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
- JBoss (versão 4.2.3-GA)
- Maven (versão 1.0.2)
- Magic Draw (versão 9.5)
- Eclipse Indigo (versão 3.7.1)

1.1 JDK

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

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

1.2 JBoss

É necessário que o JBoss esteja instalado no computador. O download pode ser feito em http://www.jboss.com/. Após a instalação do JBoss, é necessário configurar a variável `JBOSS_HOME`, a qual deve especificar o caminho de instalação do JBoss. Esse caminho para o JBoss é necessário para a instalação de aplicações (deployment).

Neste tutorial será considerado que o jboss estará instalado na pasta abaixo.

```
/home/<user>/Work/programs/jboss
```

1.3 Maven

O Maven[1] é uma ferramenta de automação e gerenciamento de projetos. O download pode ser feito através do endereço http://maven.apache.org/start/download.html e sua instalação consiste em descompactar o arquivo obtido em um diretório local.

A versão compatível 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.

Para especificar o repositório que o Maven deve acessar é necessário criar um arquivo, chamado `build.properties`, no diretório home do usuário, por exemplo `/home/<usuario>`.

Abaixo temos exemplos do arquivo build.properties para uso com repositórios externos, em repositório externo que possui um proxy para acesso à internet, e com um repositório proxy configurado na rede local.

1. Repositórios remoto quando não é necessário utilizar proxy para acesso à internet:

```
maven.repo.remote=http://www.ibiblio.org/maven,http://team.andromda.org/maven
```

2. Repositórios remoto quando é necessário utilizar proxy para acesso à internet. No exemplo, o IP do proxy é `10.0.2.15`:

```
maven.repo.remote=http://www.ibiblio.org/maven,http://team.andromda.org/maven maven
maven.proxy.port=8080
```

3. Repositório proxy na rede local. O repositório proxy contém as bibliotecas `*.jar` que o Maven poderia requisitar no repositório remoto na Internet. Esse repositório deverá ser utilizado quando não é possível ou não é desejável o acesso ao repositório remoto:

```
maven.repo.remote=http://<host>:<port>
```

Onde `<host>` é o nome da máquina utilizada como proxy e `<port>` é o número da porta do serviço do proxy. Por exemplo: `maven.repo.remote=http://146.164.34.92/repositorio/`

É importante observar que o Maven, de acordo com as tarefas executadas, irá fazer o download dos artefatos necessários e guardá-los em um cache local na estação de trabalho em um diretório chamado `.maven` localizado no diretório home de cada usuário da estação.

Para evitar o acesso a servidores na Internet é possível a instalação de um proxy específico do Maven na rede local. Neste tutorial de configuração, não abordaremos a configuração desse proxy.

Neste tutorial será considerado que o jboss estará instalado na pasta abaixo.

```
/home/<user>/Work/programs/maven
```

1.4 Varíaveis de Ambiente

Adicionar no final do arquivo `/home/<usuario>/.bashrc` o seguinte código:

```
if [ -f ~/.bashrc_mdarte ]; then
 . ~/.bashrc_mdarte
fi
```

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

Criar o script onde ficará todas as variáveis do ambiente.

```
touch ~/.bashrc_mdarte
```

Editar o arquivo /home/<usuario>/.bashrc_mdarte adicionando os seguintes valores.

```
#MDArte Configurations

export MAVEN_OPTS=-Xmx1024m

export JAVA_HOME=/usr/lib/jvm/java-6-oracle/

if [ -d ~/Work/programs/maven ] ; then
  export MAVEN_HOME=~/Work/programs/maven
fi

if [ -d ~/Work/programs/maven ] ; then
 export PATH=\$PATH:~/Work/programs/maven/bin
fi

if [ -d ~/Work/programs/jboss ] ; then
  export JBOSS_HOME=~/Work/programs/jboss
fi
```

Após será necessário reiniciar a sessão do usuário para essas variáveis estarem no sistema ou utilizar o seguinte comando abaixo.

```
source ~/.bashrc
```

1.5 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-
```

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.

1.6 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”.

1.7 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

Desenvolvimento de Projetos com o AndroMDA

Nesta seção veremos os passos necessários ao desenvolvimento de projetos com o MDArte, utilizando os cartuchos EJB, Hibernate e BPM4Struts.

2.1 Criação de um Novo Projeto e Configuração do ambiente

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. Na verdade, o projeto será gerado em um subdiretório do diretório escolhido. No Windows, não se pode ter espaços em branco no caminho desse diretório. Exemplo de diretório inválido:

   C:\Documents and Settings\MDArte.

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

3. Responda as perguntas de acordo com o seu projeto. Abaixo um exemplo com respostas típicas (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.

2.2 Configuração do Banco

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. 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-8.1-405.jdbc3.jar
- dataSource.driver.class=org.postgresql.Driver
- defaultHibernateGeneratorClass=sequence
- sql.mappings=PostgreSQL
- hibernate.db.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

Outro arquivo que deve ser alterado ou criado é o arquivo de configurações do Banco de Dados do JBoss, 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):

