Tutorial MDArte

Primeiros passos do
framework MDArte

Maio 2014

Contents

Chapter 1

Preparação do Ambiente

Nesta seção detalharemos o processo de preparação do ambiente de desenvolvimento com o `AndroMDA`, onde serão enumeradas as ferramentas utilizadas e seus respectivos procedimentos de instalação.

Ferramentas necessárias:

- Máquina Virtual Java - JDK (Java 6)
- JBoss (versão 4.2.3-GA)
- Maven (versão 1.0.2)
- Magic Draw (versão 9.5)
- Eclipse Kepler (Java EE)

1.1 JDK - Java 6

É necessário que o `JDK` esteja instalado no computador. O download pode ser feito em `http://java.sun.com/` ou utilizando algum repositório, como mostrado abaixo:

```
sudo add-apt-repository ppa:webupd8team/java
sudo apt-get update
sudo apt-get install oracle-java6-installer
```

Assegure-se de baixar a `jdk` para o `java 6`. Alguns dos recursos utilizados pelo `MDArte` ainda não são compatíveis com o `java 7` e você poderá ter problemas se baixar a versão errada.

1.2 JBoss e Maven

Antes de instalarmos tais ferramentas é importante que entendamos um pouco do que elas são e qual a finalidade dela dentro do desenvolvimento com o `MDArte`.

1.2.1 Jboss

O `JBoss` é um servidor de aplicações baseado em `Java`. Um servidor de aplicações é um software que provê um ambiente completo para que outras aplicações sejam executadas dentro dele usando uma gama

de serviços provida pelo servidor de aplicações. No caso das aplicações desenvolvidas neste tutorial, será o servidor, por exemplo, que cuidará do acesso e conexões do sistema com o banco de dados.

A grande vantagem de um servidor de aplicações é que os desenvolvedores podem se concentrar nas necessidades de negócio. Aspectos como conexões a bancos de dados, autenticação e gerenciamento de recursos são gerenciados pelo servidor de aplicações.

Veremos como instalar a versão do `JBoss` compatível com o `MDArte` logo adiante.

1.2.2 Maven

O `Maven`[1] é uma ferramenta de automação e gerenciamento de projetos, gerenciando desde as dependências para compilação até a compilação e `deploy` da aplicação e tornando muito mais fácil a integração e utilizando de diversas ferramentas empregadas no processo de desenvolvimento de software.

A versão compatível atualmente é a 1.02. O `Maven`, durante sua execução, faz acesso a repositórios remotos, de onde poderão ser obtidos diversos artefatos necessários às tarefas de automação. Por exemplo bibliotecas (arquivos `*.jar`) necessárias para compilação e execução de um projeto podem ser automaticamente obtidas. Esses artefatos e bibliotecas externos, depêndencias do projeto, a serem obtidos e incorporados pelo `Maven` no momento da geração e compilação, são definidos nos arquivos `project.xml`. Um mesmo projeto pode conter vários `project.xml`, permitindo que possamos definir dependências específicas para modulos e pacotes diferentes do nosso sistema de forma independente, segundo as nossas necessidades.

O script disponibilizado pelo pacote de instalação do `MDArte` já faz todas as configurações necessárias para o uso do `Maven`, inclusive criação de variáveis de ambiente e configuração do repositório externo a ser usado. Caso queira saber como configurar a url do repositório ou mudar algum detalhe na configuração padrão visite o `apêndiceB`.

1.3 Varíaveis de Ambiente

Variáveis de ambiente são uma forma eficiente de influenciar o comportamento das aplicações rodando em um sistema `Linux`. A variável `Lang`, por exemplo, determina qual o idioma que os programas deverão usar para se comunicar com o usuário.Se seu `Linux` tiver sido instalado em inglês, a variável `Lang` provavelmente possuirá o valor `"en_US.UTF-8"` , por exemplo.

Variáveis de ambiente consistem de nomes os quais possuem valores definidos para si. Variáveis de ambiente não possuem restrições quanto ao seu formato, tudo o que for atribuido a ela será salvo como texto, sendo responsabilidade das aplicações que as usarão interpretar seu significado e seus dados.

O MDArte usa as seguintes variáveis de ambiente durante sua execução:

- `JAVA_HOME` - Define o caminho para a pasta onde o `Java` se encontra instalado;
- `MAVEN_HOME` - Define o caminho para a pasta onde o `Maven` se encontra instalado;
- `MAVEN_OPTS` - Parâmetros de lançamento para JVM no momento da execução do `Maven`;
- `JBOSS_HOME` - Define o caminho para a pasta onde o `JBoss` se encontra instalado;

[1] http://maven.apache.org/

Além disso, a variável PATH, responsável por definir os possíveis caminhos para um executável no terminal do Linux, precisa ser alterada para que possamos acionar o Maven simplesmente digitando o comando maven.

No entanto, você não precisa configurar nenhuma destas variáveis manualmente, uma vez que o pacote de instalação do MDArte também faz essa configuração.

1.3.1 Instalando ferramentas e configurando o ambiente

Para instalar e configurar tais ferramentas baixe o pacote de instalação do MDArte no seguinte repositorio.

Caso tenha baixado o pacote comprimido, extraia-o em alguma pasta e então execute o seguinte comando, dentro da pasta do instalador:

```
sh install.sh
```

O script de instalação fará uma série de perguntas, como no momento estamos instalando o ambiente do zero, responda a todas elas yes(y).

Feito isto, será necessário reiniciar a sessão do usuário para essas variáveis serem atualizadas no sistema ou utilizar o seguinte comando abaixo.

```
source ~/.bashrc
```

1.4 MDArte

O MDArte, na verdade, não é um aplicativo, mas sim um conjunto de bibliotecas de classes. Em nosso processo de desenvolvimento, utilizaremos o MDArte como um plugin do Maven. O Maven, por sua vez, possui um mecanismo próprio para obtenção de plugins. Através de parâmetros na linha de comando podemos especificar ao Maven qual plugin queremos instalar e ele se encarrega de buscar este plugin no(s) repositório(s) para o(s) qual(is) estiver configurado.

No caso do plugin do MDArte, o seguinte comando deve ser executado para a instalação (ao copiar o comando, verificar se foi copiado corretamente, inclusive os hifens):

```
maven plugin:download -DgroupId=andromda
-DartifactId=maven-andromdapp-plugin-coppetec -Dversion=3.1.1.3.4.19-RC9
```

Após a execução desse comando o Maven terá instalado o plugin do AndroMDA no cache local do usuário e tarefas referentes ao MDArte poderão ser executadas através do Maven.

Eventualmente, dependendo das tarefas executadas, o Maven poderá buscar outros artefatos nos repositórios, contudo isso será feito de forma transparente e automática.

1.5 MagicDraw

O download do MagicDraw pode ser feito em http://www.magicdraw.com.

O MagicDraw é uma ferramenta para modelagem em UML e é recomendada para uso com o MDArte devido a seu suporte a diagramas de atividade, utilizados pelo cartucho BPM4Struts. Ainda, para que os modelos sejam corretamente utilizados pelo MDArte eles deverão conter estereótipos específicos, disponíveis através de um profile fornecido com o MDArte, que será mostrado com mais detalhes na seção “Iniciando o projeto no MagicDraw”.

1.6 Eclipse

O download do Eclipse pode ser feito em `http://www.eclipse.org/`.

Durante a geração de um projeto, o MDArte gerará automaticamente os arquivos de configuração `.project` e `.classpath` de um projeto `Eclipse`. Esses arquivos podem ser usados diretamente para importação do projeto ao `Eclipse`. O `.classpath` é o arquivo onde será indicado as bibliotecas para o `eclipse` que serão utilizados pelo projeto. Assim, o eclipse saberá completar as informações automaticamente. Já o `.project` é uma descrição das opções do projeto.

Citation of Einstein paper [1].

Chapter 2

MDArte

O MDArte é um framework voltado para o desenvolvimento de sistemas que utiliza a abordagem dirigida a modelo (MDD) que implementa a arquitetura baseada em modelo (MDA[1]) definida pela Object Management Group (OMG[2]) em 2001.

No desenvolvimento orientado a modelos, os diagramas conceituais são utilizados não somente para documentar e especificar o sistema a ser desenvolvido, mas também como componentes para o desenvolvimento do mesmo. Esses modelos são utilizados para a geração do código da aplicação e assim se tornam artefatos do desenvolvimento.

O objetivo desse tipo de desenvolvimento é o aumento da produtividade (reutilizando os modelos), simplificação do processo do design da arquitetura do sistema que será desenvolvido devido às gerações realizadas utilizando modelos como entrada e o aumento da comunicação com a equipe de desenvolvimento, de análise e de domínio.

A arquitetura baseada em modelo (MDA) é uma iniciativa da OMG que tem como intuito definir um padrão de arquitetura para o MDD. Assim, esse padrão passa a ser seguido tanto pela comunidade quanto pela indústria de desenvolvimento.

O AndroMDA é um framework de código aberto que implementa as transformações definidas pelo MDA e é dividido em núcleo e plugins (cartuchos). O núcleo é responsável por realizar a leitura nos modelos e disponibilização de suas informações para os cartuchos. Já os cartuchos definem os artefatos que de fato deverão ser gerado para a aplicação a partir das informações modeladas. Cada cartucho é organizado de forma a agregar características de mesma tecnologia. O MDArte disponibiliza hoje alguns diferentes tipos de cartuchos: EJB, Hibernate, Java, JUnit e Struts.

O MDArte surgiu da necessidade de incorporar mais funcionalidades nas transformações realizadas pelo o framework AndroMDA[3] e voltado para o desenvolvimento de software para o governo brasileiro. Ele compreende de um conjunto de cartuchos com diversas soluções de projetos e tendo a possibilidade de agregar novos cartuchos dependendo das demandas do governo e da comunidade em geral.

[1] http://www.omg.org/mda/index.htm
[2] http://www.omg.org/
[3] http://www.andromda.org/

Chapter 3

Desenvolvendo o primeiro projeto com o MDArte

Neste capítulo iremos construir um projeto básico usando o MDArte. O projeto consistirá de um sistema web simples para administrar um ambiente acadêmico, onde poderemos cadastrar cursos e gerenciar suas informações, bem como inserir alunos, inscrevê-los nos cursos e gerenciar suas informações.

3.1 Criação de um Novo Projeto

O plugin do MDArte para o `Maven` já possui um procedimento parametrizado para criação de projetos, que funciona como um `wizard`, onde o usuário deve responder a perguntas. Através das respostas fornecidas, o MDArte direcionará a criação da estrutura básica e dos artefatos básicos de configuração de projetos. O procedimento para criação de um novo projeto é:

1. Abra o terminal (`command prompt`) e vá para o diretório onde se deseja criar o projeto. O projeto será gerado em um subdiretório do diretório escolhido com o mesmo nome da pasta definido na geração do projeto.

2. Digite o comando: `maven andromdapp:generate`

3. As perguntas devem ser respondidas de acordo com o projeto a ser desenvolvido. Abaixo as respostas adequadas para o exemplo desenvolvido neste capítulo (perguntas em negrito):

   Please enter your first and last name (i.e. Rodrigo Salvador):
   MDArte
   Please enter the name of your J2EE project (i.e. Sistema Academico):
   Sistema Academico
   Please enter the id for your J2EE project (i.e. sistemaacademico):
   sistemaacademico
   Please enter a version for your project (i.e. 1.0):
   1.0
   Please enter the base package name for your J2EE project (i.e. br.mdarte.exemplo.academico):
   br.mdarte.exemplo.academico

Would you like to enable security? (enter 'yes' or 'no')?
yes
Would you like to use oAuth (enter 'yes' or 'no') ?
no
Would you like to use MDArte's default Controle Acesso (enter 'yes' or 'no') ?
yes
Would you like to use modules (enter 'yes' or 'no')?
yes
Please enter the EJB version number (enter '2' or '3'):
3
Please enter the Struts version number (enter '1' or '2'):
2
Would you like to enable the JUnit support for general testing? (enter 'yes' or 'no')?
no
Please enter the database backend for the persistence layer: (enter 'hypersonic' or 'mysql' or 'oracle' or 'postgres')
postgres

4. Após receber as respostas, o MDArte criará um subdiretório onde será gerada a estrutura inicial do projeto. A partir desse momento chamaremos esse diretório de `<DiretorioProjeto>`.

5. Ainda no console, vá para o diretório onde está seu projeto: `<DiretorioProjeto>`.

6. Digite `maven`. Isto obrigará o `Maven` a obter todos os artefatos (por exemplo, bibliotecas) de que o projeto dependerá.

3.2 Controle de Acesso

3.2.1 Baixando o Controle de Acesso e configurando propriedades do projeto

Neste tutorial estaremos utilizando funcionalidades de controle de acesso, porém não é nosso propósito explorar suas funcionalidades. Assim, estaremos utilizando um projeto de controle de acesso desenvolvido pela comunidade do MDArte.

O projeto pode ser obtido a partir do repositório `Git` do MDArte. Por fim, edite também o arquivo `project.properties` do `ControleAcesso` para configurar o tipo de Banco de Dados a ser utilizado. O arquivo deverá ficar da seguinte maneira:

```
deployExploded=false

packDependencies=true

packWar=true
```

```
dataSource.name=controleacessoDS
dataSource=java:/${dataSource.name}

dataSource.driver.jar=${env.JBOSS_HOME}/server/default/lib/postgresql.jar
dataSource.driver.class=org.postgresql.Driver
dataSource.url=jdbc:postgresql://<url-de-acesso-ao-banco>
dataSource.user=
dataSource.password=
dataSource.sql.init=core/cd/target/schema-create.sql
dataSource.sql.drop=core/cd/target/schema-drop.sql
dataSource.sql.load=core/cd/target/db/create-dummy-load.sql

dataSource.sql.onError=continue


sql.mappings=PostgreSQL

hibernate.db.dialect=org.hibernate.dialect.PostgreSQLDialect

defaultHibernateGeneratorClass=sequence
```

Note que a propriedade `dataSource.name` está definida como `controleacessoDS`.

3.2.2 Adicionando configurações do Controle Acesso ao JBoss

Precisaremos criar um arquivo de configuração do Banco de Dados (ou editá-lo, caso ja exista um), localizado no diretório `$JBOSS_HOME/server/default/deploy/`. O nome do arquivo deve seguir a mesma formatação mencionada, terminando em `-ds.xml` (ex.: `aplicacoes-ds.xml`), podendo estar no mesmo arquivo com as configurações do projeto SistemaAcademico.

Exemplo:

```
<datasources>
      <local-tx-datasource>
            <jndi-name>controleacessoDS</jndi-name>
            <use-java-context>true</use-java-context>
            <connection-url>
                  jdbc:postgresql://127.0.0.1:5432/controleacesso
            </connection-url>
            <driver-class>org.postgresql.Driver</driver-class>
```

```
            <user-name>usuario</user-name>
            <password>senha</password>
            <!--<exception-sorter-class-name>
            org.jboss.resource.adapter.jdbc.vendor.
               OracleExceptionSorter
            </exception-sorter-class-name>-->
</local-tx-datasource>
</datasources>
```

Agora, adicionaremos as configurações de login para o Controle de Acesso no servidor `JBoss`. Para tal, editaremos o arquivo `login-config.xml`, no caminho `JBOSS_HOME/server/default/conf/`. Adicionaremos então as seguintes configurações ao fim do arquivo, antes da tag `<\policy>` :

```
<application-policy name="controleacesso">
      <authentication>
            <login-module code="org.jboss.security.
               ClientLoginModule"
                  flag="required">
                  <module-option name="multi-threaded">true</
                     module-option>
            </login-module>
            <login-module
                  code="br.gov.mdarte.controleacesso.accessControl.
                     LoginModuleImpl"
                  flag="required">
                  <module-option name="dsJndiName">java:/
                     controleacessoDS
                  </module-option>
                  <module-option name="unauthenticatedIdentity">
                     guest</module-option>
                  <module-option name="principalClass">
                     accessControl.PrincipalImpl
                  </module-option>
                  <module-option name="hashEncoding">hex</module-
                     option>
                  <module-option name="hashAlgorithm">md5</module-
                     option>
                  <module-option name="principalsQuery">
                        select SENHA from USUARIO where LOGIN=?
                  </module-option>
```

```
                    <module-option name="rolesQuery">
                          select pf_usr.pf_FK, 'Roles'
                          from usuario, pf_usr
                          where LOGIN=? AND usuario.ID = pf_usr.
                             usr_FK
                    </module-option>
              </login-module>
        </authentication>
</application-policy>
```

3.2.3 Compilando o Controle de Acesso

Agora, execute os seguinte comandos, na raiz do projeto `ControleAcesso`, para gerar, compilar e copiar os pacotes para o diretório `$JBOSS_HOME/server/default/deploy/`:

```
sh mavex.sh -a
```

3.2.4 Preparando o banco do Controle de Acesso

Nós vamos agora executar alguns `scripts` no banco de dados que usaremos para o `Controle de Acesso` a fim de prepará-lo para o nosso sistema.

Primeiro executaremos o script `schema-create.sql`, no caminho `<raiz-do-projeto-controleacess`

3.3 Modelando o nosso primeiro projeto

Vamos começar agora a modelar o nosso exemplo, mostrando o quão rápido e simples pode ser usar o MDArte e todo o seu poder de geração.

Para esta parte do tutorial usaremos o `MagicDraw`. Na barra de ferramentas do `MagicDraw`, clicaremos em `Open Project` e abriremos o xml do projeto, `SistemaAcademico.xml` no caminho `<DiretorioProjeto>/mda/src/uml/`.

3.3.1 Modelando a camada de domínio

Na camada de domínio, estarão as classes do domínio da aplicação. Elas serão entidades e estarão associadas a algum modo de persistência. Afim de contornar os problemas provenientes da utilização de bancos de dados relacionais em conjunto com o paradigma de orientação à objeto, o MDArte usa o `framework Hibernate`[1] para gerenciar esta camada.

As classes da camada de domínio deverão conter o estereótipo «`Entity`» e os atributos que serão persistidos. Todas as classes de entidade serão concentradas no pacote `<PacoteProjeto>.cd`, em

[1] http://hibernate.org/orm/

que <PacoteProjeto> é o pacote definido para o projeto. Note que não é obrigatório que as classes da camada de domínio estejam todas no pacote <PacoteProjeto>.cd, no entanto esta é uma boa prática e um padrão adotado pela comunidade do MDArte.

Neste exemplo, especificamente, iremos também marcar nossas entidades com o estereótipo «Manageable», tal marcação diz para o MDArte que desejamos que seja gerado um CRUD[2] padrão para tais entidades, sem a necessidade de modelarmos o mesmo diretamente.

1. Crie a mesma estrutura de pacotes que foi definida na criação do projeto. Dentro da estrutura, crie o pacote "cd". Podemos ver o resultado na imagem 3.1.

Figure 3.1: Criação da estrutura de pacotes do projeto e do pacote cd.

2. Clique com o botão direito do mouse no pacote "cd" e selecione a opção New Diagram .Em seguida, selecione Class Diagram. Como mostra a Figura 3.2.

Figure 3.2: Criação do diagrama de classes da camada de domínio.

3. Indique o nome desejado para o diagrama (ex: Entidades).

4. No diagrama de classe, crie uma nova classe. Clique com o botão direto sobre a classe e selecione a opção Specification. Defina o nome da classe como "Estudante".

5. Crie os atributos na classe Estudante (matricula, nome) selecionando a aba Attributes e clicando no botão Add. A figura abaixo exemplifica a criação do atributo matricula. O campo Visibility deve ser public. Não é necessário modelar o atributo id, pois ele é gerado automaticamente. Como mostra a Figura 3.3.

[2]CRUD é um acrônimo de Create, Read, Update e Delete em língua Inglesa para as quatro operações básicas utilizadas em bancos de dados relacionais ou em interface para usuários para criação, consulta, atualização e destruição de dados.

