **Advanced DAO programming**. [S.l.], 2003. Disponível em: <http://www-128.ibm.com/developerworks/library/j-dao/>. Acesso em: 30 abr. 2007.

SUN MICROSYSTEMS. **Core J2EE Patterns:** Front Controller. [S.l.], 2002a. Disponível em: <http://java.sun.com/blueprints/corej2eepatterns/Patterns/FrontController.html>**.** Acesso em: 24 abr. 2007.

_____. **Front Controller**. [S.l.], 2002b. Disponível em: <http://java.sun.com/blueprints/patterns/FrontController.html>**.** Acesso em: 24 abr. 2007.

_____. **JavaServer Pages Standard Tag Library**. [S.l.], 2007a. Disponível em: <http://java.sun.com/products/jsp/jstl/reference/docs/index.html>**.** Acesso em: 14 maio 2007.

_____. **JSTL Reference**. [S.l.], 2007b. Disponível em: <http://java.sun.com/products/jsp/jstl/1.1/docs/tlddocs/index.html>**.** Acesso em: 14 maio 2007.

VALE, C. R. **Criptografia MD5.** [S.l.], 2007. Disponível em: <http://www.devmedia.com.br/articles/viewcomp.asp?comp=2944>. Acesso em: 29 maio 2007

APÊNDICE A – *Script* de criação do banco de dados para MSSqlServer

```
/*
Script de criação do banco de dados para SQLServer
Projeto: CRM
Versão: 1.0
Data: 25/06/2007 19:28:36
*/

create table TB_CLIENTE (
Codigo int not null,
Nome varchar(100) not null,
Endereco varchar(100) not null,
Bairro varchar(80) not null,
UF varchar(2) not null,
constraint CP_TB_CLIENTE primary key (Codigo)) ;

create table TB_CONTATO (
CodCliente int not null,
CodPessoa int not null,
Nome varchar(80) not null,
Email varchar(100),
Fone varchar(20),
Ramal varchar(10),
Cargo varchar(50),
constraint CP_TB_CONTATO primary key (CodCliente,CodPessoa)) ;

alter table TB_CONTATO
add constraint rel_con_cli foreign key (CodCliente)
references TB_CLIENTE (Codigo);
```

APÊNDICE B – *Script* de criação do banco de dados para Oracle

```
/*
Script de criação do banco de dados para Oracle
Projeto: CRM
Versão: 1.0
Data: 25/06/2007 19:28:36
*/

create table TB_CLIENTE (
Codigo integer not null,
Nome varchar2(100) not null,
Endereco varchar2(100) not null,
Bairro varchar2(80) not null,
UF varchar2(2) not null,
constraint CP_TB_CLIENTE primary key (Codigo)) ;

create table TB_CONTATO (
CodCliente integer not null,
CodPessoa integer not null,
Nome varchar2(80) not null,
Email varchar2(100),
Fone varchar2(20),
Ramal varchar2(10),
Cargo varchar2(50),
constraint CP_TB_CONTATO primary key (CodCliente,CodPessoa)) ;

alter table TB_CONTATO
add constraint rel_con_cli foreign key (CodCliente)
references TB_CLIENTE (Codigo);
```

APÊNDICE C – *Script* de atualização para MSSqlServer

```
/*
Script de atualização de versão do banco de dados para SQLServer
Projeto: CRM
Versão anterior: 1.0
Nova versão: 2.0
Data: 25/06/2007 19:39:22
*/

alter table TB_CONTATO
drop constraint rel_con_cli;

create table TB_PRODUTO (
Codigo bigint not null,
Nome varchar(40) not null,
Preco decimal(11,2),
Foto binary(400),
constraint CP_TB_PRODUTO primary key (Codigo)) ;

alter table TB_CONTATO drop Ramal;

alter table TB_CLIENTE add Nascimento datetime not null ;

alter table TB_CONTATO
add constraint rel_con_cli foreign key (CodCliente)
references TB_CLIENTE (Codigo);
```

APÊNDICE D – *Script* de atualização para Oracle

```
/*
Script de atualização de versão do banco de dados para Oracle
Projeto: CRM
Versão anterior: 1.0
Nova versão: 2.0
Data: 25/06/2007 19:40:28
*/

alter table TB_CONTATO
drop constraint rel_con_cli;

create table TB_PRODUTO (
Codigo integer not null,
Nome varchar2(40) not null,
Preco decimal(11,2),
Foto blob,
constraint CP_TB_PRODUTO primary key (Codigo)) ;

alter table TB_CONTATO drop Ramal;

alter table TB_CLIENTE add Nascimento date not null ;

alter table TB_CONTATO
add constraint rel_con_cli foreign key (CodCliente)
references TB_CLIENTE (Codigo);
```

APÊNDICE E – Relatório de diferenças entre as versões

```
/*
Relatório de diferenças entre versões
Projeto: CRM
Versão anterior: 1.0
Nova versão: 2.0
Data: 25/06/2007 19:43:53
*/

* Tabelas adicionadas:
TB_PRODUTO - Cadastro de produtos

* Tabelas Alteradas:

Colunas adicionadas:
TB_CLIENTE.Nascimento - Data - não permite nulo

Colunas excluídas:
TB_CONTATO.Ramal - Texto Variável(10)
```