```
<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>
```

Repare que no exemplo anterior, o nome do Data Source é sistemaacademicoDS, que deve ser o mesmo nome informado no arquivo project.properties da raiz do projetom ou seja ‘sistemaacademicoDS‘.

Além disso, o arquivo `login-config.xml`, localizado no diretório `JBOSS_HOME/server/default/conf/`, deverá ser modificado, adicionando-se a ele o seguinte trecho:

```
<!--
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 OP_C_A where LOGIN=?
                        </module-option>
                  <module-option name="rolesQuery">
                        select OP_PF.PF_OP_C_A_FK, 'Roles'
                        12from OP_C_A, OP_C_A_PF_OP_C_A OP_PF
                        where LOGIN=? AND OP_C_A.ID = OP_PF.
                           OP_C_A_FK
                  </module-option>
            </login-module>
```

```
      </authentication>
</application-policy>
```

As informações presentem nesse arquivo permitirão a aplicação do Sistema Acadêmico se conectar a base de dados e validar o usuário no momento de login.

2.3 Controle de Acesso

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, pelo endereço `https://github.com/MDArte` . Por fim, edite também o arquivo `project.properties` do ControleAcesso para configurar o tipo de Banco de Dados a ser utilziado, conforme realizado com o projeto SistemaAcademico. Note que a propriedade dataSource.name está definida como ControleAcessoDS.

Novamente, precisaremos criar um arquivo de configuração do Banco de Dados, 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:

```
<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>
```

Note que no exemplo anterior o ControleAcesso estará utilizando a mesma base de dados do projeto SistemaAcademico, definida pela tag <connection-url>. 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/`:

```
maven mda -Dprojeto=ca-core
cd common
maven jar:install deploy
cd ../core/cd
maven jar:install deploy
```

2.4 Modelando o nosso primeiro projeto

Nesta seção iremos modelar um exemplo de Sistema Acadêmico básico, 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/`

2.4.1 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. Essas classes deverão conter o estereótipo «Entity» e os atributos que serão persistidos. Todas as classes de entidade devem obrigatoriamente estar no pacote <PacoteProjeto>.cd, em que <PacoteProjeto> é o pacote definido para o projeto.

Atualmente, estamos utilizando framework Hibernate para esta camada.

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 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 2.1.

Figure 2.1: 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 2.2.

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".

Figure 2.2: Criação do diagrama de classes da camada de domínio.

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 2.3.

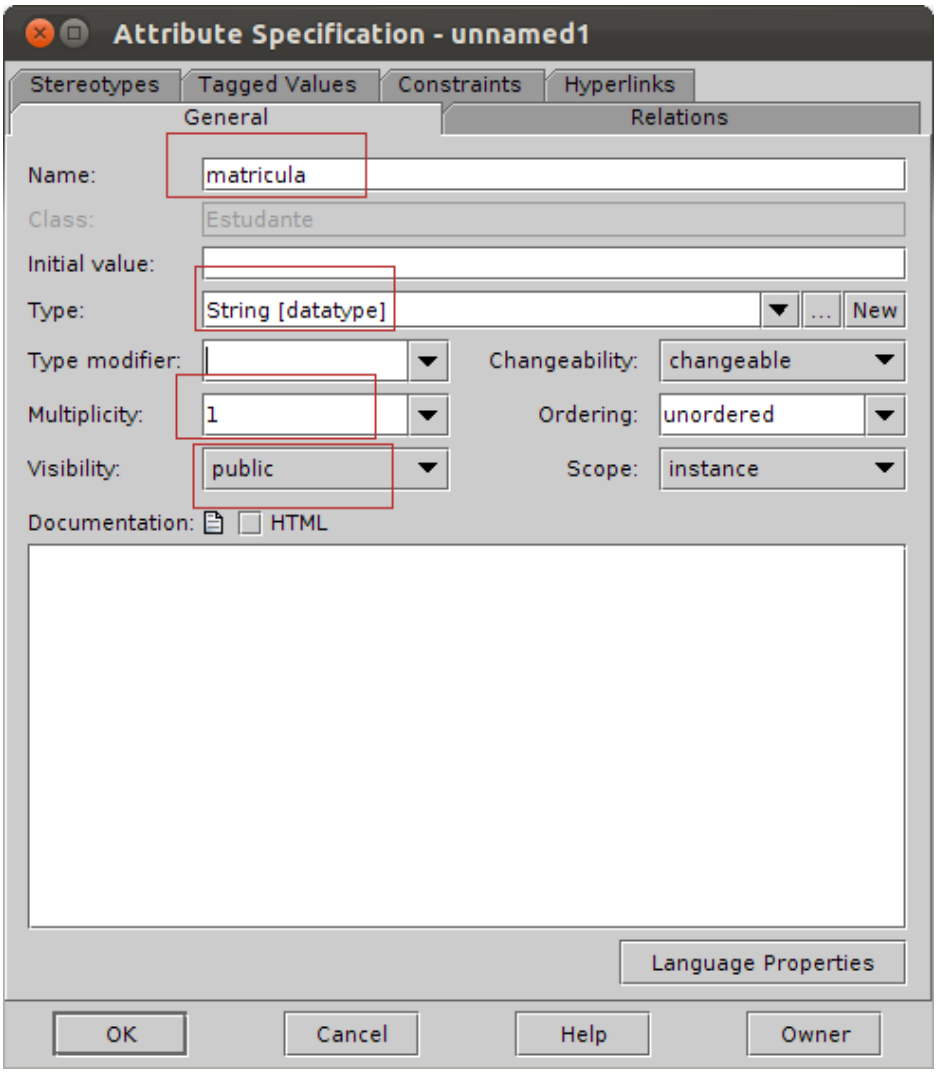

Figure 2.3: Configuração do parâmetro matrícula da classe Estudante.

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 2.4.

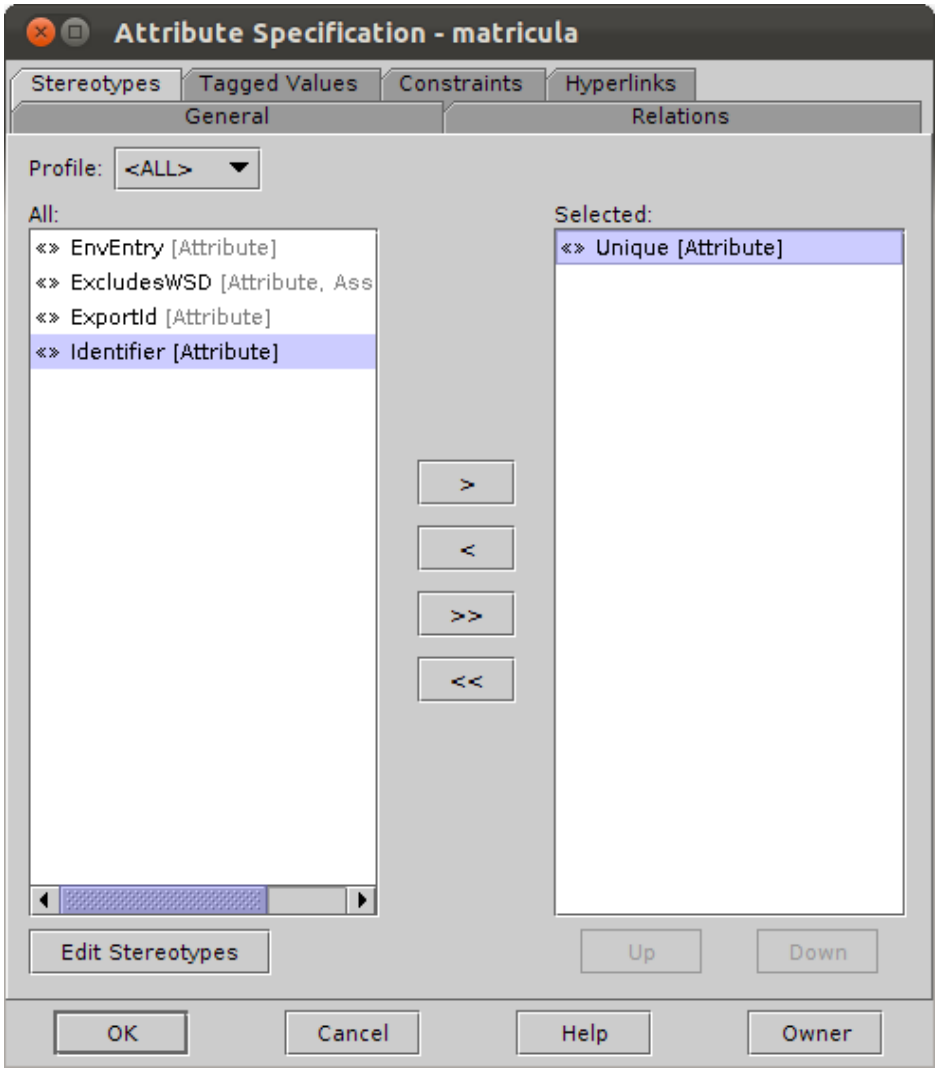

Figure 2.4: Adição de estereótipo no atributo matrícula.

7. Coloque os estereótipos «Entity» e «Manageable» na classe Estudante, como na figura 2.5.

   Para cada entidade, também podem ser atribuídos valores etiquetados para agregar ao modelo parâmetros para a geração de código. Por exemplo, o valor etiquetado @andromda.persistence.table reflete o nome da tabela a ser criada no Banco de Dados. Da mesma forma, podemos atribuir estereótipos e valores etiquetados aos