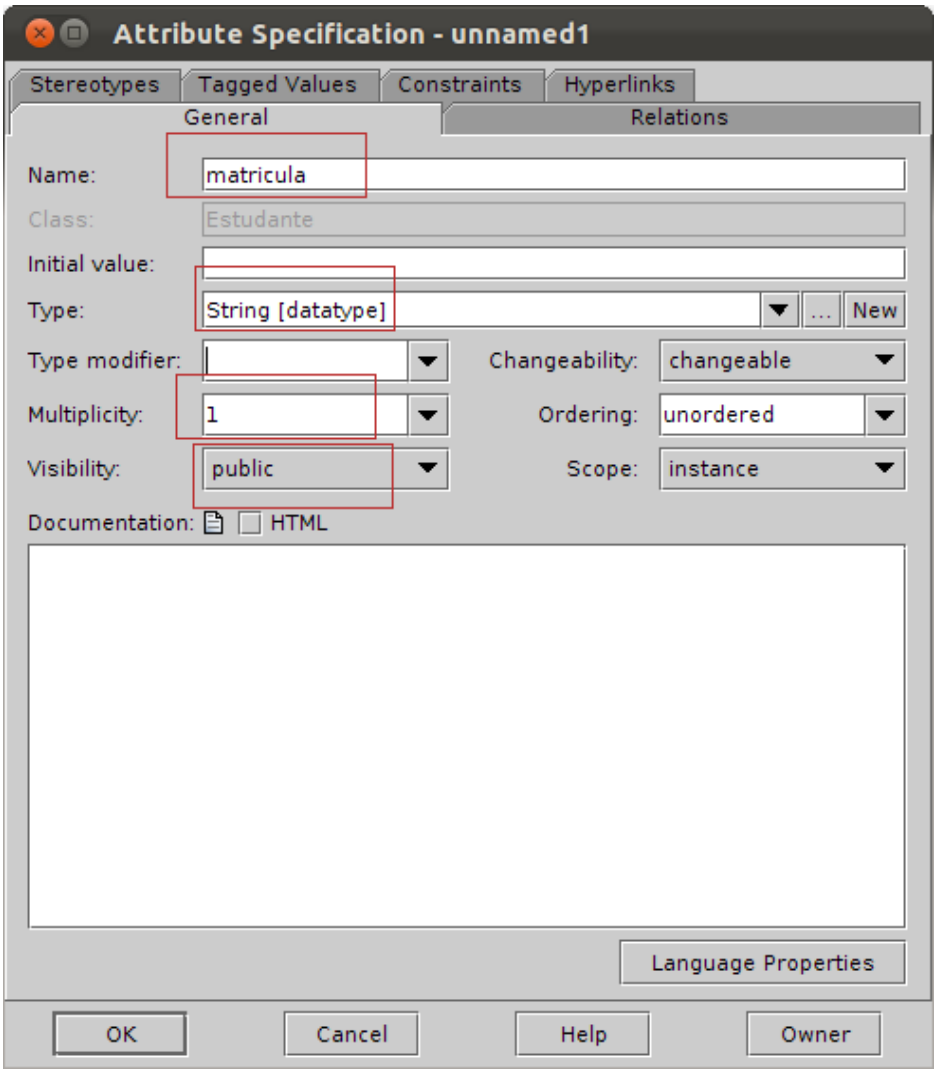

Figure 3.3: Configuração do parâmetro matrícula da classe Estudante.

A multiplicidade com valor 1 (campo `Multiplicity`) indica que o atributo é obrigatório (`NOT NULL`), já o valor `0..1` indica que o atributo não é obrigatório. Por padrão, todos os atributos são gerados como `NOT NULL`.

6. Coloque o estereótipo «`Unique`» no atributo `matricula` para indicar que cada código deve ser único, ou seja, não pode haver duas matrículas iguais. Abra a especificação do atributo `matricula` e selecione a aba `Stereotypes`. Nessa aba selecione o estereótipo «`Unique`», como na figura 3.4.

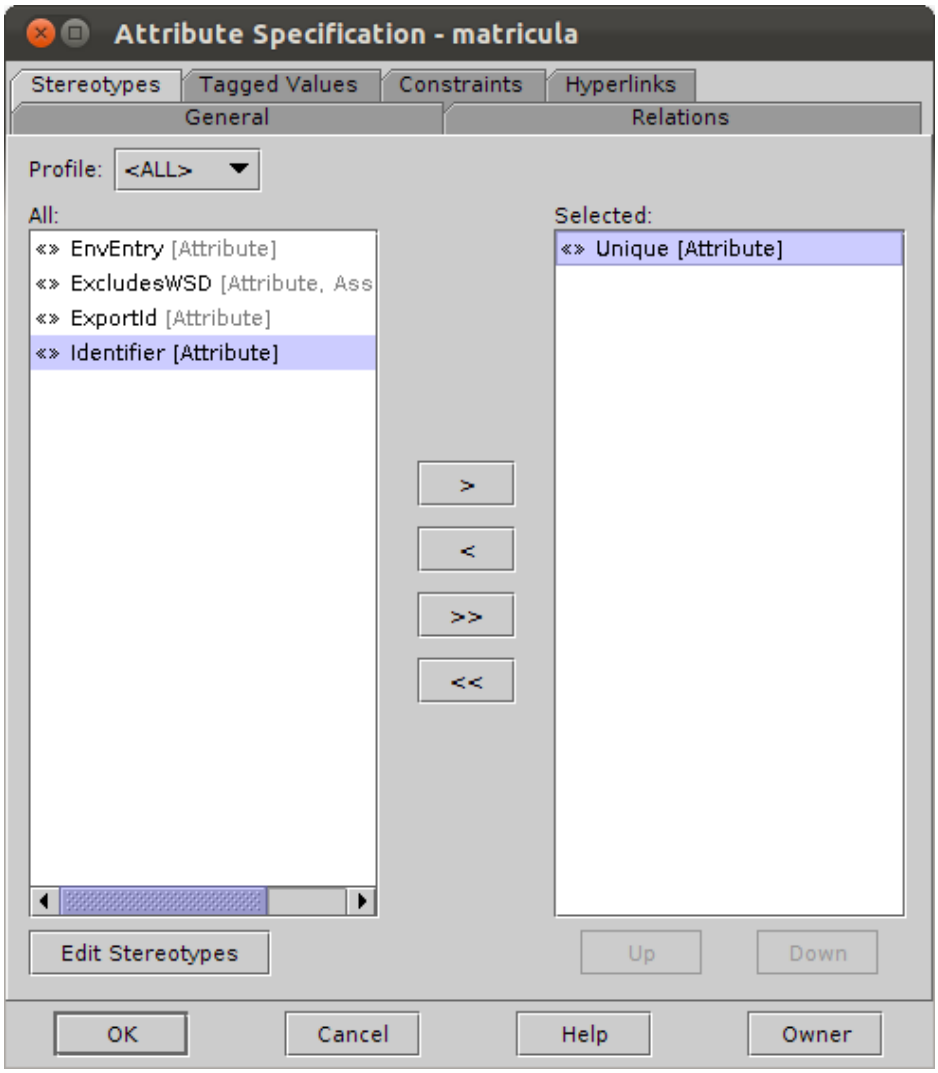

Figure 3.4: Adição de estereótipo no atributo matrícula.

7. Coloque os estereótipos «`Entity`» e «`Manageable`» na classe `Estudante`, como na figura 3.5.

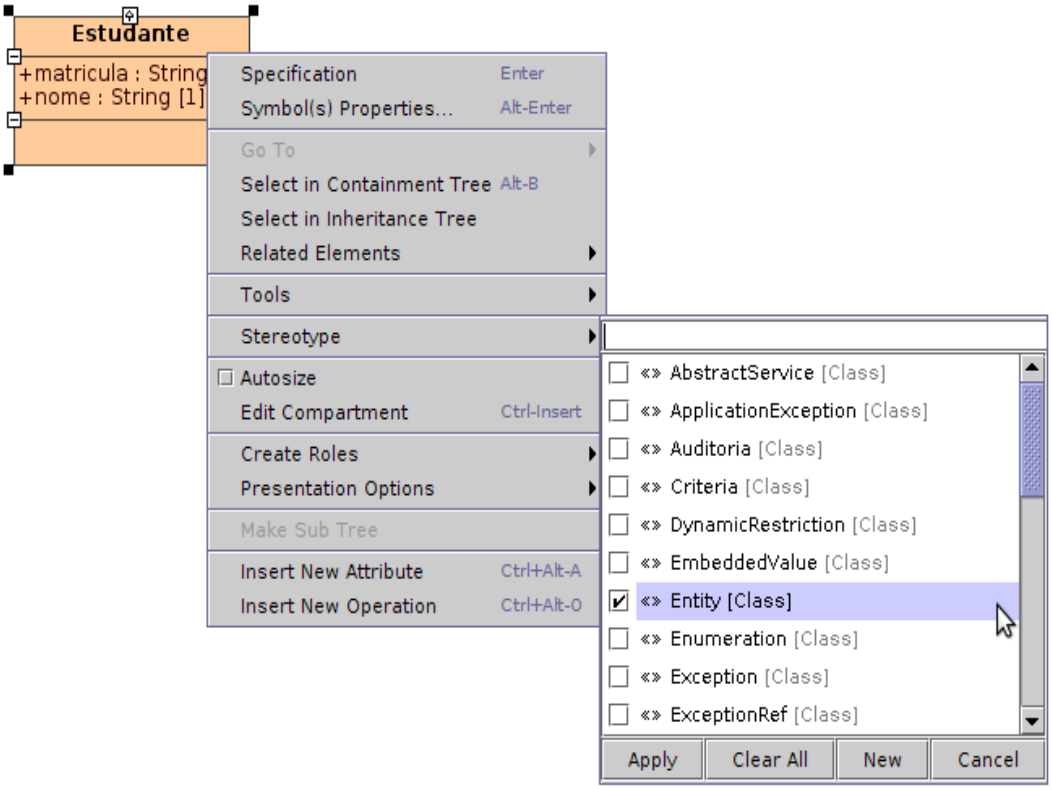

Figure 3.5: Adição dos estereótipos Entity e Manageable na classe estudante.

8. No mesmo diagrama de classes, crie outra classe. Clique com o botão direto sobre a classe e selecione a opção `Specification`. Defina o nome da classe como "`Curso`".

9. Crie os atributos na classe `Curso` (`codigo`, `nome`) selecionando a aba `Attributes` e clicando no botão `Add`. O campo `Visibility` deve ser `public`, assim como feito anteriormente.

10. Coloque o estereótipo «`Unique`» no atributo codigo para indicar que cada código deve ser único. Abra a especificação do atributo codigo e selecione a aba `Stereotypes`. Nessa aba selecione o estereótipo «`Unique`».

11. Coloque os estereótipos «`Entity`» e «`Manageable`» na classe.

12. Agora, crie uma associação entre as classes. Vá no diagrama de classes e puxe uma relação `Association` de uma classe para outra.

13. A associação será de `1 para muitos`. Assim, clique duas vezes na associação e irá aparecer a tela de especificação. Edite os campos `Multiplicity` definindo valor "0..*" para a entidade `Estudante` e "1" para a entidade `Curso`, como na figura 3.6.

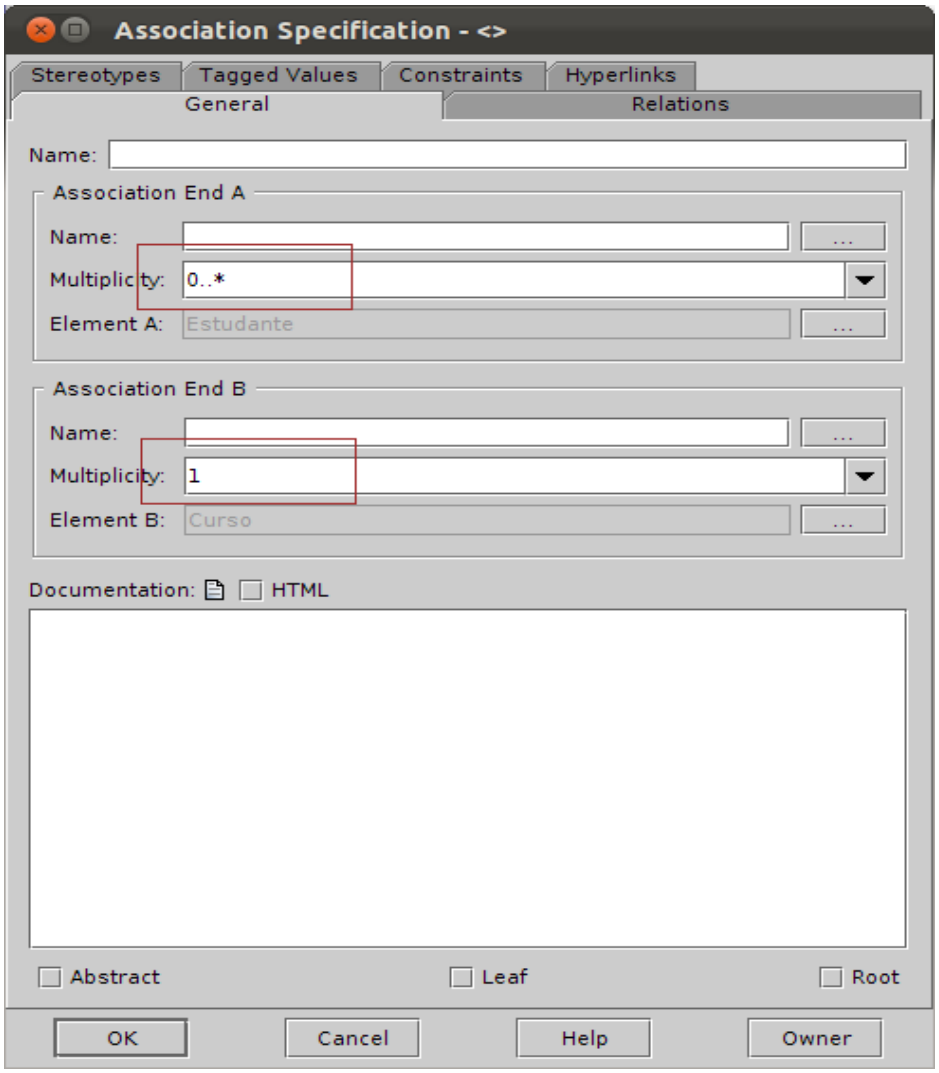

Figure 3.6: Definindo multiplicidade da associação.

14. A associação deve ser dupla, tanto `Estudant` e quanto `Curso` devem conseguir se enxergar, além disso sua `Visibility` deve ser `public`. Mantenha a `checkbox` `Navigable` marcada na associação para as duas classes. Para isso, clique no botão "..." (reticências) da tela anterior, como na imagem 3.7.

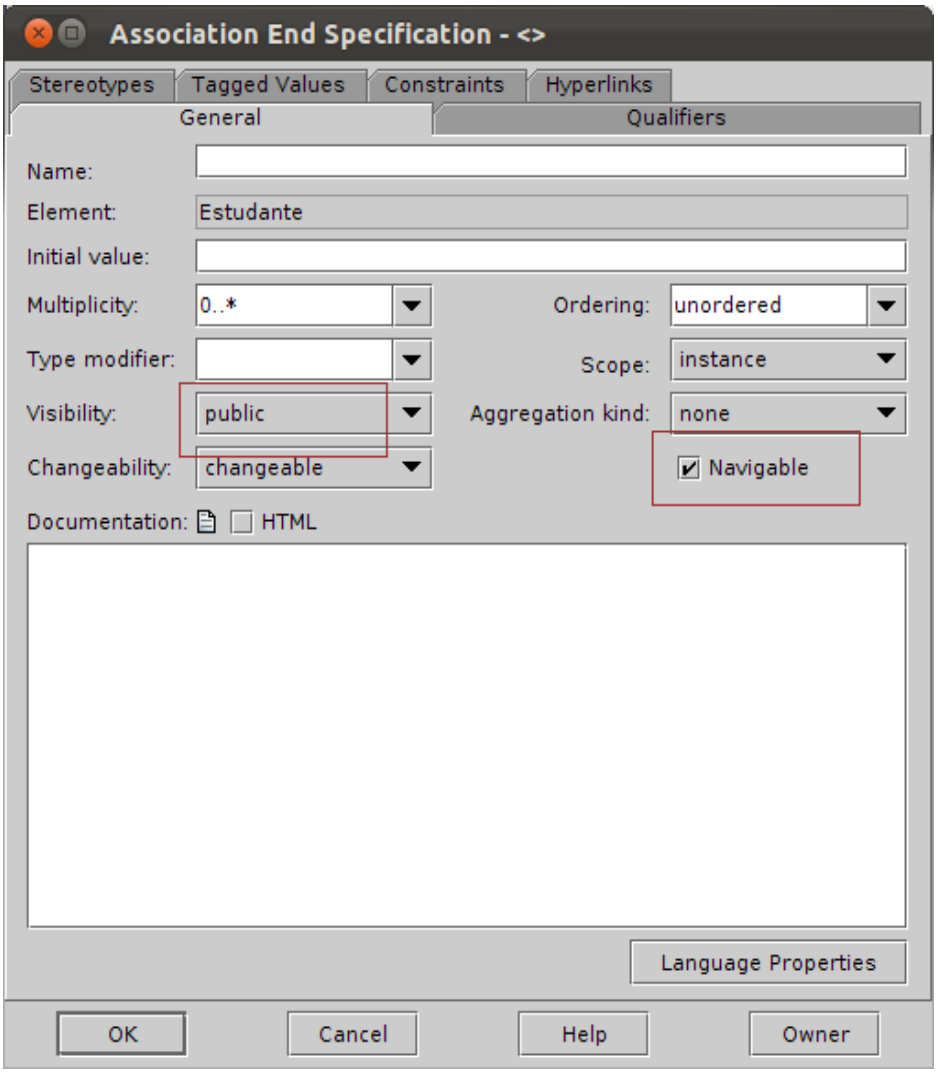

Figure 3.7: Configuração da navegabilidade da associação.

O resultado final pode ser visto na imagem 3.8:

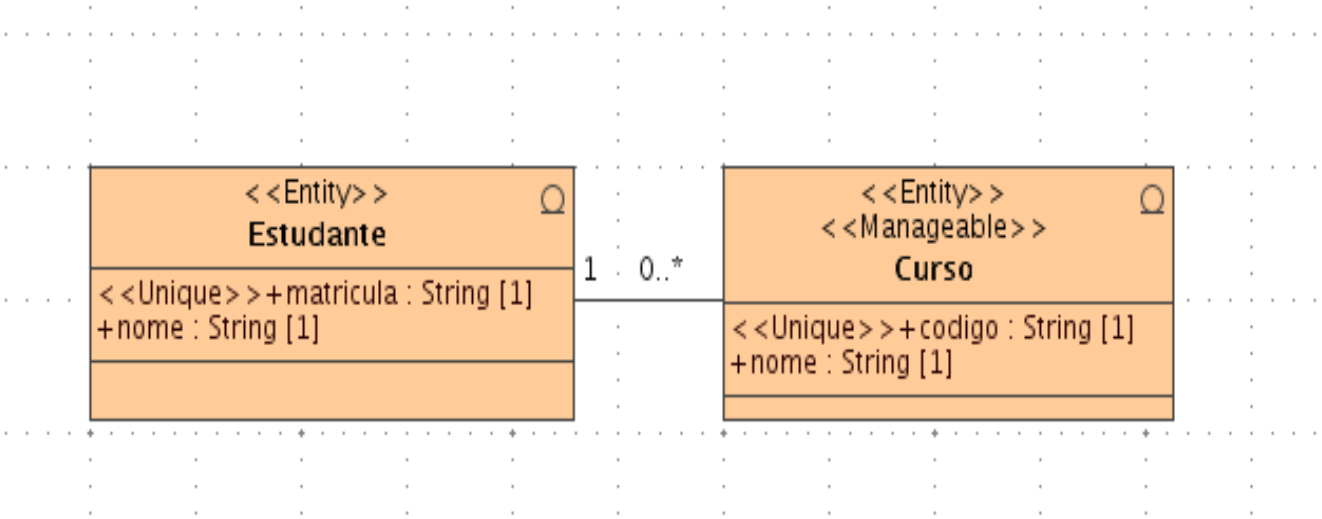

Figure 3.8: Resultado final da modelagem da camada de domínio.

15. No diretório da aplicação, execute o comando maven para validar o modelo e gerar o script `SQL` de criação do Banco de Dados. O resultado apresentado deve ser "`BUILD SUCCESFULL`".

16. Observe que dois novos arquivos `xml` terão sido criados no caminho `<DiretorioProjeto>/mda/src/uml/` com os nomes `sistemaacademico-geral-Curso.xml` e `sistemaacademico-geral-Estudante.xml`, com os `CRUD default` gerados pelo MDArte. Agora precisamos importar os casos de uso criados pelo MDArte para o `xml` geral do nosso projeto (`SistemaAcademico.xml`). Para isso iremos na barra de ferramentas do `MagicDraw` clicaremos em `file` e na lista de opções que se abrirá clicaremos em `import`. Na janela que se abrirá selecionaremos os modelos que queremos adicionar e clicaremos em `open`, como na imagem 3.9:

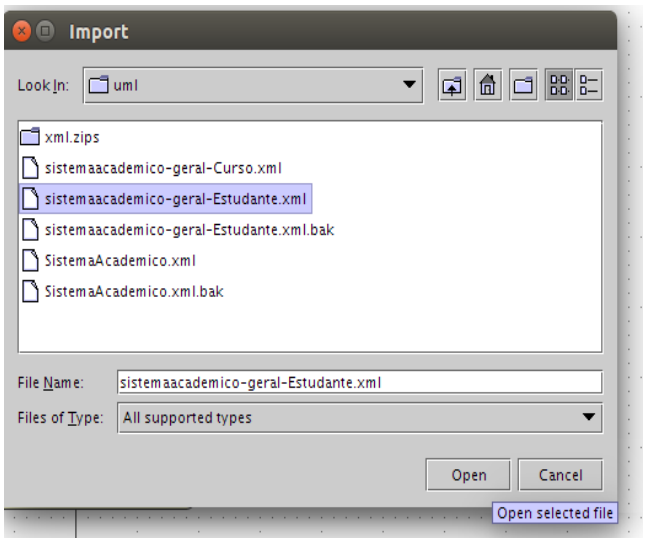

Figure 3.9: Criação do pacote para a camada de serviço.