atributos. Entre os estereótipos temos: «Identifier» que determina que o atributo será o identificador do objeto (possível chave primária) e «Entity» que determina que o valor do atributo deverá ser único.

   Como exemplo de valores etiquetados temos `@andromda.persistence.column` que define o nome da coluna a ser criada no Banco de Dados e `@andromda.persistence.column.lenght` que define o tamanho da coluna.

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”.

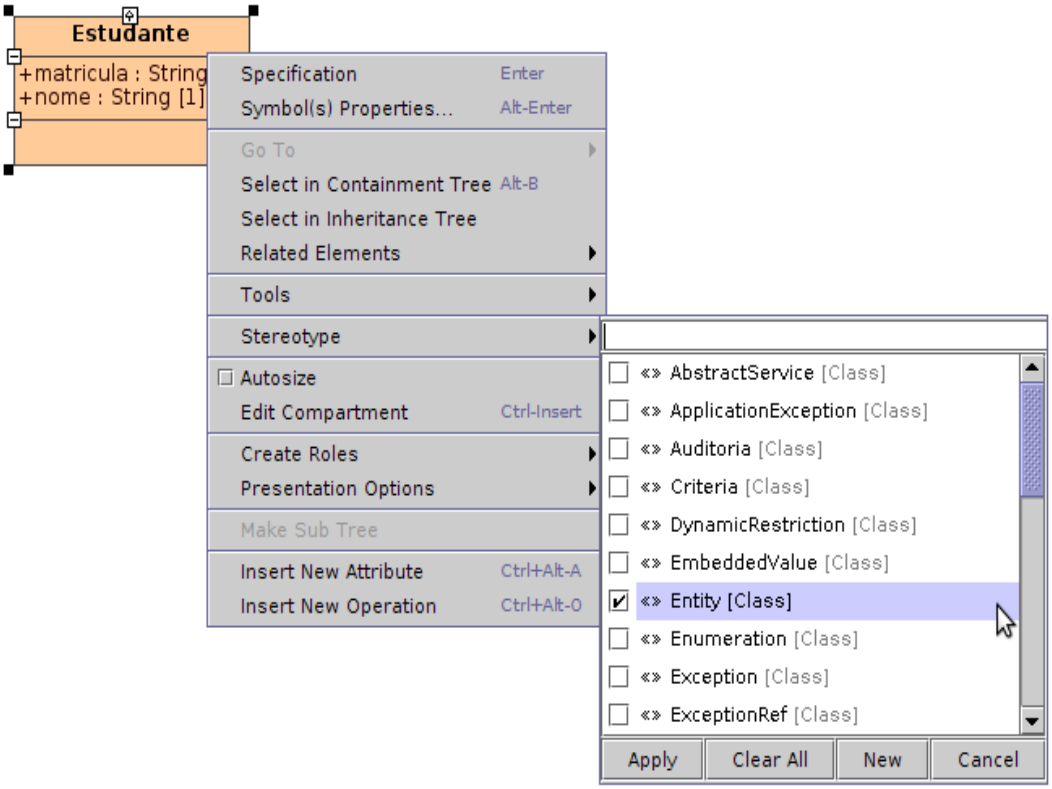

Figure 2.5: Adição dos estereótipos Entiti e Manageable na classe estudante.

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 o estereótipo «Entity» 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 2.6.

14. A associação deve ser dupla, tanto Estudante quanto Curso devem ser visíveis. Dessa forma, mantenha a checkbox Navigable marcada na associação para as duas classes. Para isso, clique no botão "..." (reticências) da tela anterior, como na imagem 2.7.

    O resultado final pode ser visto na imagem 2.8:

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".

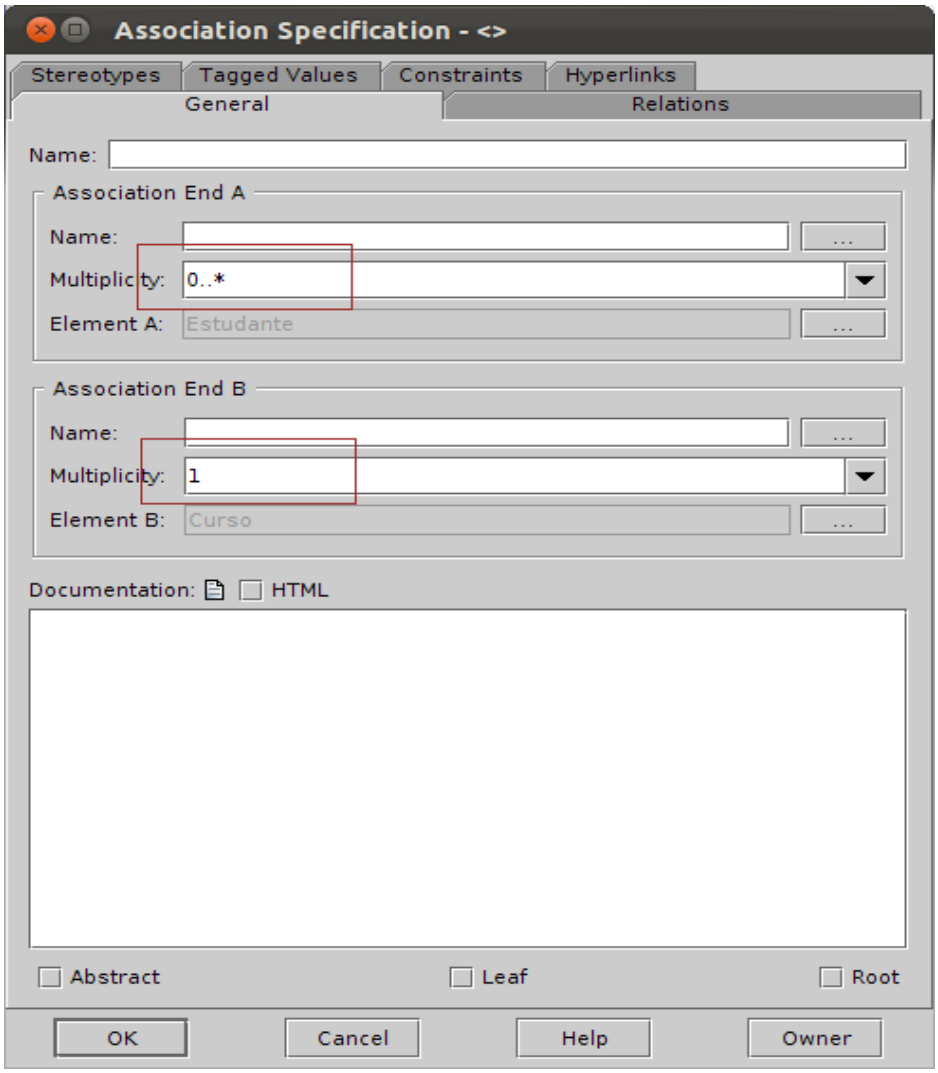

Figure 2.6: Definindo multiplicidade da associação.

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.

    Para compilar e gerarmos estes módulos separados executaremos os comandos:

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

2.4.2 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.

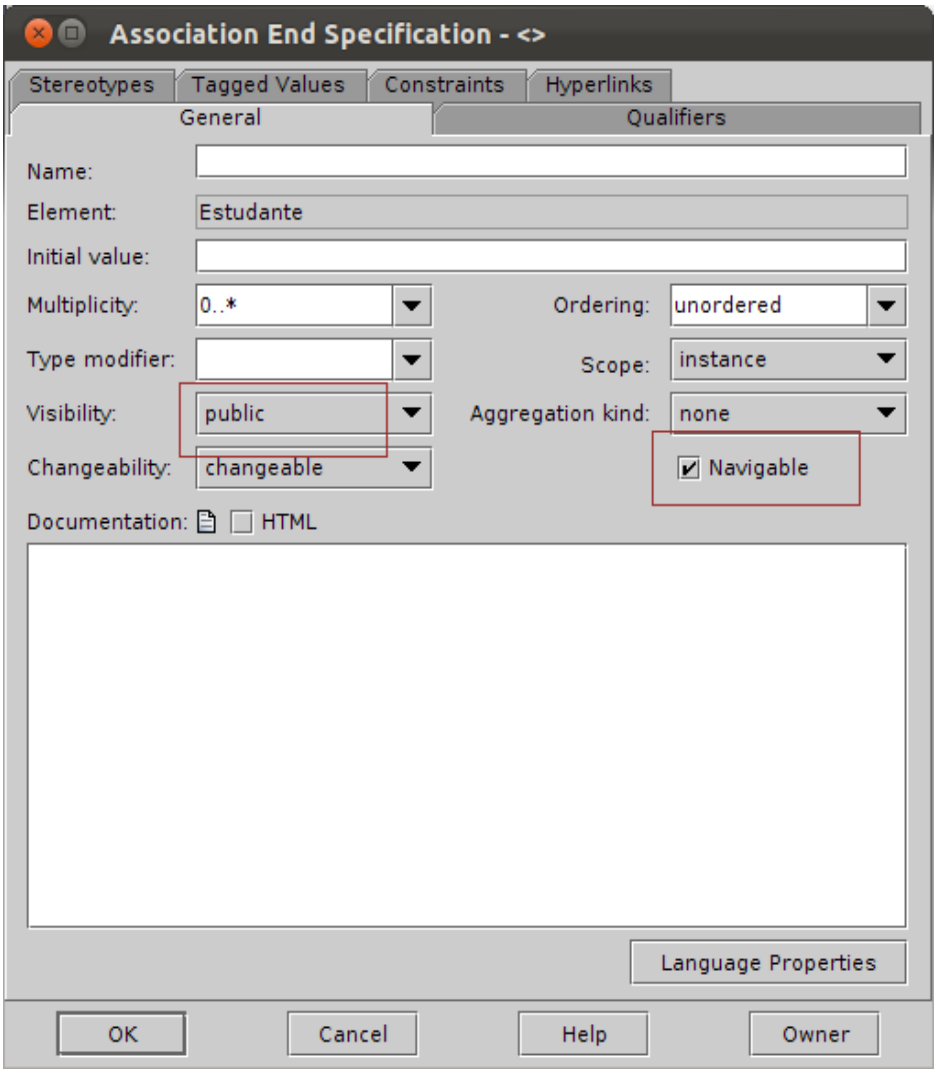

Figure 2.7: Configuração da navegabilidade da associação.

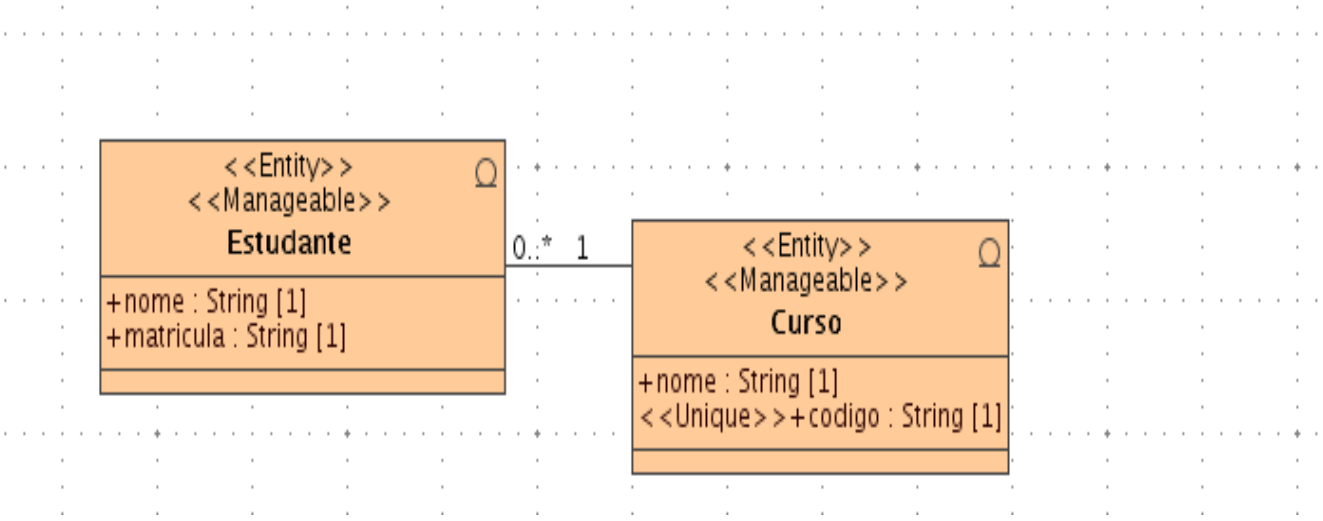

Figure 2.8: Resultado final da modelagem da camada de domínio.

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);
```