O resultado da importação na nossa estrutura de diretórios pode ser visto na imagem 3.10.

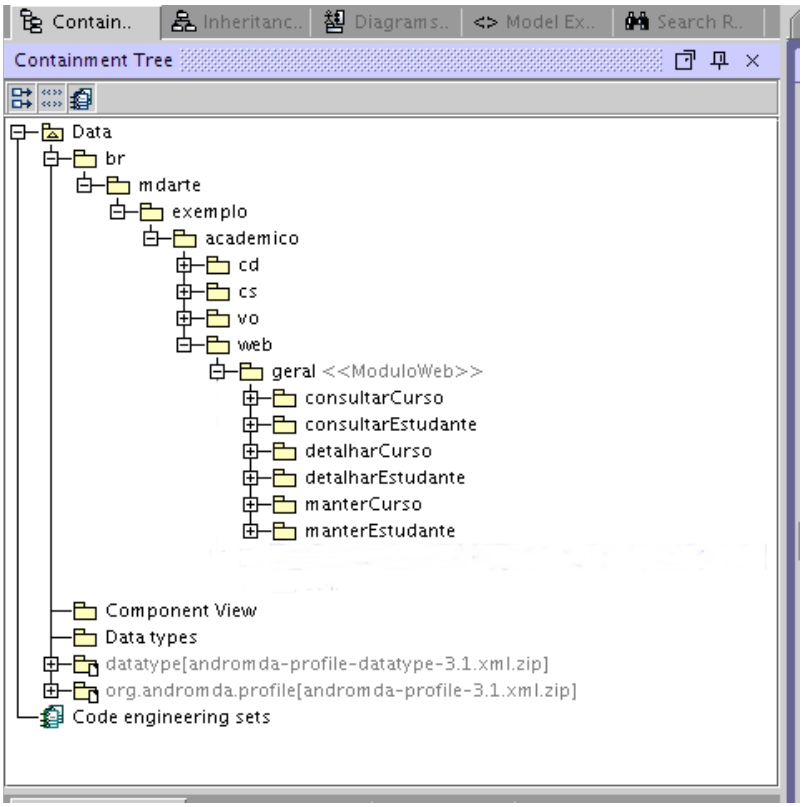

Figure 3.10: Arvore de diretórios do projeto após a importação dos CRUD gerados automaticamente.

17. Agora, ainda no `MagicDraw`, alteraremos o modelo, adicionando ao pacote `'geral'` o estereótipo «`ModuloWebPrincipal`», o resultado pode ser visto na imagem 3.11.

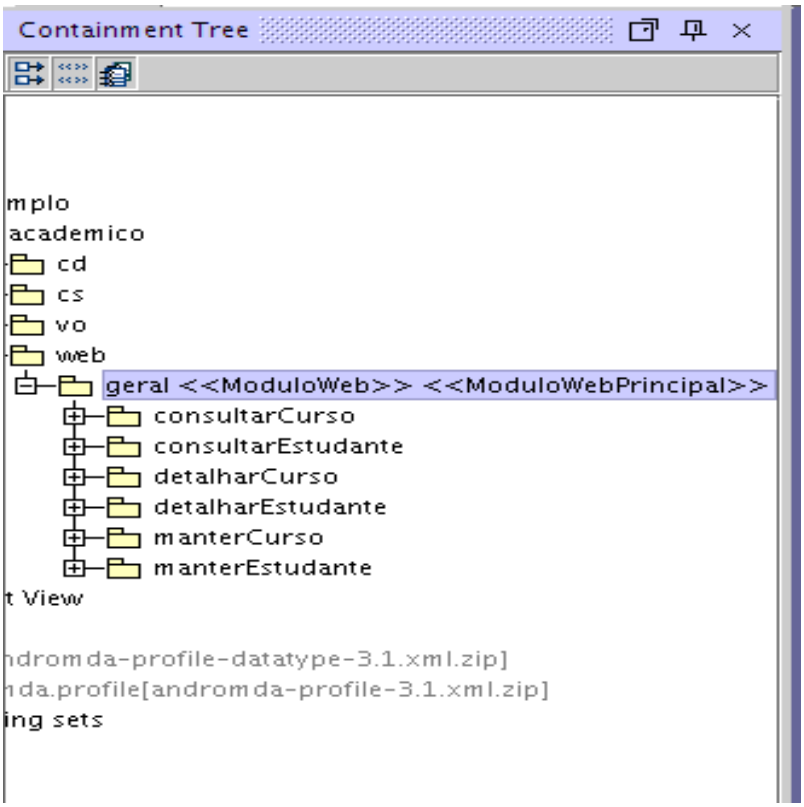

Figure 3.11: Arvore de diretórios do projeto após a importação dos CRUD gerados automaticamente.

18. Dentro do módulo `'geral'`, abriremos o pacote `consultarEstudante` e editaremos a especificação do `usecase ConsultaEstudanteUC`, adicionando a ele o estereótipo «`FrontEndApplication`».

19. Agora precisamos regerar o projeto, bem como os módulos separados, para isso, executaremos então os comandos:

```
maven mda -Dprojeto=sistemaacademico-geral-Estudante
maven mda -Dprojeto=sistemaacademico-geral-Curso
maven install
```

3.3.2 Criando o Banco de Dados

Durante a execução do comando `maven`, todas as classes são criadas automaticamente. Além disso, também é gerado o código `SQL` de criação de tabelas do Banco de Dados. O script `SQL` pode ser encontrado em `<DiretorioProjeto>/core/cd/target/schema-create.sql`. Abrindo o arquivo é possível notar a presença de comandos de crição das tabelas `ESTUDANTE` e `CURSO`.

Execute o conteúdo do arquivo no Banco de Dados utilizado.

Como exemplificação dos casos de usos que serão elaborados por este documento, execute o seguinte script `SQL` para criar a base inicial. Note que o `script` foi escrito para `PostgreSQL` e deve ser adaptado para o Banco de Dados escolhido.

```
INSERT INTO CURSO (ID, HIBERNATE_VERSION, CODIGO, NOME) VALUES
      (nextval('curso_seq'), 0, '001', 'Curso 1'),
      (nextval('curso_seq'), 0, '002', 'Curso 2');

INSERT INTO ESTUDANTE (ID, MATRICULA, NOME, CURSO_FK) VALUES
      (nextval('estudante_seq'), '0001', 'Estudante 1', 1),
      (nextval('estudante_seq'), '0002', 'Estudante 2', 2),
      (nextval('estudante_seq'), '0003', 'Estudante 3', 1),
      (nextval('estudante_seq'), '0004', 'Estudante 4', 1);
```

3.3.3 Criando Value Objects

Em breve, implementaremos os métodos das classes de controle dos casos de uso gerados, onde o usuário poderá, entre outras coisas, filtrar a consulta preenchendo um ou mais atributos da entidade desejada. Por exemplo, ele poderá realizar a consulta por um Estudante digitando sua `matrícula` (uma restrição), ou poderá digitar também seu `nome` (outra restrição).

Precisaremos, então, de uma classe que carregue esses valores atráves das camadas do nosso sistema, funcionando como uma estrutura de armazenamento de dados, para que a consulta seja feita com as restrições corretas, a qual chamos `ValueObject`, ou, abreviadamente, `VO`. Os `VOs` são responsáveis por transportar os dados fornecidos pelo usuário, e sendo utilizados na criação de restrições para a consulta sobre esses dados. Ou seja, esta classe será preenchida no controle com os dados digitados pelo usuário, será passada para um método da camada de serviços e daí seguirá até o objeto de acesso aos dados (`DAO`) onde será executada a consulta.

Para uma classe ser entendida pelo `MDArte` como um `ValueObject`, ela precisa ser marcada com o estereótipo «`ValueObject`». A comunidade do MDArte adota como padrão a prática de concentrar todas as classes `ValueObject` em um mesmo pacote, `<PacoteProjeto>.vo`.

Para criar os `ValueObject`:

1. Crie uma pasta "`vo`" dentro do pacote do projeto.

2. Crie um diagrama de classes dentro da pasta `vo`.

3. No diagrama, crie uma classe com nome "`EstudanteVO`".

4. Para cada atributo da classe `Estudante`, crie um atributo com o mesmo nome e tipo dentro da classe `EstudanteVO`. Ou seja, devem ser criados os atributos, todos públicos, `id(Long[datatype])`, `matricula(String[datatype])` e `nome(String[datatype])`.

5. Adicione o estereótipo «`ValueObject`». Assim, clique na classe o botão direito, na opção `Stereotype`, selecione o estereótipo e clique em `Apply`.

6. No mesmo diagrama de classe, crie outro `VO` com nome `CursoVO` e estereótipo «`ValueObject`».

7. Para cada atributo da classe `Curso`, crie um atributo com o mesmo nome e tipo dentro da classe `CursoVO`. Ou seja, devem ser criados os atributos, todos públicos, `id(Long[datatype])`, `codigo(String[datatype])` e `nome(String[datatype])`.

O diagrama de classe resultante pode ser visto na imagem 3.12.

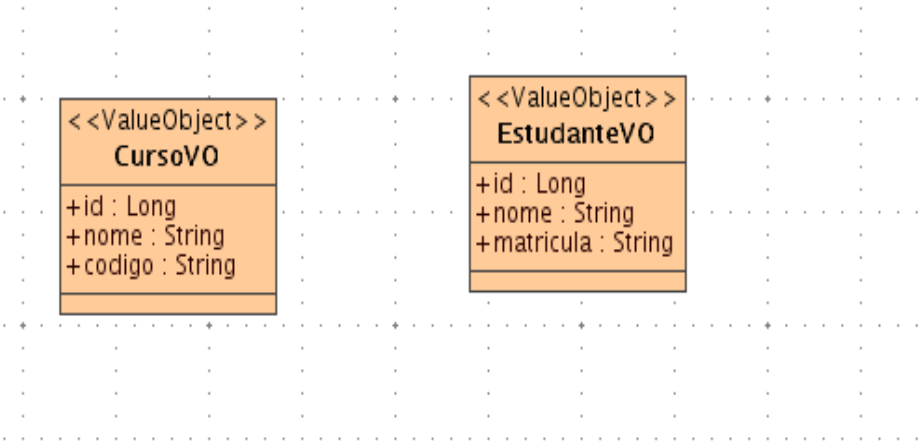

Figure 3.12: Criação dos ValueObject.

Note que devido ao fato de nosso sistema ser extremamente simples, se resumindo a `CRUD`, só necessitaremos de `ValueObject` que representem as entidades da camada de domínio. No entanto, `ValueObjects` não precisam estar obrigatoriamente associados a uma entidade específica, podendo ser usados para transportar todo o tipo de dados que precisarmos. Se tivermos um serviço que envolva dados de mais uma entidade, por exemplo, podemos para tal criar um `ValueObject` que contenha os campos de ambas as entidades que se façam necessários.

Ao final desta seção a estrutura de pacotes do sue projeto deve estar como na imagem 3.13.

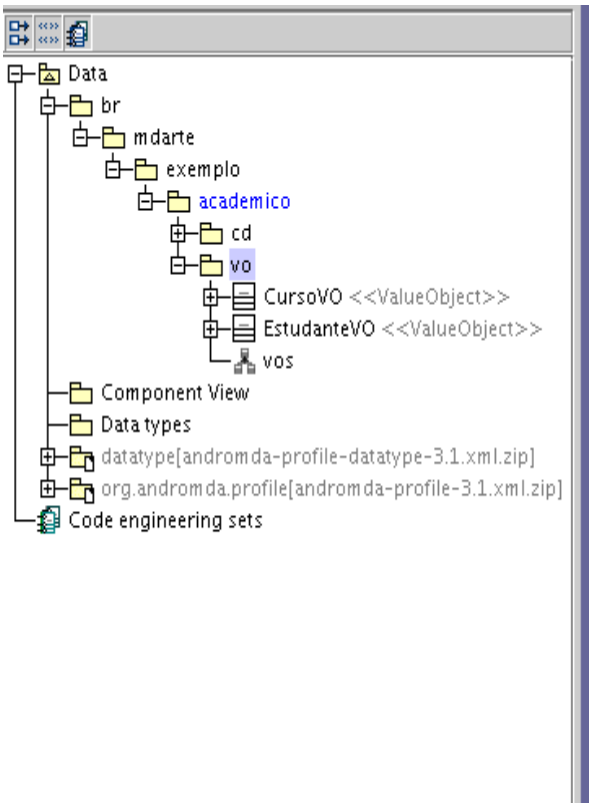

Figure 3.13: Estrutura de pacotes após a criação dos ValueObject.

3.3.4 Modelando a camada de serviços

Na camada de serviço serão implementadas as classes responsáveis pela lógica de negócio da aplicação. As classes especificadas se tornarão os serviços (`API`) da aplicação. Os serviços definidos no modelo se tornarão disponíveis através de `Session Beans`.

Os `Session Beans` são componentes de negócio. A lógica de negócio dos componentes `EJB` se encontram nestes componentes. Existem dois tipos de Componentes `Session Bean`, o `Stateless Session Bean` e o `Stateful Session Beans`. O `Stateless` é um componente de negócio que não mantém conversação com o usuário, não há garantia que chamadas sucessivas de métodos remotos vão ser feitas no mesmo objeto. O `Stateful` é um componente que mantêm estado, com ele temos a garantia que chamadas sucessivas de métodos remotos feitas por um mesmo cliente serão processadas por um mesmo objeto.

Os `beans EJB` precisam ser modelados em um diagrama de classes. As classes destes beans precisam ter o estereótipo «`Service`». Todas as classes de serviço devem estar no pacote `<PacoteProjeto>.cs`, em que `<PacoteProjeto>` é o pacote definido para o projeto.

1. Crie um pacote `<PacoteProjeto>.cs.estudante`. Clique então com o botão direito sobre a pasta estudante, na opção `Stereotype`, selecione o estereótipo «`ModuloServico`» e clique em `Apply`. O resultado pode ser visto na imagem 3.14:

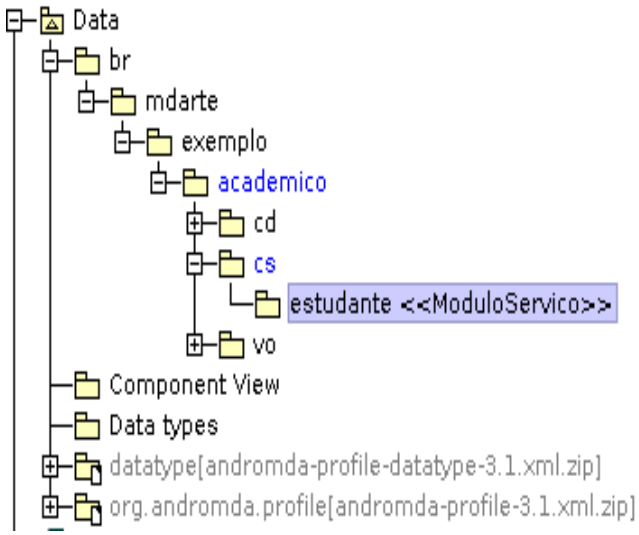

Figure 3.14: Criação do pacote para a camada de serviço.

2. Crie um diagrama de classe dentro do pacote `estudante`, com o nome que desejar.

3. Crie uma classe com nome `EstudanteHandler` e estereótipo «`Service`».A classe `EstudanteHandler` deve ficar como na figura 3.15.

Figure 3.15: Criação da classe de serviço EstudanteHandler.

4. Crie uma classe com nome `EstudanteException` e estereótipo «`ApplicationException`», como na imagem 3.16.

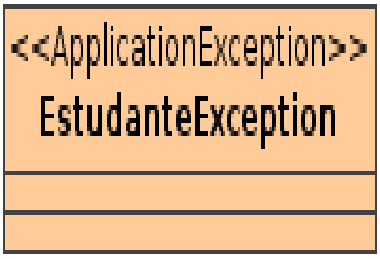

Figure 3.16: Criação da classe estudante exception.

5. Arraste para o diagrama de classes recém criado no pacote estudante a classe `Estudante`.

6. Crie uma relação de dependência entre as classes `EstudanteHandler` e `Estudante`, assim como entre `EstudanteHandler` e `EstudanteException`. Para isso, utilize a opção do `MagicDraw` ilustrada na figura abaixo. Clique na opção, depois clique na classe, ou método, de origem e arraste a seta até a classe destino, como na imagem 3.17.

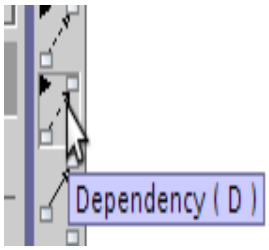

Figure 3.17: Criação da dependência entre as classes do serviço.

7. Verifique se o diagrama está como a figura 3.18.

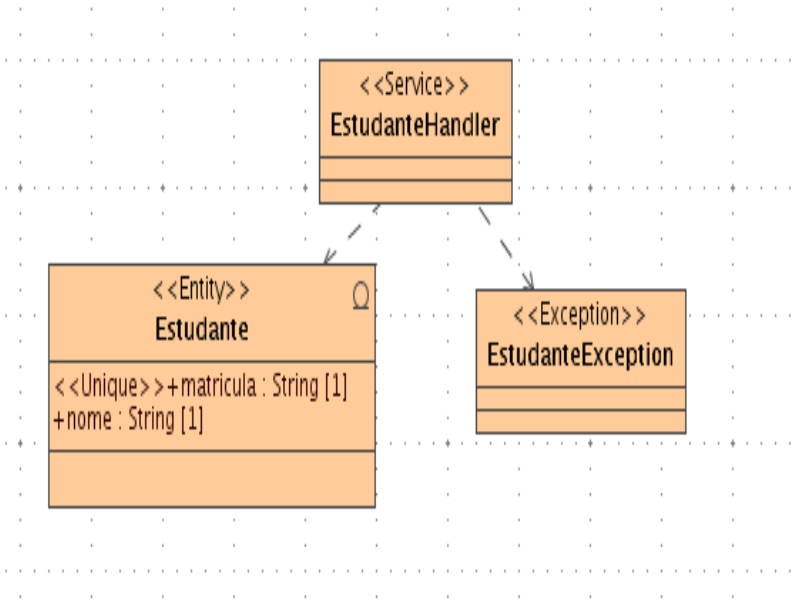

Figure 3.18: Diagrama de classe completo do módulo de serviço para a classe Estudante.

8. A dependência entre `EstudanteHandler` e `EstudanteException` fará com que todos os métodos de `EstudanteHandler` possam lançar a exceção `EstudanteException`. Se a dependência tivesse sido entre algum método de `EstudanteHandler` e não com a própria classe, somente o método com dependência poderia lançar a exceção.

   A dependência entre `EstudanteHandler` e `Estudante` cria os métodos de acesso ao banco na classe de serviço.

9. Agora faça o mesmo para criar um modulo de serviço para a classe `Curso`.

10. No diretório da aplicação, execute o comando maven para validar o modelo e gerar as classes de serviço. O resultado apresentado deve ser "BUILD SUCCESSFULL".

11. Agora, rode no banco de dados do Controle de Acesso o script Servicos.sql, que se encontra no caminho <raizProjeto>/core/cs/compartilhado/target/classes/br/mdarte/exemp
Não se esqueça de descomentar as linhas comentadas que contem código sql, essas linhas criarão um usuário de nome 'su' e senha 'su' no banco do Controle de Acesso, já com todas as permissões necessárias para a utilização do Sistema Acadêmico.

3.4 Implementando as classes de controle dos CRUD gerados

Nessa seção vamos implementar as classes de controle dos casos de uso gerados. Para isso, abriremos os arquivos <nomeCasoDeUso>ControleImpl.java. Esses arquivos são pontos de implementação onde deve ser concentrado todo o código que se queira adicionar manualmente às classes de controle. Como tais arquivos só são gerados caso ainda não existam, o código colocado neles não será sobrescrito, ao contrario do que ocorre se inserirmos manualmente codigo nos arquivos <nomeCasoDeUso>Controle.java.

De acordo com os casos de uso gerados automaticamente pelo gerador de CRUD do MDArte, iremos implementar respectivamente os seguintes código nos seguintes arquivos, que devem portanto ser abertos no Eclipse ou em outra IDE desejada:

1. ConsultaEstudanteControleImpl.java :

```
package br.mdarte.exemplo.academico.web.geral
      .consultarEstudante;

import br.mdarte.exemplo.academico
      .ServiceLocator;
import br.mdarte.exemplo.academico.util.Constantes;
import br.mdarte.exemplo.academico.util.Util;
import br.mdarte.exemplo.academico.cd.Estudante;
import br.mdarte.exemplo.academico.cd.EstudanteImpl;
import br.mdarte.exemplo.academico.to.EstudanteTO;
import br.mdarte.exemplo.academico.to.EstudanteTOImpl;
import br.mdarte.exemplo.academico.action
      .DefaultFilterAction;

import java.util.Collection;
import java.util.ArrayList;

import br.mdarte.exemplo.academico.vo.EstudanteVO;

import org.andromda.presentation.bpm4struts.ViewContainer;
```

```
/**
 * @see br.mdarte.exemplo.academico.web.geral
 *  .consultarEstudante.ConsultaEstudanteControle
 */
public class ConsultaEstudanteControleImpl extends
      ConsultaEstudanteControle
      {
   /**
    * @see br.mdarte.exemplo.academico.web.geral
    *  .consultarEstudante.ConsultaEstudanteControle
    *  #consultaEstudante(br.mdarte.exemplo.academico
    *  .web.geral.consultarEstudante
    * .ConsultaEstudanteForm)
    */
   public final void consultaEstudante(
            br.mdarte.exemplo.academico.web.geral
            .consultarEstudante.ConsultaEstudanteForm form,
            ViewContainer container ) throws Exception {

        Integer paginacao = ((Double)container
                    .getAttribute(Constantes.PARAMETRO_PAGINA))
                    .intValue();

        EstudanteTO estudanteTO = new EstudanteTOImpl();

        estudanteTO.setNome(form.getNome());
        estudanteTO.setMatricula(form.getMatricula());

        Collection estudantes = ServiceLocator.instance()
                  .getEstudanteHandlerBI().manipulaEstudante(
                  new EstudanteImpl(), new
                  DefaultFilterAction(estudanteTO,paginacao));

        ArrayList<EstudanteVO> estudanteVOs =
                    new ArrayList<EstudanteVO>();

        if(!Util.checkEmpty(estudantes)) {
            for(Estudante estudante :
                  (Collection<Estudante>) estudantes){
            EstudanteVO estudanteVO = new EstudanteVO();
```

```
                estudanteVO.setNome(estudante.getNome());
                estudanteVO.setIdEstudante(estudante.getId());
                estudanteVO.setMatricula(estudante.getMatricula());
                estudanteVOs.add(estudanteVO);
              }

                form.setEstudantes(estudanteVOs);
           }
      }

}
```

2. MantemEstudanteControleImpl.java :

```
package br.mdarte.exemplo.academico.web.geral.manterEstudante;

import br.mdarte.exemplo.academico.ServiceLocator;
import br.mdarte.exemplo.academico.cd.Estudante;
import br.mdarte.exemplo.academico.cd.EstudanteImpl;
import br.mdarte.exemplo.academico.util.Constantes;

import org.andromda.presentation.bpm4struts.ViewContainer;

/**
 * @see br.mdarte.exemplo.academico.web.geral.manterEstudante
 *  .MantemEstudanteControle
 */
public class MantemEstudanteControleImpl
      extends MantemEstudanteControle
      {
      /**
       * @see br.mdarte.exemplo.academico.web.geral
       *  .manterEstudante.MantemEstudanteControle
       * #carregaEstudante(
       *  br.mdarte.exemplo.academico.web.geral.manterEstudante
       *  .CarregaEstudanteForm)
       */
      public final void carregaEstudante(
            br.mdarte.exemplo.academico.web.geral
```

```
            .manterEstudante.CarregaEstudanteForm
            form, ViewContainer container
            ) throws Exception {

            Estudante estudante = new EstudanteImpl();

            estudante.setId(form.getId());

            estudante = (Estudante) ServiceLocator.instance()
                  .getEstudanteHandlerBI()
                  .selectEstudante(estudante).get(0);

            form.setNome(estudante.getNome());

            form.setMatricula(estudante.getMatricula());

            form.setIdEstudante(estudante.getId());

      }

      /**
       * @see br.mdarte.exemplo.academico.web.geral
       *  .manterEstudante.MantemEstudanteControle
       *  #salvaEstudante(br.mdarte.exemplo.academico
       *  .web.geral.manterEstudante
       * .SalvaEstudanteForm)
       */
      public final void salvaEstudante(
            br.mdarte.exemplo.academico.web.geral
            .manterEstudante.SalvaEstudanteForm
            form, ViewContainer container
            ) throws Exception {

            Estudante estudante = new EstudanteImpl();

            estudante.setId(form.getId());

            estudante.setNome(form.getNome());

            estudante.setMatricula(form.getMatricula());
```

```
                ServiceLocator.instance().getEstudanteHandlerBI()
                      .updateEstudante(estudante);

        }
    }
```

3. DetalhaEstudanteControleImpl.java:

```
    package br.mdarte.exemplo.academico.web.geral
          .detalharEstudante;

    import br.mdarte.exemplo.academico.ServiceLocator;
    import br.mdarte.exemplo.academico.util.Constantes;
    import org.andromda.presentation.bpm4struts.ViewContainer;
    import br.mdarte.exemplo.academico.cd.Estudante;
    import br.mdarte.exemplo.academico.cd.EstudanteImpl;
    import br.mdarte.exemplo.academico.vo.EstudanteVO;

    /**
     * @see br.mdarte.exemplo.academico.web.geral.detalharEstudante
     *  .DetalhaEstudanteControle
     */
    public class DetalhaEstudanteControleImpl
          extends DetalhaEstudanteControle
          {
       /**
        * @see br.mdarte.exemplo.academico.web.geral
        *  .detalharEstudante.DetalhaEstudanteControle#
        *  carregaEstudante(br.mdarte.exemplo.academico.web
        *  .geral.detalharEstudante.CarregaEstudanteForm)
        */
       public final void carregaEstudante(
          br.mdarte.exemplo.academico.web.geral.detalharEstudante
          .CarregaEstudanteForm form, ViewContainer container
          ) throws Exception {

          Estudante estudante = new EstudanteImpl();

          estudante.setId(form.getIdEstudante());
```

```
        estudante = (Estudante) ServiceLocator.instance()
              .getEstudanteHandlerBI().selectEstudante(estudante)
              .iterator().next();

        if(estudante != null){
              form.setNome(estudante.getNome());

              form.setMatricula(estudante.getMatricula());

              form.setIdEstudante(e.getId());
        }
    }
}
```

4. ConsultaCursoControleImpl.java :

```
import br.mdarte.exemplo.academico
      .ServiceLocator;
import br.mdarte.exemplo.academico.util.Constantes;
import br.mdarte.exemplo.academico.util.Util;
import br.mdarte.exemplo.academico.cd.Curso;
import br.mdarte.exemplo.academico.cd.CursoImpl;
import br.mdarte.exemplo.academico.to.CursoTO;
import br.mdarte.exemplo.academico.to.CursoTOImpl;
import br.mdarte.exemplo.academico.action
      .DefaultFilterAction;
import java.util.Collection;
import java.util.ArrayList;
import br.mdarte.exemplo.academico.vo.CursoVO;
import org.andromda.presentation.bpm4struts.ViewContainer;

/**
 * @see br.mdarte.exemplo.academico.web.geral
 *  .consultarCurso.ConsultaCursoControle
 */
public class ConsultaCursoControleImpl extends
      ConsultaCursoControle
      {
```

```
/**
 * @see br.mdarte.exemplo.academico.web.geral
 *  .consultarCurso.ConsultaCursoControle
 *  #consultaCurso(br.mdarte.exemplo.academico
 *  .web.geral.consultarCurso
 * .ConsultaCursoForm)
 */
public final void consultaCurso(
        br.mdarte.exemplo.academico.web.geral
        .consultarCurso.ConsultaCursoForm form,
        ViewContainer container ) throws Exception {

    Integer paginacao = ((Double)container
                .getAttribute(Constantes.PARAMETRO_PAGINA))
                .intValue();

    CursoTO cursoTO = new CursoTOImpl();

    cursoTO.setNome(form.getNome());
    cursoTO.setMatricula(form.getMatricula());

    Collection cursos = ServiceLocator.instance()
              .getCursoHandlerBI().manipulaCurso(
              new CursoImpl(),
              new DefaultFilterAction(cursoTO,paginacao));

    ArrayList<CursoVO> cursoVOs = new ArrayList<CursoVO>();

    if(!Util.checkEmpty(cursos)) {
        for(Curso curso :
              (Collection<Curso>) cursos){
        CursoVO cursoVO = new CursoVO();
        cursoVO.setNome(curso.getNome());
        cursoVO.setIdCurso(curso.getId());
        cursoVO.setMatricula(curso.getMatricula());
        cursoVOs.add(cursoVO);
       }

        form.setCursos(cursoVOs);
    }
}
```

```
}
```

5. MantemCursoControleImpl.java :

```
import br.mdarte.exemplo.academico.ServiceLocator;
import br.mdarte.exemplo.academico.util.Constantes;
import br.mdarte.exemplo.academico.cd.CursoImpl;
import br.mdarte.exemplo.academico.cd.Curso;
import org.andromda.presentation.bpm4struts
      .ViewContainer;

/**
 * @see br.mdarte.exemplo.academico.web.geral
 * .manterCurso.MantemCursoControle
 */
public class MantemCursoControleImpl
      extends MantemCursoControle
{
   /**
    * @see br.mdarte.exemplo.academico.web.geral
    * .manterCurso.MantemCursoControle
    * #carregaCurso(br.mdarte.exemplo.academico
    * .web.geral.manterCurso.CarregaCursoForm)
    */
   public final void carregaCurso(
      br.mdarte.exemplo.academico.web.geral
      .manterCurso.CarregaCursoForm form,
      ViewContainer container)
                  throws Exception {

      Curso curso = new CursoImpl();
      curso.setId(form.getIdCurso());

      curso = (Curso) ServiceLocator.instance()
            .getCursoHandlerBI().selectCurso(curso)
            .iterator().next();

      if(curso != null) {
```

```
                form.setNome(curso.getNome());
                form.setMatricula(curso.getMatricula());
                form.setIdCurso(curso.getId());
            }
        }

        /**
         * @see br.mdarte.exemplo.academico.web.geral
         * .manterCurso.MantemCursoControle
         * #salvaCurso(br.mdarte.exemplo.academico
         * .web.geral.manterCurso.SalvaCursoForm)
         */
        public final void salvaCurso(
            br.mdarte.exemplo.academico
            .web.geral.manterCurso.SalvaCursoForm form,
            ViewContainer container)
                        throws Exception {
            Curso curso = new CursoImpl();
            curso.setId(form.getIdCurso());

            curso =(Curso) ServiceLocator.instance()
                  .getCursoHandlerBI().selectCurso(curso)
                  .iterator().next();

            if(curso != null) {
                  curso.setMatricula(
                        form.getMatricula());
                  curso.setNome(form.getNome());

                  ServiceLocator.instance()
                        .getCursoHandlerBI().insertCurso(curso);

                  this.saveSuccessMessage(
                        "mantem.curso.inserido.sucesso",
                        container);
            }
        }
    }
```

6. DetalhaCursoControleImpl.java :

```
import br.mdarte.exemplo.academico.ServiceLocator;
import br.mdarte.exemplo.academico.util.Constantes;
import org.andromda.presentation.bpm4struts.ViewContainer;
import br.mdarte.exemplo.academico.cd.Curso;
import br.mdarte.exemplo.academico.cd.CursoImpl;
import br.mdarte.exemplo.academico.vo.CursoVO;

/**
 * @see br.mdarte.exemplo.academico.web.geral.detalharCurso
 *  .DetalhaCursoControle
 */
public class DetalhaCursoControleImpl
      extends DetalhaCursoControle
      {
   /**
    * @see br.mdarte.exemplo.academico.web.geral
    *  .detalharCurso.DetalhaCursoControle#
    *  carregaCurso(br.mdarte.exemplo.academico.web
    *  .geral.detalharCurso.CarregaCursoForm)
    */
   public final void carregaCurso(
      br.mdarte.exemplo.academico.web.geral.detalharCurso
      .CarregaCursoForm form, ViewContainer container
      ) throws Exception {

      Curso curso = new CursoImpl();

      curso.setId(form.getIdCurso());

      curso = (Curso) ServiceLocator.instance()
            .getCursoHandlerBI().selectCurso(curso)
            .iterator().next();

      if(curso != null){
            form.setNome(curso.getNome());

            form.setMatricula(curso.getCodigo());

            form.setIdCurso(e.getId());
      }
   }
```

```
    }
```

Agora, no terminal, no <DiretorioProjeto> executaremos o seguinte comando :

```
maven compile deploy
```

3.4.1 Inicializando o servidor e testanto a aplicação

O `eclipse` nos fornece a possibilidade de fazer o gerenciamento do servidor `JBoss` dentro da própria IDE. Para isso precisamos criar, na IDE, um novo servidor.

Primeiramente, certifique-se de que você está usando a perspectiva de desenvolvimento `Java EE`. No conjunto de abas na parte inferior da IDE, abaixo da parte onde fica o código, selecione a aba `servers` como na imagem 3.19 .

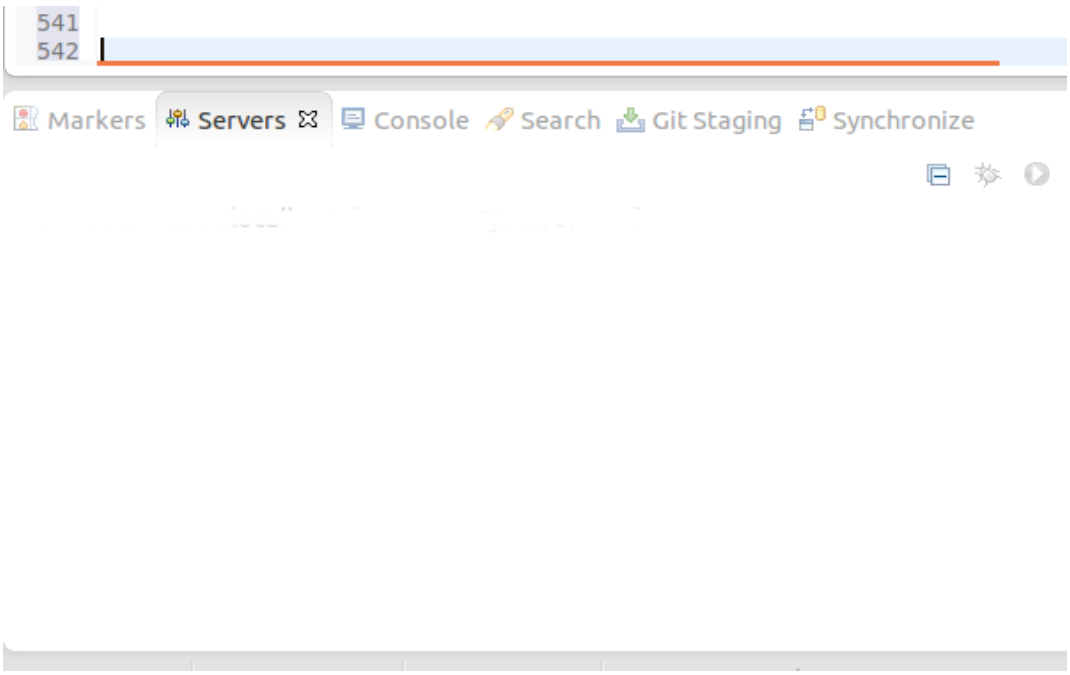

Figure 3.19: Inicializando o servidor JBoss.

Agora clique com o botão direito no espaço vazio, vá em `new > server`. Será aberta a janela `'New Server'`. Preencha os dados como na imagem 3.20.

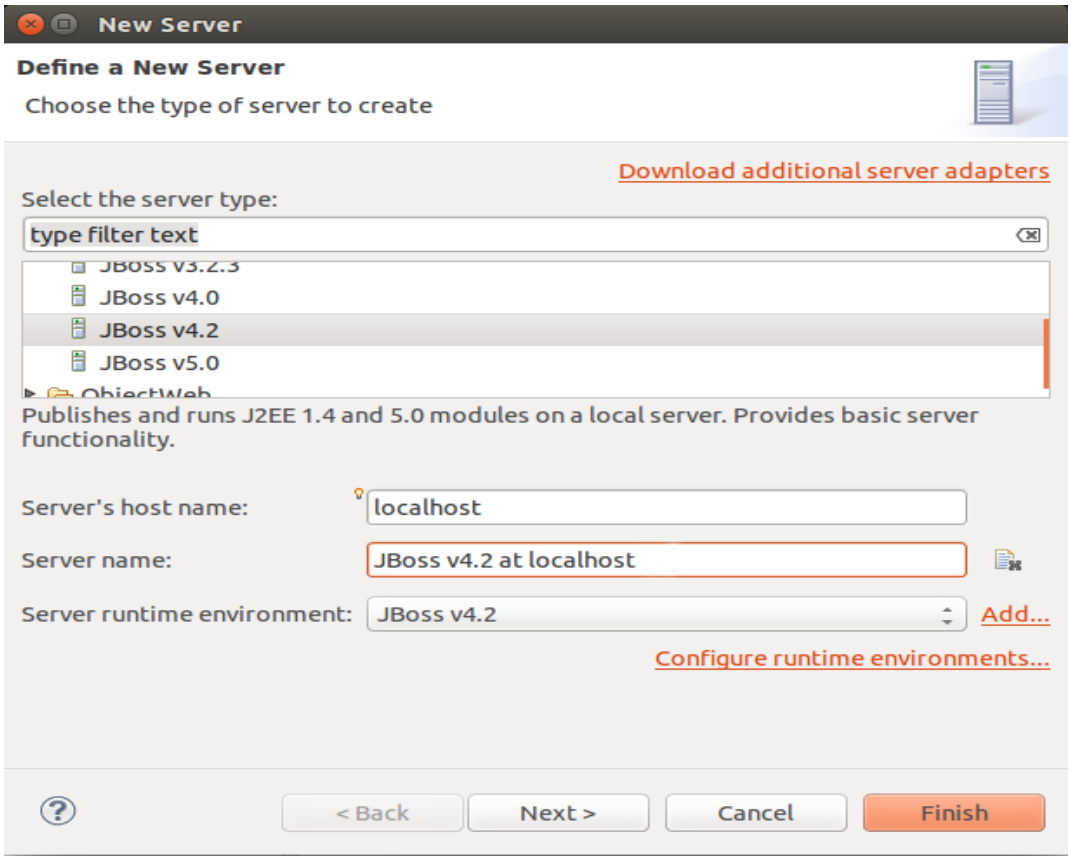

Figure 3.20: Inicializando o servidor JBoss.

Agora daremos `Start` no servidor `Jboss` e verificaremos então no navegador o resultado do nosso sistema. Para dar `Start` no servidor iremos na aba `Servers` no nossa `IDE`, selecionaremos o servidor e clicaremos então no botão de `Start` (Verde), como na imagem 3.21:

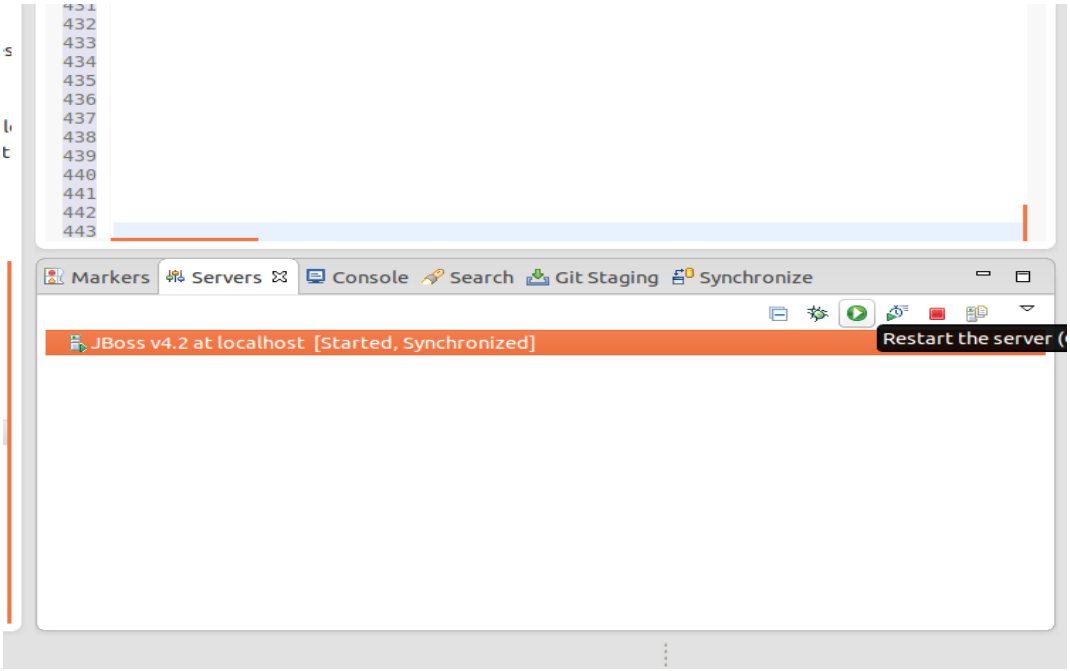

Figure 3.21: Inicializando o servidor JBoss.