2.4.3 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 2.9:

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 2.10.

4. Crie uma classe com nome EstudanteException e estereótipo «ApplicationException», como na imagem 2.11.

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

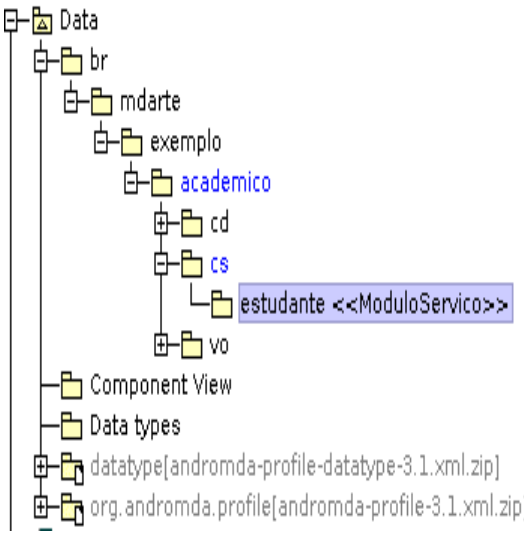

Figure 2.9: Criação do pacote para a camada de serviço.

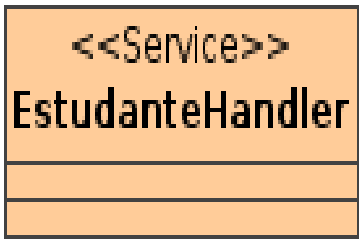

Figure 2.10: Criação da classe de serviço EstudanteHandler.

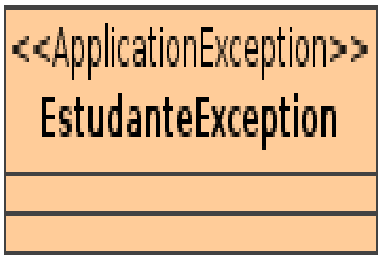

Figure 2.11: Criação da classe estudante exception.

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 2.12.

7. Verifique se o diagrama está como a figura 2.13.

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

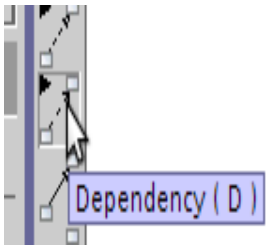

Figure 2.12: Criação da dependência entre as classes do serviço.

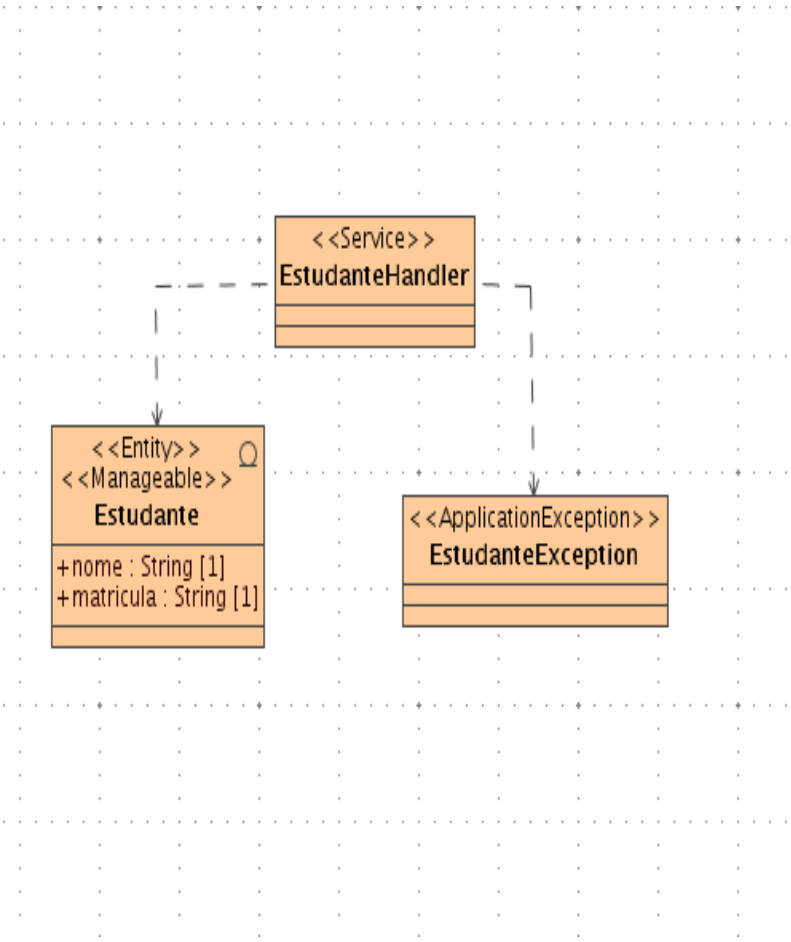

Figure 2.13: Diagrama de classe completo do módulo de serviço para a classe Estudante.