Para acessar o sistema, abriremos o navegador e acessaremos a url `http://localhost:8080/sistemaacademic` Primeiramente será aberta a página de login do sistema, como na imagem 3.22.

O usuário padrão criado pelo script rodado é 'su' e a senha 'su'.

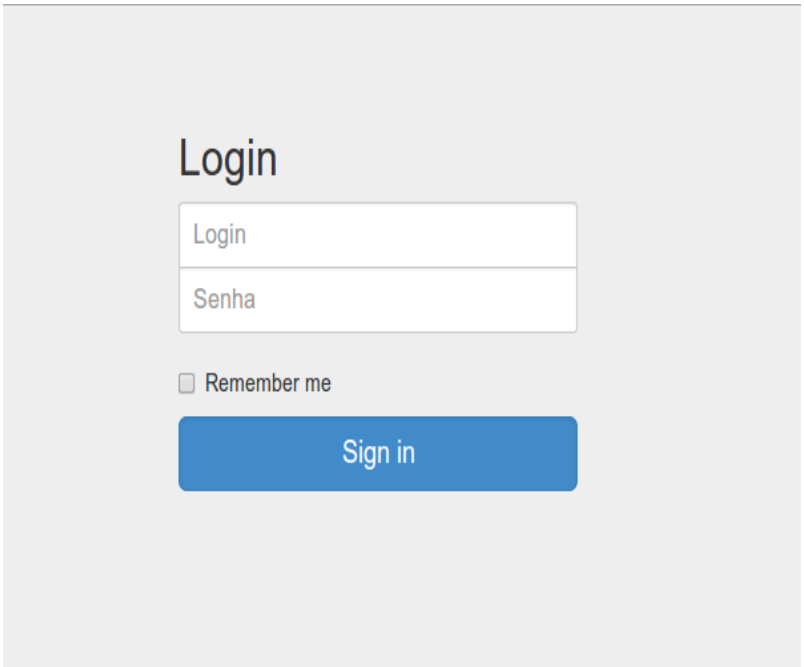

Figure 3.22: Página de login do Sistema Acadêmico.

Feito o login, a tela inicial do sistema será aberta como na imagem

Ao clicar no botão `Consulta Estudante` seremos redirecionados para a página de consulta no caso de uso de mesmo nome como na imagem 3.23.

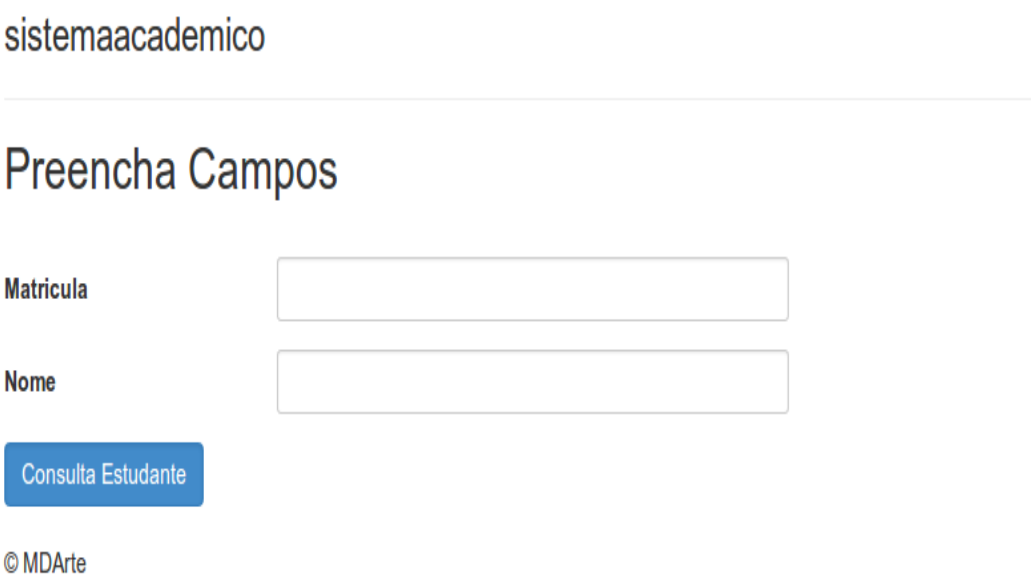

Figure 3.23: Página incial de Consulta Estudante.

Se pesquisarmos sem nenhum filtro veremos uma lista completa dos estudantes registrados como na imagem 3.24.

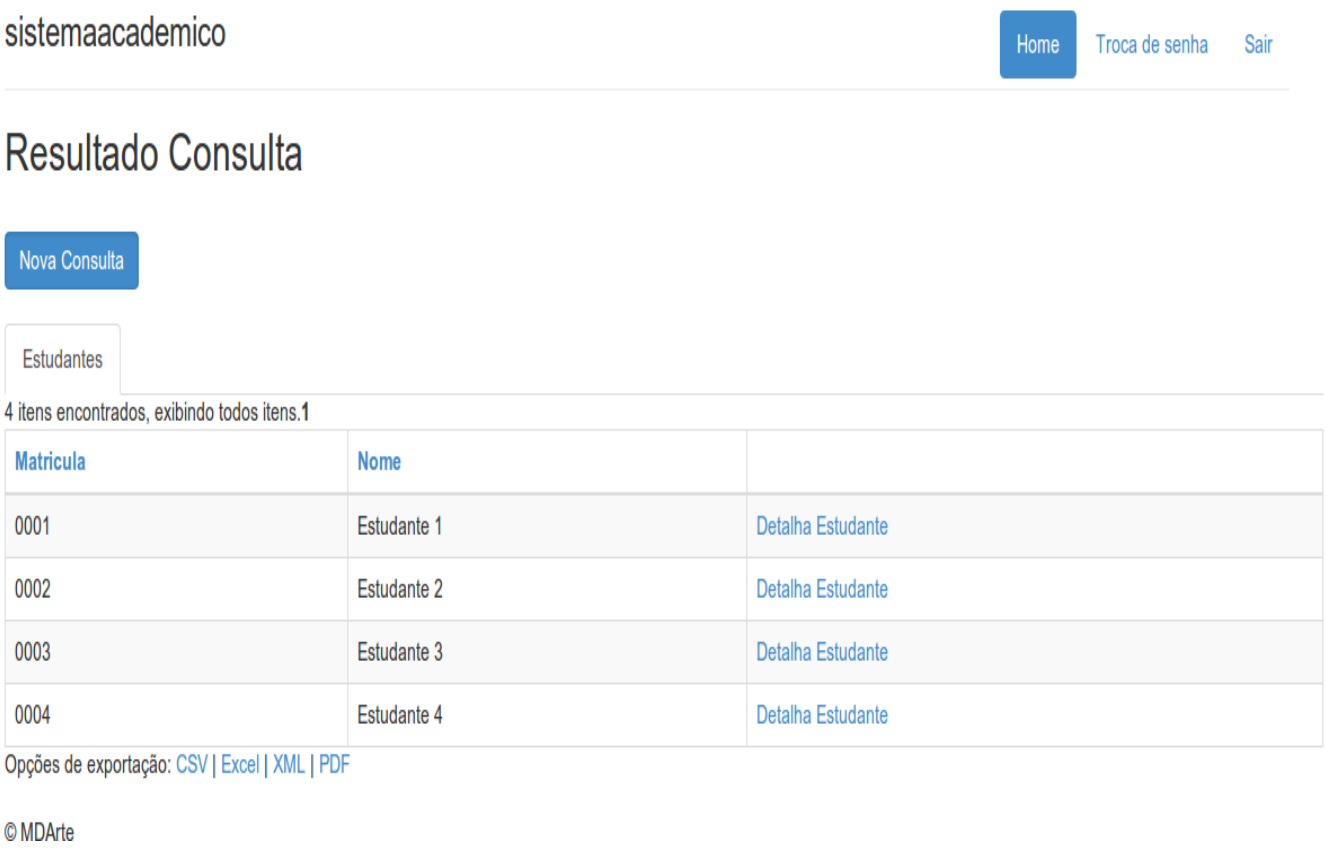

Figure 3.24: Resultado da busca sem filtro pelos estudantes.

Chapter 4

Funcionalidades do MDArte

Neste capítulo iremos explorar algumas funcionalidades que já existem no MDArte, a fim de agilizar e simplificar o processo de desenvolvimento. Para tal alteraremos os modelos de `CRUD` gerados automaticamente pelo `MDArte`. Para evitar que as alterações feitas sejam sobrescritas por engano, vá no diagrama de classe que descreve as entidades do Banco de Dados, abra a especificação da classe `Estudante`, selecione a aba `stereotypes` e remova o estereótipo «`Manageable`».

4.1 Campo com Autocomplete

Nesta seção veremos como implementar um `autocomplete` para um determinado campo de texto. Iremos transformar o campo `matricula` do caso de uso `Consulta Estudante` em um campo com `autocomplete`.

O modelo inicial do caso de uso `Consulta Estudante` no `CRUD` para a entidade estudante pode ser visto na imagem 4.1:

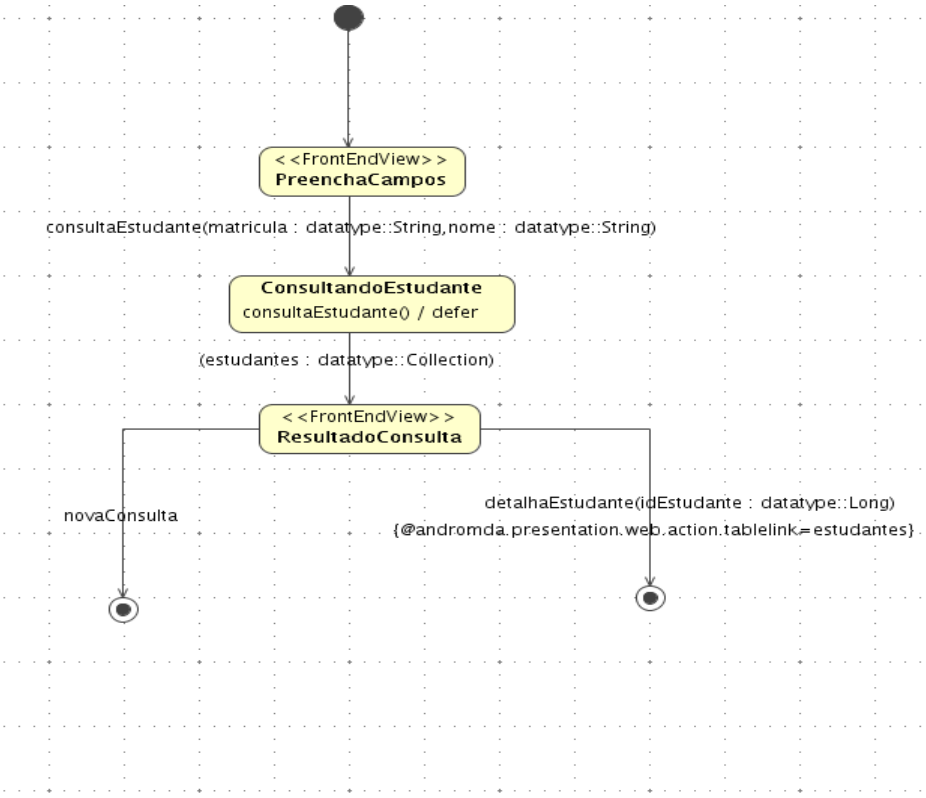

Figure 4.1: Modelo do caso de uso Consulta Estudante.

Abriremos a especificação da `transition` que sai da `front end view` de nome `preencha os campos`, clicaremos no botão `edit`, no `fieldset` `trigger`. Na aba `parameters`, da janela `signal event specification`, que será aberta automaticamente, dê duplo clique no nome do parâmetro `matricula` e será então aberta a especificação do parâmetro. Selecione então a aba `tagged values`, selecione o `tagged value` `@andromda.presentation.web.view.field.type` e clique no botão `create value`. Selecione então a opção `autocomplete`, como na imagem 4.2.

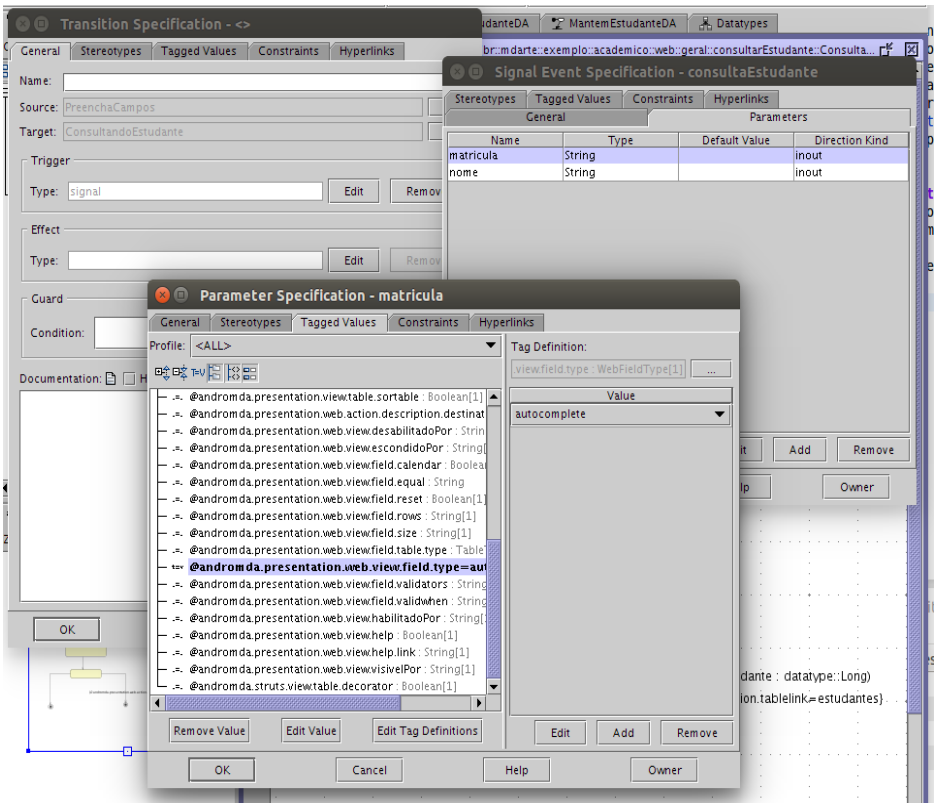

Figure 4.2: Adicionando o field type 'autocomplete' à um campo da view.

Agora executaremos o seguinte comando no terminal na raiz do projeto:

```
maven mda -Dprojeto=sistemaacademico-geral-Estudante
```

Feito isto, o `MDArte` gerará automaticamente toda a estrutura responsável por receber e tratar as requisições assíncronas para o preenchimento do `autocomplete`, restando ao desenvolver apenas implementar no `ControleImpl` a filtragem dos valores retornados, de acordo com o valor do campo. Para isto, criaremos, na classe `ConsultaEstudanteControleImpl`, um método seguindo o seguinte padrão `protected String[] <nome-do-campo><nome-do-caso-de-uso>AutoComplete(java.lang.query, org.andromda.bpm4struts.ViewContainer container)`. Vejamos abaixo um exemplo de implementação para o `autocomplete` do nosso campo de `matrícula`:

```
protected String[] matriculaPreenchaCamposAutoComplete(
            String query, ViewContainer container) throws
            Exception {

      EstudanteTO estudanteTO = new EstudanteTOImpl();

   estudanteTO.setMatricula(query);

   Collection estudantes = ServiceLocator.instance()
            .getEstudanteHandlerBI().manipulaEstudante(
            new EstudanteImpl(),
            new DefaultFilterAction(estudanteTO,0));
```

```
   String[] matriculas = new String[10];

   int i =0;

   if(!Util.checkEmpty(estudantes)) {
      Iterator iterator = estudantes.iterator();
            while(i<10 ){
                  matriculas[i++] = iterator.hasNext() ?
                              ((Estudante)iterator.next())
                              .getMatricula() : "";
            }
   }

   return matriculas;
}
```

Agora executaremos o seguinte comando para compilar e dar `deploy` no `Sistema Acadêmico`:

```
maven compile deploy
```

Feito isto, daremos `start` no `JBoss` e abriremos o sistema e faremos login. Na tela `Preencha os Campos` do caso de uso `Consulta Estudante` podemos agora verificar o `autocomplete` funcionando, como na imagem 4.3.

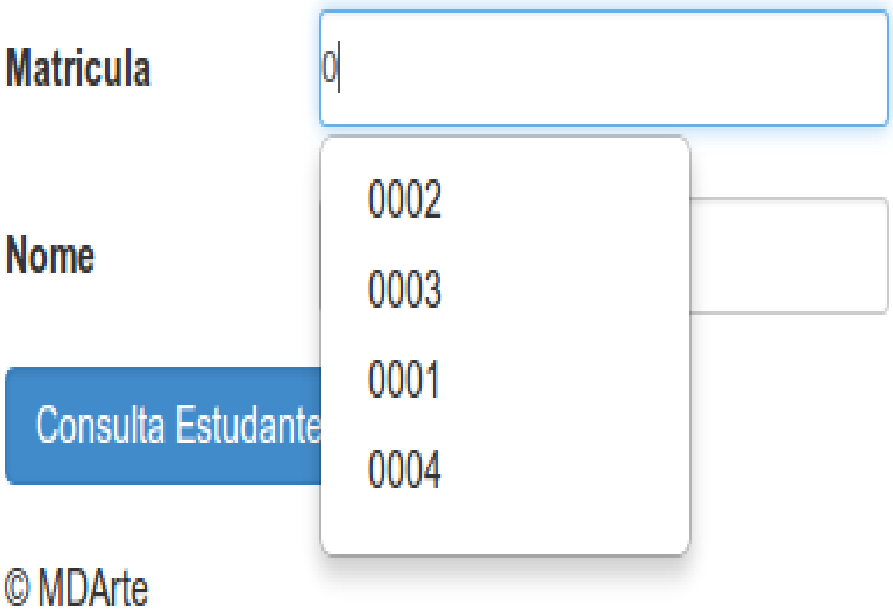

Figure 4.3: Exemplo de autocomplete.

4.2 Estratégia de paginação

Estratégias de paginação foram criadas para comportar diferentes tipos de tabela e paginação. Nas versões anteriores, a paginação era feita do mesmo jeito para as tabelas Struts e Ajax porém elas tinham estruturas diferentes o que gerava erros nas tabelas. Dessa forma, não poderíamos usar a tabela Ajax pois tinhamos como prioridade ter a tabela Struts funcionando. Com as estratégias de paginação podemos utilizar qualquer tipo de tabela sem ter que alterar a estrutura dos DAOs e damos a opção para o desenvolvedor customizar a paginação do jeito que quiser.

Nesta seção demonstraremos como utilizar estratégias de paginação. Primeiro, apresentaremos a classe abstrata `PaginationStrategy`:

```
package br.mdarte.exemplo.academico.util;

import br.mdarte.exemplo.academico.Constantes;

public abstract class PaginationStrategy {

      protected Integer paginaAtual;
      protected Integer linhas;
      protected Integer paginas;

      public PaginationStrategy () {
```

```
        this.paginaAtual = null;
        this.linhas = Constantes.TABLE_LINES;
        this.paginas = Constantes.TABLE_PAGES;
    }

    public PaginationStrategy (Integer paginaAtual) {
        this.paginaAtual = paginaAtual;
        this.linhas = Constantes.TABLE_LINES;
        this.paginas = Constantes.TABLE_PAGES;
    }

    public PaginationStrategy (Integer paginaAtual, Integer linhas,
        Integer paginas) {
        this.paginaAtual = paginaAtual;
        this.linhas = linhas;
        this.paginas = paginas;
    }

    public abstract void paginateResult(org.hibernate.Query res);

    public abstract void paginateResult(org.hibernate.Criteria res)
       ;

    public Integer getPaginaAtual() {
        return this.paginaAtual;
    }

    public void setPaginaAtual(Integer pagina) {
        this.paginaAtual = pagina;
    }

    public Integer getLinhas() {
        return this.linhas;
    }

    public void setLinhas(Integer linhas) {
        this.linhas = linhas;
    }

    public Integer getPaginas() {
        return this.paginas;
    }
```

```
	public void setPaginas(Integer paginas) {
		this.paginas = paginas;
	}

}
```

A classe abstrata `PaginationStrategy` é gerado no pacote util do projeto. Nele você seta a página a ser obtida, o número de linhas por página e o número de páginas a serem retornadas pela `query` e `criteria`.

Nela há a função `paginateResult` que possui duas assinaturas diferentes. O desenvolvedor tem que implementar essas funções caso deseje criar a sua própria estratégia. Um exemplo de implementação de estratégia é o `PaginationSimple.java` que é utilizado para a paginação da tabela ajax.

```
package br.mdarte.exemplo.academico.util;

public class PaginationSimple extends PaginationStrategy {

	public PaginationSimple(Integer paginaAtual) {
		super(paginaAtual);
	}

	public PaginationSimple(Integer paginaAtual, Integer linhas) {
		super(paginaAtual);
		this.linhas = linhas;
	}

	public PaginationSimple(Integer paginaAtual, Integer linhas,
	   Integer paginas) {
		super(paginaAtual, linhas, paginas);
	}

	public void paginateResult(org.hibernate.Query res) {

		if (paginaAtual != null){
			res.setFirstResult(((paginaAtual)/paginas)*linhas*
			   paginas);
			res.setMaxResults(linhas*paginas);
		}

	}

	public void paginateResult(org.hibernate.Criteria res) {
```

```
            if (paginaAtual != null){
                  res.setFirstResult(((paginaAtual)/paginas)*linhas*
                     paginas);
                  res.setMaxResults(linhas*paginas);
            }

      }

}
```

Além dessa estratégia, criamos outras duas: `PaginationDisplaytag` para as tabelas Struts e `NoPagination` caso não seja preciso de paginação.

A pasta em que sua implementação deve ser criada é <projeto>/common/src/java/<pacote_do_projeto>/util e o import a ser feito é `<pacote_do_projeto>.util.<nome>`.

Demonstramos um simples exemplo para instanciar uma estratégia abaixo:

```
      import br.mdarte.exemplo.academico.util.PaginationDisplaytag;
      import br.mdarte.exemplo.academico.util.Constantes;

      Integer pagina = ((Integer)request.getAttribute(Constantes.
         PARAMETRO_PAGINA)); //Struts 1
      Collection exemplos = ServiceLocator.instance().
         getExemploHandlerBI().recuperaExemplos(new
         PaginationDisplaytag(paginacao));
```

4.3 Internacionalização

Internacionalizar aplicações web é cada vez mais uma tarefa corriqueira de todo desenvolvedor web e é um dos processos importantes para o aumento da acessibilidade do sistema. O framework deve permitir mecanismo para facilitar a utilização de diversas línguas. Para tal, foi construido um conjunto de ferramentas para facilitar esse processo. A maioria dos frameworks web tem a sua maneira particular de prover esse mecanismo, mas o que muita gente desconhece é que existe uma forma padrão de fazer isso, definida na especificação do Java EE, através da JSP Standard TagLibs.

O MDArte utiliza esse mecanismo e já provê ela configurada. Assim, a especificação de novas linguas se torna uma tarefa ainda mais fácil.

Segue abaixo os passos para a definições de novas linguas.

4.3.1 Mensagens

Cada arquivo properties conterá todas as traduções do sistema. Todos os textos do sistemas serão representados por uma `key`. Cada `key` e sua respectiva mensagem, ou seja, `label` serão listadas em cada arquivo como properties. Abaixo segue um exemplo:

```
      label.key=Mensagem
```

Todas os recursos modelados no diagrama de atividades serão gerados com uma `key`. Caso essa `key` não esteja no arquivo properties, o sistema utilizará a `key` como `label` contudo sem o ponto.

Cada uma das possibilidades será abordada.

Título de uma Página

Todo título de página é gerado com uma key para o desenvolvedor possa definir no custom-resources.

```
      <tiles:put name="title" type="string">
      <bean:message key="pagina.exemplo.title"/>
      </tiles:put>
```

Campo ou Botão de uma Página

Todo campo ou botão é gerado com uma key definida pelo o nome do caso de uso somado com o nome do campo/botão. Segue abaixo um exemplo.

```
      <td class="field">
      <s:set name="__value" value="#session.form.nome"/>
      <div id="divnomeConsultaCursoUC" class="textfield field">
      <label class="textfieldLabel" for="nome"><bean:message key="
         consulta.curso.uc.preencha.campos.consulta.curso.param.nome"
         /></label>
      <s:textfield id="nomeConsultaCursoUC" name="nome" label="%{
         getText('consulta.curso.uc.preencha.campos.consulta.curso.
         param.nome')}" value="%{#session.form.nome}" title=""
         styleId="consultaCursoNome" />
</div>
```

Exception

Toda mensagem carregando uma `exception` também é substituída quando a `key` na `exception` é encontrada no `custom-resource`. Segue abaixo um exemplo.

```
      throw new Exception("ocorre.erro.esperado");

      throw new Exception("ocorre.erro.nao.esperado", exception);
```

Existem algumas funções que não interrompem a execução, mas possui o mesmo efeito do `exception`. Segue abaixo um exemplo.

```
      saveErrorMessage(request, "informando.erro.key"); //Struts 1
      saveWarningMessage(request, "informando.aviso.key"); //Struts 1
      saveSuccessMessage(request, "informando.sucesso.key"); //Struts
          1
      saveErrorMessage("informando.erro.key", container); //Struts 2
```

```
saveWarningMessage("informando.aviso.key", container); //Struts
    2
saveSuccessMessage("informando.sucesso.key", container); //
   Struts 2
```

A função `saveErrorMessage` gera a seguinte mensagem na página:

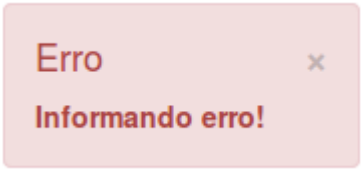

Figure 4.4: Mensagem de erro

Passagem de Parâmetros

Existe a possibilidade de passar parâmetros para a mensagem. Será passando um `array` de `string` e na mensagem existirá marcadores informando onde será colocado o conteúdo do `array`.

Exemplo:

`custom-resources.properties`

```
label.key=Mensagem com parametro {0}
```

`Código:`

```
String[] parametro = new String[1];
parametro[0] = "param1";
saveErrorMessage("label.key", parametro, container); //Struts 2
//saveErrorMessage(request, "label.key", parametro); //Struts 1
```

Assim será exibido a mensagem `Mensagem com parametro param1` na figura abaixo:

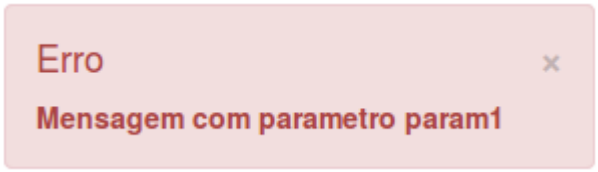

Figure 4.5: Mensagem de erro com parâmetros

4.3.2 Arquivo de Configurações

Edite o arquivo **<DiretorioProjeto>**/mda/conf/andromda.xml e na propriedade `languages` informe os locales que serão utilizados.

Exemplo:

```
<property name="languages">pt,en,fr</property>
```

Após a geração utilizando essa configuração será criado novos três arquivos:

```
custom-resources_en.properties
custom-resources_pt.properties
custom-resources_fr.properties
```

Além do arquivo já criado no geração da aplicação:

```
custom-resources.properties
```

Automaticamente o sistema irá detectar qual locale do browser que o usuário está utilizando e utilizará o locale correto. Caso não exista o locale pré-definido, o sistema utilizará o `custom-resources` padrão.

O desenvolvedor poderá forçar um locale específico pelo código. Exemplo:

```
//Recuperando o Locale
Locale locale = (Locale) request.getSession().getAttribute("org
   .apache.struts.action.LOCALE");

//Definindo um Locale
request.getSession().setAttribute("org.apache.struts.action.
   LOCALE", new Locale("pt", "BR"));
```

4.4 Pontos de decisão

Pontos de decisão são criados quando sua aplicação precisa tomar decisões dependendo do resultado de uma ação prévia. Utilizamos uma forma de modelagem ofericida pela especificação UML.

Nela você desenha uma linha de transição a partir da ação conectando-a ao ponto de decisão. Um ponto de decisão é desenhado como um losango em UML. Já que uma decisão tem pelo menos dois resultados diferentes, o ponto de decisão terá múltiplas transições para diferentes ações.

Usaremos como exemplo para introduzir um ponto de decisão o seguinte diagrama de atividades:

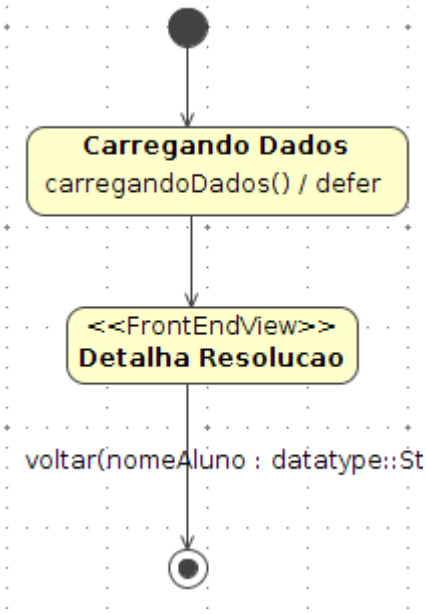

Figure 4.6: Modelo inicial do exemplo

Criamos um ponto de decisão e colocamos uma transição do estado **Carregando Dados** para o ponto de decisão:

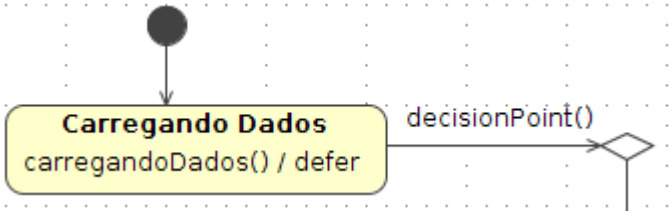

Figure 4.7: Criando o ponto de decisão

Em seguida na classe de controle do diagrama de atividades criamos uma função chamada `decisionPoint`:

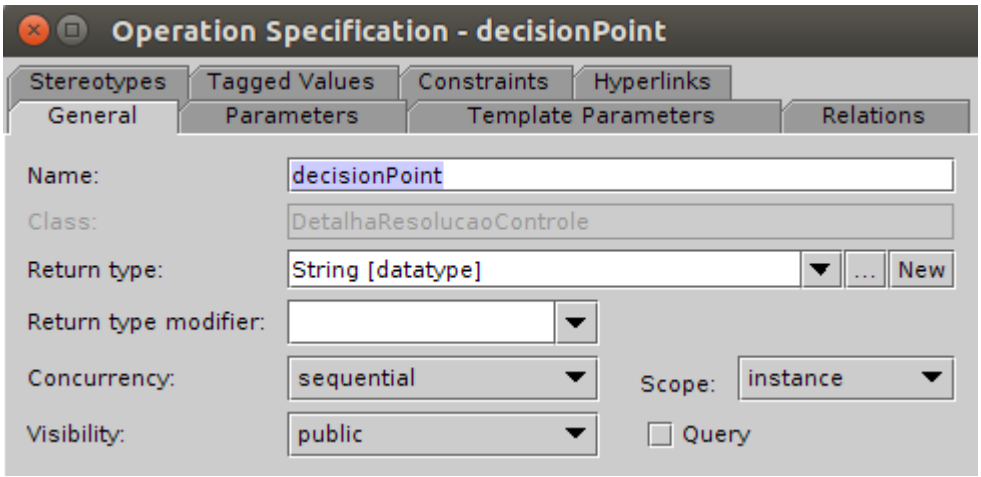

Figure 4.8: Criando função do ponto de decisão

Na transição para o ponto de decisão, alteramos a `trigger` e colocamos o tipo dela como `call` e a operação como `decisionPoint`:

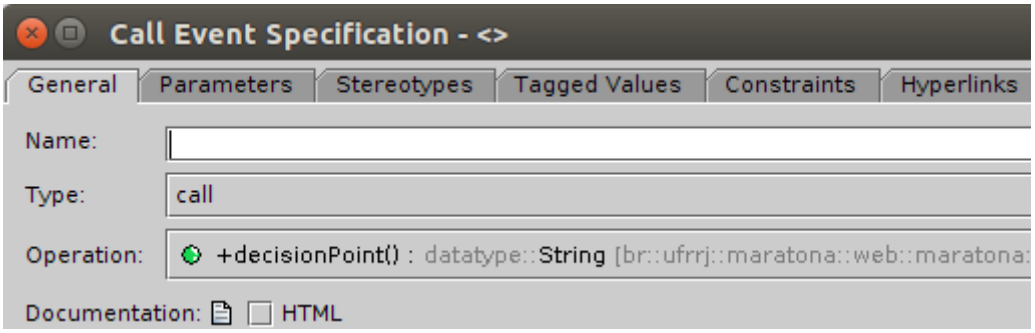

Figure 4.9: Colocando a função na transição

O próximo passo é passar o início da transição que ia de **Carregando Dados** para **Detalha Resolucao** para o ponto de decisão e criar uma transição do ponto de decisão para o estado final.

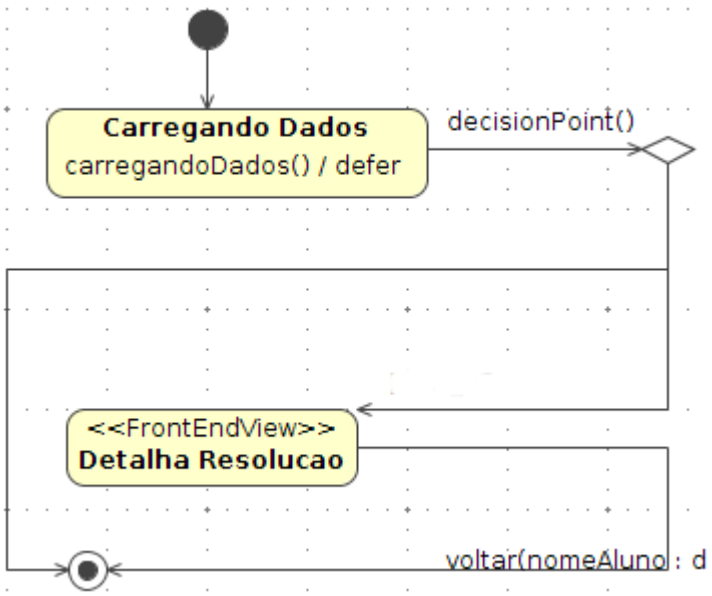

Figure 4.10: Colocando a função na transição

Em seguida devemos criar a condição de guarda dessas transições. Na aba `General`, clicamos no botão `Edit` da seção `Guard`:

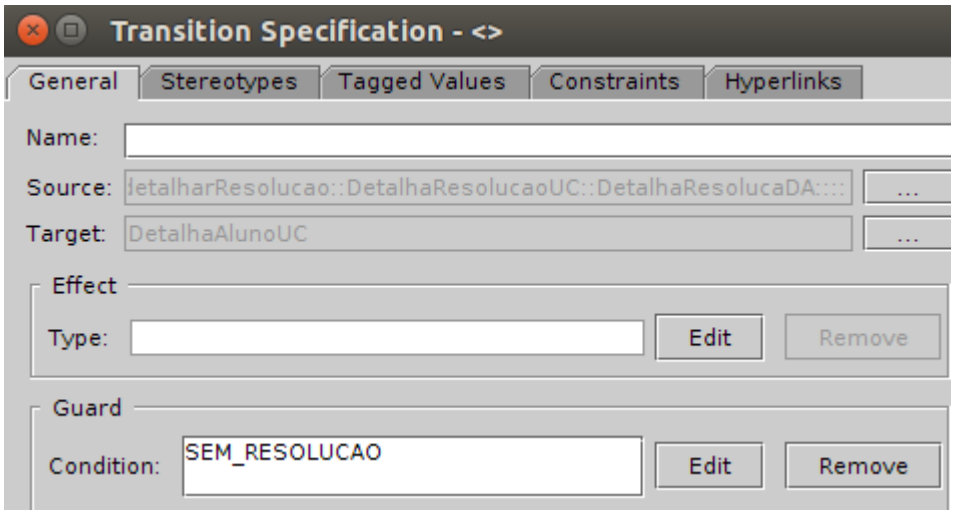

Figure 4.11: Abrindo a transição

Agora especificamos o nome da condição de guarda e a condição em si. Ambos tem que ter o mesmo nome:

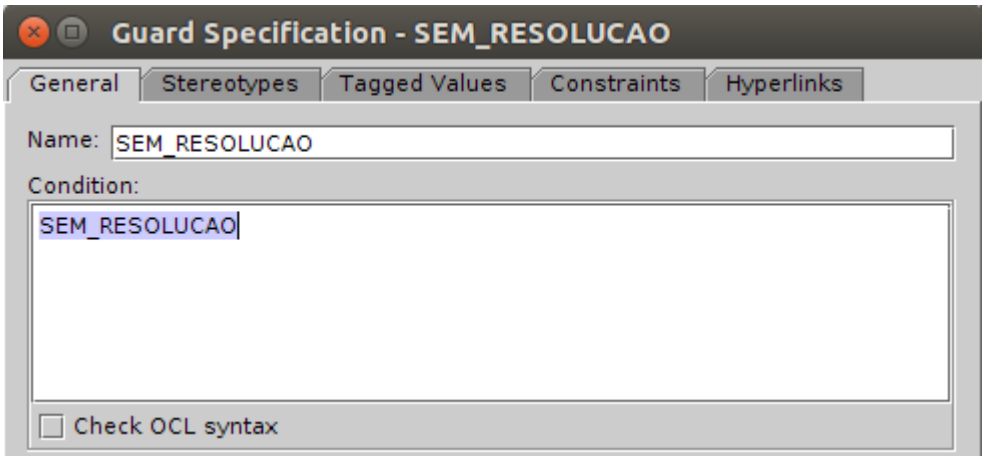

Figure 4.12: Especificando a condição de guarda

Fazemos o mesmo para a outra transição e o modelo final é o seguinte:

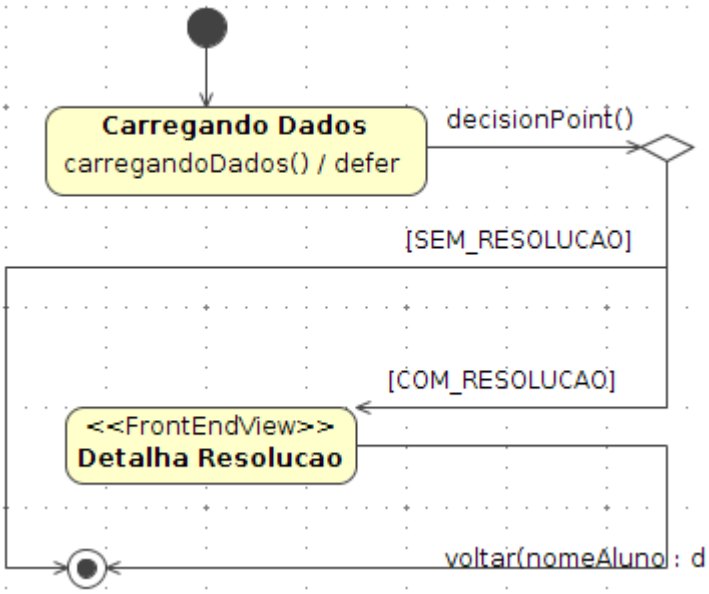

Figure 4.13: Modelo final

Agora temos que implementar a função `decisionPoint`. Ela tem que retornar uma String que seja uma das condições de guarda especificadas no modelo. Exemplo de código:

```
public String decisionPoint(DecisionPointForm form,
   ViewContainer container) throws Exception
{
      if (form.getIdResolucao() != null) {
            return "COM_RESOLUCAO";
      }
      else {
            return "SEM_RESOLUCAO";
      }
}
```

4.5 Tabela assíncrona (JTable)

Nesta seção veremos como implementar uma tabela `assíncrona` (`JTable`) usando o `MDArte`. Para tal, vamos considerar como ponto de partida o modelo do caso de uso `Consulta Estudante` conforme as alterações feitas no tópico anterior. Certifique-se de ter removido o estereótipo «`Manageable`» da classe `Estudante` no diagrama de classes que descreve a `Camada de domínio`, a fim de evitar que o `CRUD` seja re-gerado, sobrescrevendo assim as alterações que faremos.