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 Estudante.

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”.

2.5 Implementando as classes de controle dos cruds 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 cruds 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 :

```
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();

   EstudanteVO vo = new EstudanteVO();

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

   // nothing to be done for this operation,
   //there are no properties that can be set

   Collection estudantes =
      ServiceLocator.instance().getEstudanteHandlerBI()
      .manipulaEstudante(new EstudanteImpl(),
      new FilterAction(vo, paginacao));

   form.setEstudantes(estudantes);
}
```

2. MantemEstudanteControleImpl.java :

```
    public final void carregaEstudante(
                br.mdarte.exemplo.academico.web.geral.
                manterEstudante.CarregaEstudanteForm form,
                ViewContainer container) throws Exception {

                Estudante e = new EstudanteImpl();

                e.setId(form.getId());

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

                form.setNome(e.getNome());
                form.setId(e.getId());
                form.setMatricula(e.getMatricula());

          }

          public final void salvaEstudante(
                      br.mdarte.exemplo.academico.web.geral
                      .manterEstudante.SalvaEstudanteForm form,
                      ViewContainer container) throws Exception {

                Estudante e = new EstudanteImpl();

                e.setId(form.getId());
                e.setMatricula(form.getMatricula());
                e.setNome(form.getNome());

                ServiceLocator.instance().getEstudanteHandlerBI().
                      insertOrUpdateEstudante(e);

          }
```

3. DetalhaEstudanteControleImpl.java:

```
    public final void carregaEstudante(
          br.mdarte.exemplo.academico.web.geral.detalharEstudante
```

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

        Estudante e = new EstudanteImpl();

        e.setId(form.getId());

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

        form.setNome(e.getNome());
        form.setId(e.getId());
        form.setMatricula(e.getMatricula());

    }
```

4. ConsultaCursoControleImpl.java :

```
    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();

       CursoVO vo = new CursoVO();

       vo.setNome(form.getNome());
       vo.setCodigo(form.getCodigo());

       // nothing to be done for this operation,
       //there are no properties that can be set
       Collection cursos = ServiceLocator.instance()
          .getCursoHandlerBI().manipulaCurso(
          new CursoImpl(), new FilterAction(vo, paginacao));

       form.setCursos(cursos);
```

```
}
```

5. MantemCursoControleImpl.java :

```
public final void carregaCurso(
      br.mdarte.exemplo.academico.web.geral.manterCurso
      .CarregaCursoForm form, ViewContainer container)
      throws Exception {

      Curso c = new CursoImpl();

      c.setId(form.getId());

      c = (Curso)ServiceLocator.instance().getCursoHandlerBI()
            .selectCurso(c).get(0);

      form.setNome(c.getNome());
      form.setId(c.getId());
      form.setCodigo(c.getCodigo());

}

public final void salvaCurso(
      br.mdarte.exemplo.academico.web.geral.manterCurso
      .SalvaCursoForm form, ViewContainer container)
      throws Exception {

      Curso c = new CursoImpl();

      c.setId(form.getId());

      c = (Curso)ServiceLocator.instance().getCursoHandlerBI()
            .selectCurso(c).get(0);

      c.setCodigo(form.getCodigo());
      c.setNome(form.getNome());

      ServiceLocator.instance().getCursoHandlerBI()
            .insertOrUpdateCurso(c);
```

```
    }
```

6. DetalhaCursoControleImpl.java :

```
    public final void carregaCurso(
          br.mdarte.exemplo.academico.web.geral.detalharCurso
          .CarregaCursoForm form, ViewContainer container)
          throws Exception {

          Curso c = new CursoImpl();

       c.setId(form.getId());

       c = (Curso)ServiceLocator.instance().getCursoHandlerBI()
          .selectCurso(c).get(0);

       form.setNome(c.getNome());
       form.setId(c.getId());
       form.setCodigo(c.getCodigo());
    }
```

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

```
maven compile deploy
```

Abriremos agora o eclipse, daremos Start no servidor jboss e verificaremos então no navegador o resultado do nosso sistema.

Bibliography

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