Veremos agora, por subseções, algumas das funcionalidades disponíveis na tabela assíncrona.

4.5.1 Implementando uma tabela simples

Para implementar uma tabela assíncrona, precisamos primeiramente abrir o modelo do caso de uso `Consulta Estudante`, abriremos então a especificação da `transition` que sai da `action`

ConsultandoEstudante para Front End View 'ResultadoConsulta', clicar no botão edit no fieldset trigger, iremos então na aba parameters, na janela signal event specification, aberta automaticamente. Daremos então um duplo clique no nome do parâmetro (estudantes), que representa a tabela que será mostrada na view. Selecionaremos a aba tagged values selecione o tagged value @andromda.presentation.web.view.field.table.type e clique no botão create value. Selecione então a opção jtable, como na imagem 4.14.

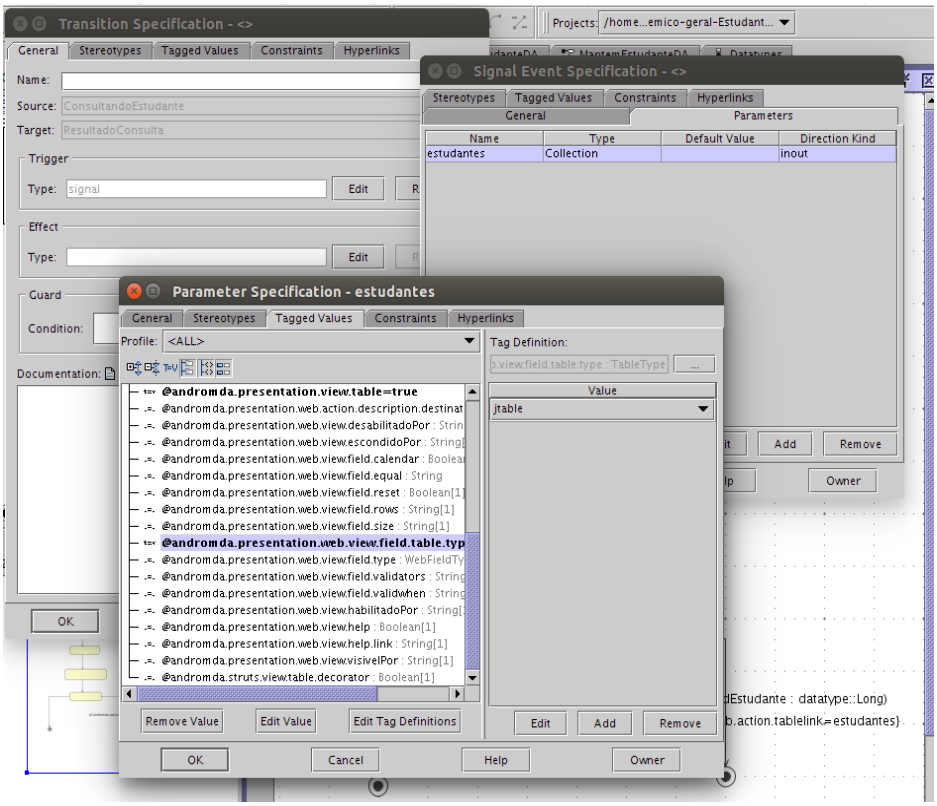

Figure 4.14: Mudando tipo da tabela para 'jtable'.

Agora executaremos o seguinte comando no terminal na raiz do projeto:

```
maven mda -Dprojeto=sistemaacademico-geral-Estudante
```

Feito isto, o MDArte já terá gerado toda a estrutura necessária pela recepção e tratamento das requisições assíncronas da tabela, restando ao desenvolvedor implementar somente dois métodos: um para indicar o numero total de elementos a serem exibidos na tabela, usado para fazer a paginação da mesma, e outro para retornar a coleção de objetos a serem exibidos na página atual da tabela.

A assinatura do método para o carregamento da tabela segue o seguinte padrão:

```
protected Collection load[nome-da-view][nome-da-tabela]Table(
    PaginationStrategy
paginacao, String propriedade, Boolean desc, ViewContainer container)
```

A assinatura do método que retorna o total de elementos na tabela segue o seguinte padrão:

```
protected Integer get[nome-da-view][nome-da-tabela]TableLength(
    PaginationStrategy
```

```
paginacao, String propriedade, Boolean desc, ViewContainer
container) throws Exception
```

Também será necessário alterar o método `carregaDados` do `ControllerImpl` para disponibilizar as informações preenchidas na `PreenchaCampos` para serem usadas pela tabela disponível na `view ResultadoConsulta`.

Abaixo o exemplo de implementação para a tabela `estudantes` da `view ResultadoConsulta` no caso de uso `Consulta Estudante`.

```
public final void consultaEstudante(br.mdarte.exemplo.academico.web.
   geral.consultarEstudante.ConsultaEstudanteForm form, ViewContainer
    container) throws Exception {
   form.setNome(form.getNome());
   form.setMatricula(form.getMatricula());
   form.setIdEstudante(form.getIdEstudante());
}

protected Integer getResultadoConsultaEstudantesTableLength(
   PaginationStrategy paginacao, String propriedade, Boolean desc,
   String nome,Long idEstudante,String matricula,ViewContainer
   container) throws Exception
{
      EstudanteTO estudanteTO = new EstudanteTOImpl();

      estudanteTO.setNome(nome);
      estudanteTO.setMatricula(matricula);


      return ServiceLocator.instance().getEstudanteHandlerBI().
         manipulaEstudante(new EstudanteImpl(), new
         DefaultFilterAction(estudanteTO, null)).size();
}

protected Collection loadResultadoConsultaEstudantesTable(
   PaginationStrategy paginacao,String propriedade,Boolean desc,
   String nome,Long idEstudante,String matricula,ViewContainer
   container) throws java.lang.Exception
{
      EstudanteTO estudanteTO = new EstudanteTOImpl();

      estudanteTO.setNome(nome);
      estudanteTO.setMatricula(matricula);

      Collection estudantes = ServiceLocator.instance().
```

```
        getEstudanteHandlerBI().manipulaEstudante(new EstudanteImpl
        (), new DefaultFilterAction(estudanteTO, paginacao));

    ArrayList<EstudanteVO> estudantesVO = new ArrayList<EstudanteVO
        >();

    for(Object object : estudantes) {
          EstudanteVO estudanteVO = new EstudanteVO();

          estudanteVO.setNome(((Estudante)object).getNome());
          estudanteVO.setMatricula(((Estudante)object).getMatricula
              ());
          estudanteVO.setIdEstudante(((Estudante)object).getId());

          estudantesVO.add(estudanteVO);
    }

    return estudantesVO;
}
```

Não se esqueça de fazer os `imports` necessários de acordo com as alterações feitas. Se o seu `.classpath` estiver corretamente configurado, você pode usar o atalho `Ctrl + Shift + O`, do eclipse, que importará automaticamente todas as classes que estão faltando.

Agora, compilaremos e daremos deploy do código da aplicação, com o seguinte comando:

```
maven compile deploy
```

Abriremos agora a view `PreenchaCampos` e digitaremos um valor para filtragem dos dados da entidade `Estudante`. Como na imagem 4.15.

Figure 4.15: Filtrando dados da tabela assíncrona de Estudantes.

Na imagem 4.16, podemos ver a tabela carregada com os dados resultantes da filtragens.

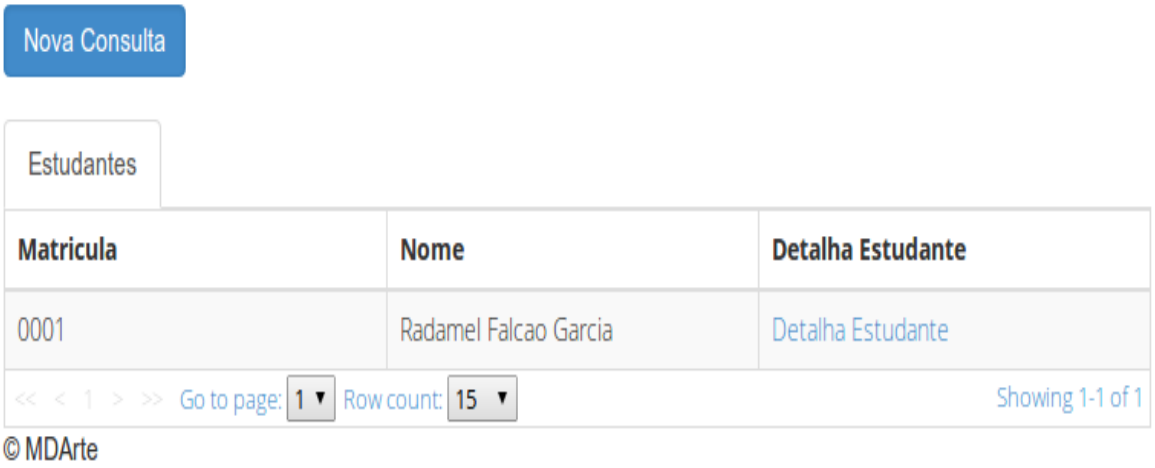

Figure 4.16: Filtragem dos dados da tabela assíncrona de Estudantes.

Note que o funcionamento da tabela se dá por requisições assíncronas ao servidor, acabando com a necessidade de recarregar a página para alterar seus dados.

O `MDArte` já dá suporte a diversos tipos de chamadas assíncronas que se possa querer fazer fazer com a tabela, no entanto, caso haja a necessidade de mais informações sobre o funcionamento da mesma, a documentação pode ser acessada aqui.

4.6 Criação de Componente customizado

Nesta seção veremos como criar um campo customizado em uma tela. Iremos partir do modelo de `CRUD` gerado automaticamente pelo `MDArte`, para a entidade `Estudante`, e faremos as alterações necessárias para a criação de um novo componente. O componente a ser desenvolvido, somente a título de exemplo, será um campo texto com máscara para `CPF`.

Podemos ver o estado inicial do modelo na imagem 4.17.

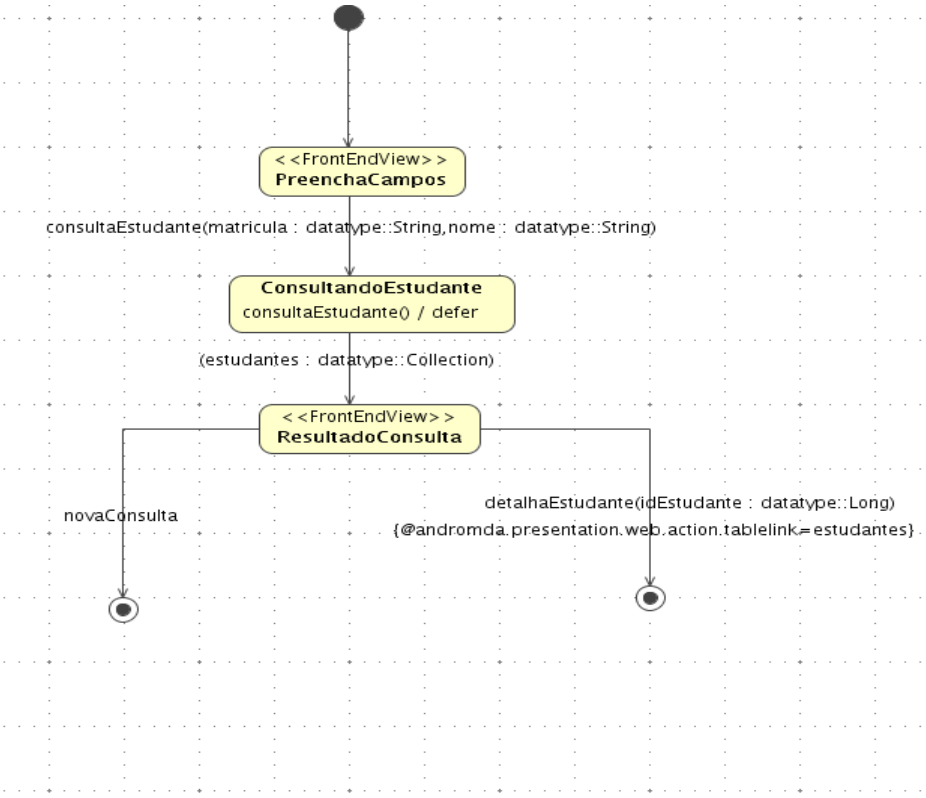

Figure 4.17: Modelo do caso de uso Consulta Estudante.

Abriremos então a especificação da `transition 'consultaEstudante'`, clicaremos no botão `'edit'`, no `fieldset 'trigger'`, selecionaremos então a aba `'parameters'`, da janela aberta quando clicamos o botão anterior, e clicaremos então no botão `add`.

Preencheremos então os dados do campo conforme a imagem 4.18, SEM, no entanto, clicar no botão `Ok`.

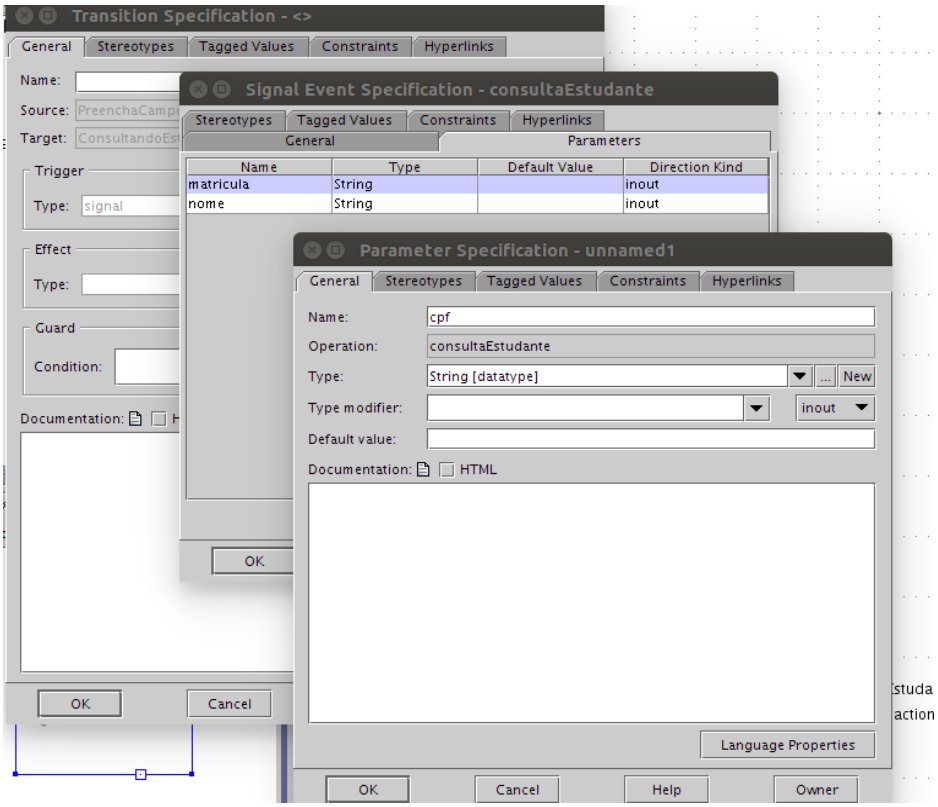

Figure 4.18: Dados do campo 'cpf'.

Ainda na mesma janela, selecionaremos a aba `tagged values`, selecionaremos o `tagged value @andromda.presentation.web.view.field.type`, clicaremos no botão `create value` e selecionaremos a opção `'custom'`, no campo `combobox` que será exibido, como na imagem 4.19.

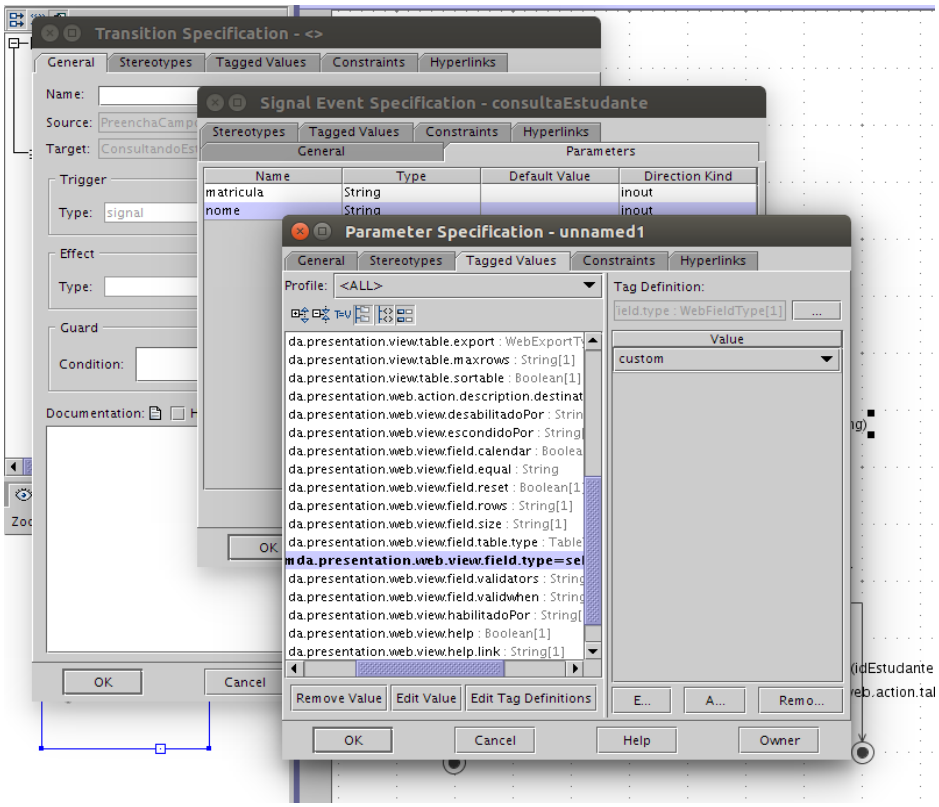

Figure 4.19: Selecionando tipo custom para o campo 'cpf'.

Salvaremos então o modelo e digitaremos os seguintes comandos para regerar o modelo:

```
maven mda -Dprojeto=sistemaaacademico-geral-Estudante
```

Feito isto, o `MDArte` gerará um arquivo no padrão `<nome-do-campo>.jsp`, neste caso, `cpf.jsp`, no caminho `<nome-sistema>/web/<modulo-web>/src/jsp/<caminho-do-pacote-base>/web/<modul` mas você também pode encontrá-lo, no eclipse, através do comando `ctrl+shift+r`, digitando o nome do arquivo na janela que é aberta por esse comando.

Aberto o arquivo, adicionaremos a este o seguinte código `jsp`:

```
<label class="control-label col-sm-2" for="cpf">
      <bean:message
      key="consulta.estudante.uc.preencha.
      campos.consulta.estudante.param.cpf"/>
</label>
<div class="col-xs-4">
      <s:textfield id="cpfConsultaEstudanteUC" name="cpf"
      label="%{getText('consulta.estudante.uc.preencha
      .campos.consulta.estudante.param.cpf')}"
      value="%{#form.$propertyValue}"
      cssClass="form-control" title=""
      styleId="$styleId" />
</div>
```

O `html` adicionado será importado para a tela do sistema no espaço do formulário dedicado ao campo `cpf`.

Adicionado o `jsp` do nosso componente, uma vez que se trata de um campo de texto com um determinado comportamento (formatar a entrada no modelo do CPF), precisamos agora adicionar um mecanismo de controle para o comportamento do campo. Para tal, utilizaremos o `framework` para `javascript` JQuery, uma vez que este já vem com o `MDArte`, além do fato de o `JQuery` já possuir uma funcionalidade nativa que faça isso.

Para adicionar código `Javascript` manualmente a uma `view` precisamos abrir o arquivo `<nome-da-view>-i` no mesmo caminho do arquivo `jsp` alterado acima. O arquivo alterado é destinado ao código adicionado pelo desenvolvedor para customizar o comportamento da aplicação, não sendo sobrescrito durante a geração. Certifique-se, portanto, de estar adicionando o seu código nestes pontos de implementação, a fim de não perdê-lo na próxima geração.

Adicionaremos agora o seguinte código `JavaScript` ao arquivo `preencha-campos-impl.js`:

```
$(document).ready(function(){
      $("#cpfConsultaEstudanteUC")
            .mask("999.999.999-99", { placeholder:"" });
});
```

Appendix A

Configuração do JBoss e acesso ao banco de dados

Neste apêndice veremos como configurar as informações de acesso ao banco de dados do nosso projeto, bem como demais configurações do nosso servidor de aplicação (`JBoss`).

A.1 Configuração das propriedades do projeto para acesso ao banco de dados

Para se configurar o Banco de Dados é necessário modificar o arquivo `project.properties` da raiz do projeto, onde se encontram as propriedades que devem ser alteradas. Os arquivos `project.properties` são arquivos onde são definidas propriedades que são usadas pelo `MDArte` durante a sua exucação, este, na raiz do projeto, especificamemte concentra propriedades de acesso ao banco e de `deploy` do projeto.

Abaixo estão as propriedades do arquivo de configuração para cada um dos Bancos de Dados:

Oracle

- dataSource.driver.jar=${env.JBOSS_HOME}/server/default/lib/ojdbc14.jar
- dataSource.driver.class=oracle.jdbc.driver.OracleDriver
- sql.mappings=Oracle9i
- hibernate.db.dialect=org.hibernate.dialect.Oracle9Dialect

SQLServer

- dataSource.driver.jar=${env.JBOSS_HOME}/server/default/lib/jtds-1.1.jar
- dataSource.driver.class=net.sourceforge.jtds.jdbc.Driver
- sql.mappings=MSSQL
- hibernate.db.dialect=org.hibernate.dialect.SQLServerDialect

Postgres

- dataSource.driver.jar=${env.JBOSS_HOME}/server/default/lib/postgresql.jar

- dataSource.driver.class=org.postgresql.Driver
- defaultHibernateGeneratorClass=sequence
- sql.mappings=PostgreSQL
- hibernate.db.dialect=org.hibernate.dialect.PostgreSQLDialect

MySQL

- dataSource.driver.jar=${env.JBOSS_HOME}/server/default/lib/mysql-connector-java-5.1.6-bin.jar
- dataSource.driver.class=com.mysql.jdbc.Driver
- defaultHibernateGeneratorClass=native
- sql.mappings=MySQL
- hibernate.db.dialect=org.hibernate.dialect.MySQLDialect

Tais propriedades são as responsáveis por definir qual banco de dados estará sendo usado no projeto, bem como qual biblioteca será usada para a comunicação com o banco. Propriedades como a `url` do banco, nome de usuário, senha etc. não precisam ser alteradas uma vez que, a seguir, as definiremos diretamente na configuração do `JBoss`.

A.2 Configuração do JBoss

Agora veremos como configurar o servidor `JBoss` e qual a finalidade dos arquivos utilizados para tal fim.

A.2.1 Configuração dos datasources utilizados pelo JBoss

Para a configuração dos datasources utilizados pelo `JBoss` é preciso criar ou alterar o arquivo responsável por registrar e gerenciar tais fontes de dados. O arquivo que deve estar localizado no diretório `JBOSS_HOME/server/default/deploy/`, com formação do nome terminando com `-ds.xml` (ex.: `aplicacoes-ds.xml`), que deve ter a `tag <local-tx-datasource>` preenchida de acordo com as informações fornecidas no arquivo `<projeto>/project.properties`.

Exemplo (usando banco `Postgres`):

```
<datasources>
      <local-tx-datasource>
            <jndi-name>sistemaacademicoDS</jndi-name>
            <use-java-context>true</use-java-context>
            <connection-url>
                  jdbc:postgresql://127.0.0.1:5432/
                     sistemaacademico
            </connection-url>
            <driver-class>org.postgresql.Driver</driver-class>
                  <user-name>usuario</user-name>
```

```
                <password>senha</password>
      <!--
            <exception-sorter-class-name>
                  org.jboss.resource.adapter.jdbc.vendor.
                     OracleExceptionSorter
            </exception-sorter-class-name>
      -->
      </local-tx-datasource>
</datasources>
```

Repare que no exemplo anterior, o nome do `Data Source` é `sistemaacademicoDS`, que deve ser o mesmo nome informado no arquivo `project.properties` no diretório raiz do projeto. Aqui também definimos algumas outras propriedades do datasource `sistemaacademicoDS` que haviam ficado em aberto antes como a `url` de conexão com o servidor de banco de dados, usuario e senha.

A.2.2 Configuração do acesso das aplicações ao banco de dados

Para tal, alteraremos o arquivo `login-config.xml`, localizado no diretório `JBOSS_HOME/server/default/c`
Alteraremos o arquivo adicionando uma tag `<application-police name='<nomeAplicacao>'>`
com seus campos devidamente preenchidos como no exemplo abaixo, onde temos a configuração para o `Sistema Acadêmico` deste tutorial.

```
<!--
SistemaAcademico Policy
-->
<application-policy name="sistemaacademico">
      <authentication>
            <login-module code="org.jboss.security.
               ClientLoginModule"
                  flag="required">
                  <module-option name="multi-threaded">
                        true
                  </module-option>
            </login-module>
            <login-module code="accessControl.LoginModuleImpl"
                  flag="required">
                  <module-option name="dsJndiName">java:/
                     controleacessoDS
                        </module-option>
```

```
                    <module-option name="unauthenticatedIdentity">
                       guest
                          </module-option>
                    <module-option name="principalClass">
                          accessControl.PrincipalImpl</module-option
                             >
                    <module-option name="hashEncoding">hex</module-
                       option>
                    <module-option name="hashAlgorithm">md5</module-
                       option>
                    <module-option name="principalsQuery">
            select SENHA from USUARIO where LOGIN=?
         </module-option>
         <module-option name="rolesQuery">
            select pf_usr.pf_FK, 'Roles'
                          from usuario, pf_usr
                          where LOGIN=? AND usuario.ID = pf_usr.
                             usr_FK
         </module-option>
            </login-module>
      </authentication>
</application-policy>
```

O arquivo alterado concentra as informações de login para as diversas aplicações sendo rodadas no servidor, informações como: datasource que contém os dados de usuário usados no `login`, `query` a ser rodada para buscar os dados de usuário, algoritmo de `hash` da senha etc. As informações presentes nesse arquivo permitirão a aplicação do `Sistema Acadêmico` se conectar a base de dados e validar o usuário no momento de `login`.

Appendix B

Configurando repositório externo do Maven

Appendix C

Alterações e cuidados a serem feitos ao migrar uma aplicação que utiliza a versão 17-RC9 para 19-RC9

C.1 Correção dos imports de classes do util

Essa alteração se deve ao surgimento de conflito entre pacotes do jar do controle de acesso que tem um pacote util próprio e o pacote util do projeto. Portanto, invés do pacote ser util, ele será <pacote do projeto>.util . Por exemplo:

```
import util.Constantes; //errado
import br.mdarte.exemplo.academico.util.Constantes; //correto
```

C.2 Remoção do pacote util antigo

Remover o pacote util antigo para evitar conflitos na compilação já que ele não é removido pelo maven.

C.3 Atualizar o Constantes.java

Adicionar as seguintes linhas no seu Constantes.java:

```
static public final String SISTEMA = SistemaAcademico; //aqui voce poe o valo
```

C.4 Cuidados com uso de select e double select

Na versão 17-RC9 parâmetros web, sem ter o valor etiquetado @andromda.presentation.web.view.field.type setado, eram gerados com variáveis extras no forms. Por exemplo:

```
private java.lang.Object[] mostrarAteValueList;
private java.lang.Object[] mostrarAteLabelList;
private java.lang.Object[] mostrarAteLabelListDouble;
private java.lang.Object[] mostrarAteLabelListHints;
private java.lang.Object[] mostrarAteLabelListDestination;
```

Estas variáveis são utilizadas para implementação do campo select ou dobleselect do Struts 1. Na versão 19-RC9 eles só serão gerados caso @andromda.presentation.web.view.field.type esteja setado como select ou doubleselect.

C.5 Adição do PaginationStrategy

Na versão 19-RC9 adicionamos a classe abstrata PaginationStrategy com o intuito de dar suporte a diferentes tipos de paginação, já que a paginação para um tipo de tabela pode não servir para outro tipo. Foram adicionados PaginationDisplaytag para tabelas normais, PaginationSimple para tabelas Ajax e NoPagination para caso uma paginação não seja especificada. Estas classes estão localizadas no pacote util do projeto.

Na versão 19-RC9, métodos de serviços que retornam Collection recebem como parâmetro PaginationStrategy. Na versão 17-RC9, recebiam como parâmetro um Integer. Por exemplo:

```
import br.mdarte.exemplo.academico.util.PaginationStrategy; //correto

public Collection handleExemplo(Integer paginacao) //errado
public Collection handleExemplo(PaginationStrategy paginacao) //correto
```

Em projetos que estão fazendo migração, verificar essas funções e os lugares em que elas são chamadas para evitar erros de compilação.

C.6 Adicionar novos valores no <projeto>/mda/project.properties

Adicionar as seguintes linhas no arquivo, já que ele e gerado apenas na criação do projeto:

```
maven.andromda.web.modulo.manual.jsp.dir=${maven.src.dir}/../../
   web/${maven.andromda.module.name.outlet.replace}/src/jsp

maven.andromda.web.custom.jsp.dir=${maven.src.dir}/../../web/src/
   jsp
```

C.7 Adição de propriedades no <projeto>/mda/conf/andromda.xml

Adicionar a seguinte propriedade nos namespaces ejb e bpm4struts:

```
<property name="controleAcessoImplJava"> {maven.andromda.web.
   manual.java.dir}</property>
```

C.8 Alterar build.properties

Adicionar ou alterar os seguintes atributos:

```
cartridge.version=3.1.1.3.4.19-RC9
security.version=1.1.3-RC1
commons.codec.version=1.4
json.simple.version=1.1
```

Não se esquecer de adicionar json-simple-1.1 e common-codec-1.4 no $JBOSS_HOME/server/default/lib.

C.9 Adição de depêndencias para o maven

Alterar <projeto>/common/project.xml, adicionado a seguinte dependência:

```
<dependency>
      <groupId>hibernate</groupId>
      <artifactId>hibernate</artifactId>
      <version>${hibernate.version}</version>
      <type>jar</type>
      <properties>
```

```
            <application.dependency>true</application.dependency>
        </properties>
</dependency>
```

C.10 Atualizar profiles da pasta <projeto>/mda/src/uml/xml.zips

Para utilizar os novos valores etiquetados, atualize os profiles substituindo-os pelos arquivos com os xml.zips gerados quando o comando maven foi executado no projeto. Eles estão localizados no repositório do maven que tem caminho <HOME>/.maven/repository/andromda/xml.zips.

C.11 Adição do LoginControllerImpl

Adicionamos LoginControllerImpl com função handlePosLogin(Operador operador, HttpServletRequest request). Ele substitui a função posLogin do ControleAcessoImpl, retire esta função do ControleAcessoImpl.

Local é <projeto>/web/modelo/compartilhado/src/java/<pacote do projeto setado na geração>/accessControl/LoginControl

Appendix D

Changelog

D.1 19-RC1

- Identação corrigida de vários arquivos
- Correção de vários bugs.
- Remoção de código inútil
- Alteração de nomes de macros dos templates.
- Retirada de constantes em alguns templates.
- Adição do campo AutoComplete.
- Adição de novs arquivos de JavaScript.
- Adição do BootStrap para páginas web.
- Adicionada a opção autocomplete, no WebFieldType do andromda-profile-presentation.
- Adicionado o atributo autocomplete no CoppetecStrutsParameter no BPM4StrutsMetafacadeModel Redirecionameno de módulos do struts 2 funcionando.
- Profile do bpm4Struts alterado, antes a opção web.view.technology.other estava vindo com valor default "true", mudado para "false".
- Adicionada dependência pro json simple.
- Corrigido bug em que actions que transicionavam para casos de uso de outro módulo não estavam tendo seu JavaScript gerado.
- Corrigido bug no redirecionamento entre casos de uso de versões diferentes do struts.
- Correção da geração do caminho da transição entre casos de usos.
- Corrigido a transição de struts 2 para struts 1.
- Adição do DefaultFilter

- Templates que geram páginas web para cada tipo de Struts.
- Arquivos gerados com nome Action2 tem o mesmo nome de Action de Struts 1.
- Obter o request e o response do ViewContainer é possível
- Corrigido bug que impedia a produção do page-actions se o caso de uso só tiver ações como tablelinks.
- Adição do custom component

D.2 19-RC2

- Correções de formatação de código.
- Adicionando propriedade controleAcessoImplJava no andromda.xml.
- Adicionando dependências que faltam ao cartucho hibernate.
- Retirando a geração de dummys nos pontos de implementação.
- Corrigido bug de geração de tabelas repetidas.
- Corrigido erro no setter de campos do tipo hidden.
- Aprimorando geração do classpath do eclipse.
- Organizando a geração de projetos renomeando-os com extensão vsl.
- Geração de projetos tem uma nova pergunta para especificar o banco de dados a ser usado.
- Corrigido bug do hidden parameter.
- Adicionando pergunta para oAuth e corrigindo dependências e imports relacionados.
- Correção do destino da geração dos arquivos OAuth.
- Corrigindo tela de login do Oauth.
- Tabelas sendo mostradas com abas.
- Corrigido bug na hora de demonstrar textfields com valor já setado previamente.
- Protótipo do CRUD funcionando.
- MArteManageableModelsLogicImpl implementado.
- Nomes do arquivos gerados pelo CRUD seguem um padrão.
- Adição do valor etiquetado @andromda.manageable.web.crudPackageName para o estereótipo Manageable Gerando xml.zips corretos na hora da geração de projetos novos.
- Corrigido bug de não gerar page-fields tendo apenas tablelinks.

- Corrigido bugs de geração de páginas web para o Struts 2.
- Sistema de erros melhorado.
- Corrigido layout do login e stacktrace.
- Adicionados os campos webtype no arquivo na geração.
- Corrigindo bug na hora da geração de actions por causa de controller sem operações.
- Corrigido geração de forms ao impedir geração de código inútil. Template Form.java.vm criado.
- Implementação do Remember Me.

D.3 19-RC3

- Retirada de código inútil.
- Corrigindo posicionamento de elementos de páginas web.
- Consertando identação de vários templates.
- Geração dos forms não gera mais bugs.
- Migrando autocomplete do bootstrap para typeahead.js.
- Troca de Senha atualizada para o bootstrap.
- Error page adaptada para bootstrap.
- Bug na passagem de parametros pelo nextPath corrigido.
- Pagina de login migrada para o bootstrap 3.0.3
- Migrando páginas de segurança para o bootstrap 3.
- Adicionando page-javascript.jspf.vsl para o struts 2.
- Atualizando formulário pro bootstrap 3.
- Corrigido bug de passagem de parâmetros.
- Messages migrado pro bootstrap 3.
- Titulo da página de login customizável por bean message.
- Nomes de tabelas customizáveis por bean messages.
- Tabela já sendo carregada com a primeira aba selecionada e marcação das abas corrigido.
- Atualizando classes das divs para bootstrap 3.
- Atualizando campo de data.

- Consetando layout das paginas e ajustando geração do blockquote para campos required para só gerar se houver um campo deste tipo.
- Imports desnecessários removidos no ControllerImpl do struts 2.
- Bug quando você seleciona um enumeration como atributo de uma trigger no modelo consertado.
- Select construido automaticamente a partir de um atributo enumeration.
- ControllerImpl gerando os métodos para tratar o autocomplete.
- Consertando bug no taggedvalue readonly.
- Geração de enumerados corrigida.
- Bug do custom-resources dos enumerados corrigido.

D.4 19-RC4

- Nova tag @andromda.common.enumeration.emptyValue que indica se o enumerado é gerado com um valor vazio default
- Criado o valor etiquetado @andromda.presentation.web.view.field.enumeration.emptyValue que irá definir se o enumerado gerado automaticamente será com vazio ou não. Colocado uma constraint que essa opção só será colocada se for um select e um enumerado.
- Layout personalizado (layout e o menu) para os casos de usos open access
- Criação de um ControllerAbstract para agrupar os métodos comuns aos controles Mudança dos pacotes do Util.java para não entrar em conflito com outros projetos
- Criação do UtilAbstract.java para criação de novos métodos auxiliares e não ter impacto nos projetos Correção de um bug no select no struts 2 onde tinha um valor vazio extra
- Criação de métodos auxiliares para o Enumerados
- Criação de métodos auxiliares para recuperar o resource para o struts 2
- Correção da tela de trocar-senha para o struts 2
- Correção do bug que não estava guardando a senha quando o remember me é ativado Adicionando div para tags p
- Correção de um bug de geração do tiles para struts2
- Correção do bug de caracteres estranhos na tabela
- Correção do estilo do typehead
- Correção da geração do CRUD para enumerados
- Correção do bug no struts2 que não exibia as exceptions
- Função isTrue(String) adicionada no UMLMetafacadeUtils.

D.5 19-RC5

- Adicionado o componente editor wysiwyg
- Correção do bug do caminho do botão em um módulo principal
- Adicionando diversos métodos para controle do modo operação no ControllerAbstract Correção do bug onde no Controle do Struts2 estava sendo gerado com .do
- Identação de diversos arquivos
- Correção de um bug que não ia para a próxima página em uma tabela
- Correção de alguns acentos no displaytag

D.6 19-RC6

- Correção da localização do pacote util
- Implementação do Controle de Acesso 2.0
- Correção do Servicos.sql para o novo controle de acesso
- Adição do valor etiquetado @andromda.presentation.web.view.table.type que define o tipo da tabela gerada Adicionado no Servicos.sql as querys para criar usuario, perfil e sistema
- Aprimoramento no Servicos.sql para gerar a sequence corretamente
- Correção do classpath para o novo controle de acesso
- Correções no componente multibox
- Correções no componente doubleselect
- Correções da tagged values readonly para alguns componentes
- Correções de indentação
- Correções de algumas máscaras no struts 2 (int, double, float e date)
- *** Adicionando uma biblioteca jquery.mask. Deverá adicionar a biblioteca no includes-javascript Adicionado métodos no UtilAbstract
- Correção no número fixo no project.xml
- *** No project.xml da raiz no compartilhado está fixo o valor 1.0. Trocar para ${application.version}

D.7 19-RC7

D.7.1 Aprimoramentos

- Nova tabela utilizando carregamento através de AJAX através do valor etiquetado @andromda.presentation.web.action.aj
- Implementação de hiperlink da tabela AJAX
- Implementação de imagem com link na tabela AJAX
- Adicionado nova estrutura para os métodos do DAO para passar a paginação *

D.7.2 Correções

- Correções de dependências de JS
- Correções de CSS **
- Correções do LookUpGrid para o Struts 2
- Correções do popup para o Struts 2
- Correções na geração do Servicos.sql
- Correções de validação do campo money, float e outros
- Correção do bug no carregar de um caso de uso não exibir a exception correta (Struts 1 e 2) Correção da validação por e-mail, cartão e range para o Struts 2.
- Correção do ControleAcessoImpl de não pegar os perfis filhos
- Atualizando jquery para versão 1.9, pois a 1.7 é incompatível com o jquery.jtable Removendo dependências desnecessárias (jquery-hashchange)
- Correções de multibox e double select
- Remoções de códigos não utilizados
- Correções feitas na geração de funções com o valor etiquetado @andromda.persistence.DAOMethod Correções de alguns bugs dos enumerados para serem preenchidos automaticamente na tela

* - Terá que adicionar os campos novos nos métodos do DAOImpl
** - Foi removido a classe row dos fields

D.8 19-RC8

- Adicionando constantes TABLE_LINES e TABLE_PAGES
- Atualizando classpath do eclipse para struts 2.
- Assinatura de funções que retornam Collections tem linhas e paginas como parametros adicionados. (removido) Removendo imports desnecessários

- Adicionando referencia utilDir ao cartucho do hibernate.
- Corrigindo bug de perda de parametros durante o login.
- Corrigindo bug do campo radio para struts 2
- Corrigindo identação
- Criando valor default para quando tiver uma opção default para o radio button Indicando qual o template que origina o artefato.
- Corrigindo endereço do DOCTYPE dos arquivos de configuração do hibernate
- Melhorias feitas na geração do radio button.

D.9 19-RC9

- Adicionando no pacote util a classe abstrata PaginationStrategy. Ela tem a função de auxiliar a paginação de tabelas. Com essa classe podemos ter diferentes estratégias de paginação para tabelas diferentes ao invés de apenas uma estratégia de paginação em releases anteriores.
- Adição do método paginateList no UtilAbstract.
- Corrigido bug de validação do struts 1.
- Corrigido bug na geração do Controllerimpl.
- Corrigido bug da geração de arquivos de configuração do hibernate sem mapeamentos.
- Correção de bugs na geração do CRUD.
- Correção de bugs da tabela AJAX.

Bibliography

[1] Albert Einstein. Zur Elektrodynamik bewegter Körper. (German) [On the electrodynamics of moving bodies]. *Annalen der Physik*, 322(10):891–921, 1905.