

Tutorial MDArte

Primeiros passos do
framework MDArte

Maio 2014

Contents

1	Preparação do Ambiente	5
1.1	JDK - Java 6	5
1.2	JBoss e Maven	5
1.2.1	Jboss	5
1.2.2	Maven	6
1.3	Variáveis de Ambiente	6
1.3.1	Instalando ferramentas e configurando o ambiente	7
1.4	MDArte	7
1.5	MagicDraw	7
1.6	Eclipse	8
2	MDArte	9
3	Desenvolvendo o primeiro projeto com o MDArte	11
3.1	Criação de um Novo Projeto	11
3.2	Controle de Acesso	12
3.2.1	Baixando o Controle de Acesso e configurando propriedades do projeto	12
3.2.2	Adicionando configurações do Controle Acesso ao JBoss	13
3.2.3	Compilando o Controle de Acesso	15
3.2.4	Preparando o banco do Controle de Acesso	15
3.3	Modelando o nosso primeiro projeto	15
3.3.1	Modelando a camada de domínio	15
3.3.2	Criando o Banco de Dados	24
3.3.3	Criando Value Objects	25
3.3.4	Modelando a camada de serviços	27
3.4	Implementando as classes de controle dos CRUD gerados	30
3.4.1	Inicializando o servidor e testando a aplicação	40
A	Configuração do JBoss e acesso ao banco de dados	45
A.1	Configuração das propriedades do projeto para acesso ao banco de dados	45
A.2	Configuração do JBoss	46
A.2.1	Configuração dos datasources utilizados pelo JBoss	46

A.2.2	Configuração do acesso das aplicações ao banco de dados	47
B	Configurando repositório externo do Maven	49
	Bibliography	51

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 Maven¹ 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 módulos 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 Variáveis 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 atribuído 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].

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 (MDA¹) definida pela Object Management Group (OMG²) 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` AndromDA³ 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/>

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 já 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 seguintes 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>controleacesso`.

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 CRUD² 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 direito 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.

²CRUD 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.

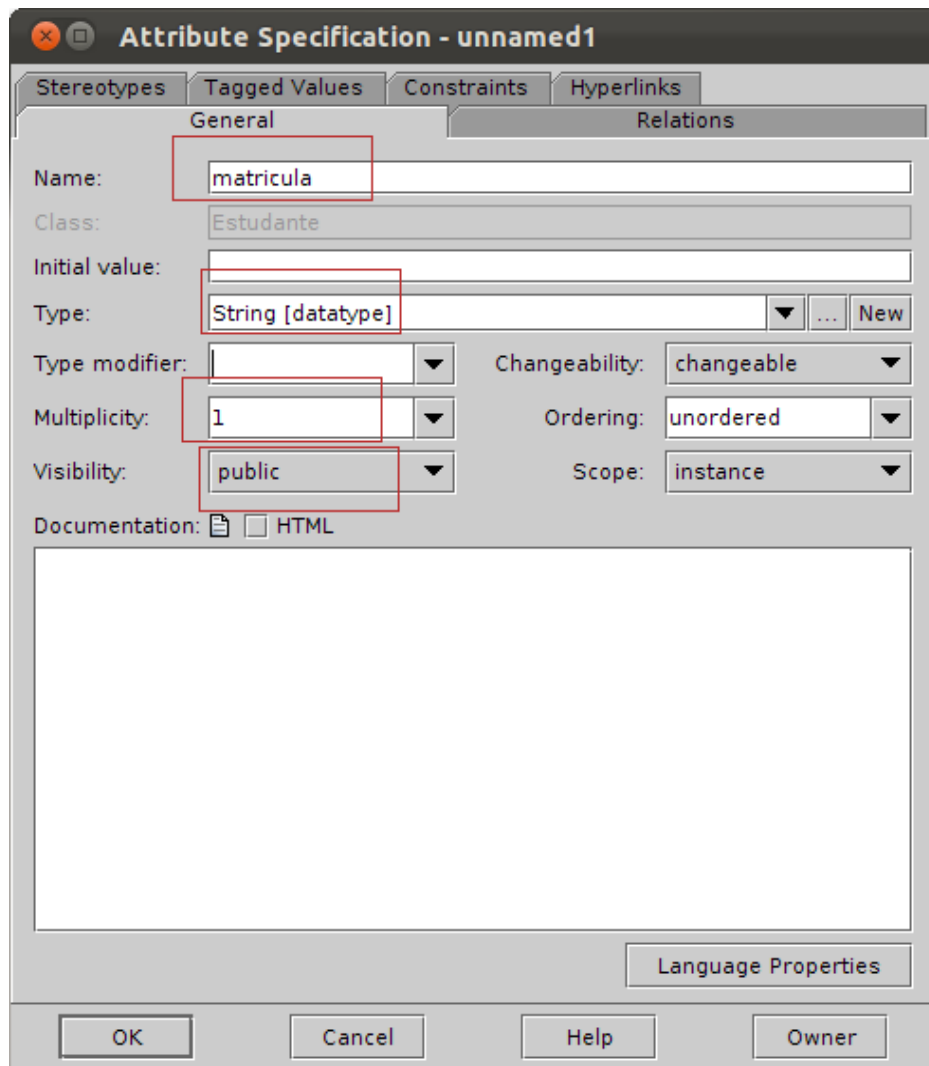


Figure 3.3: Configuração do parâmetro matrícula da classe Estudante.

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.

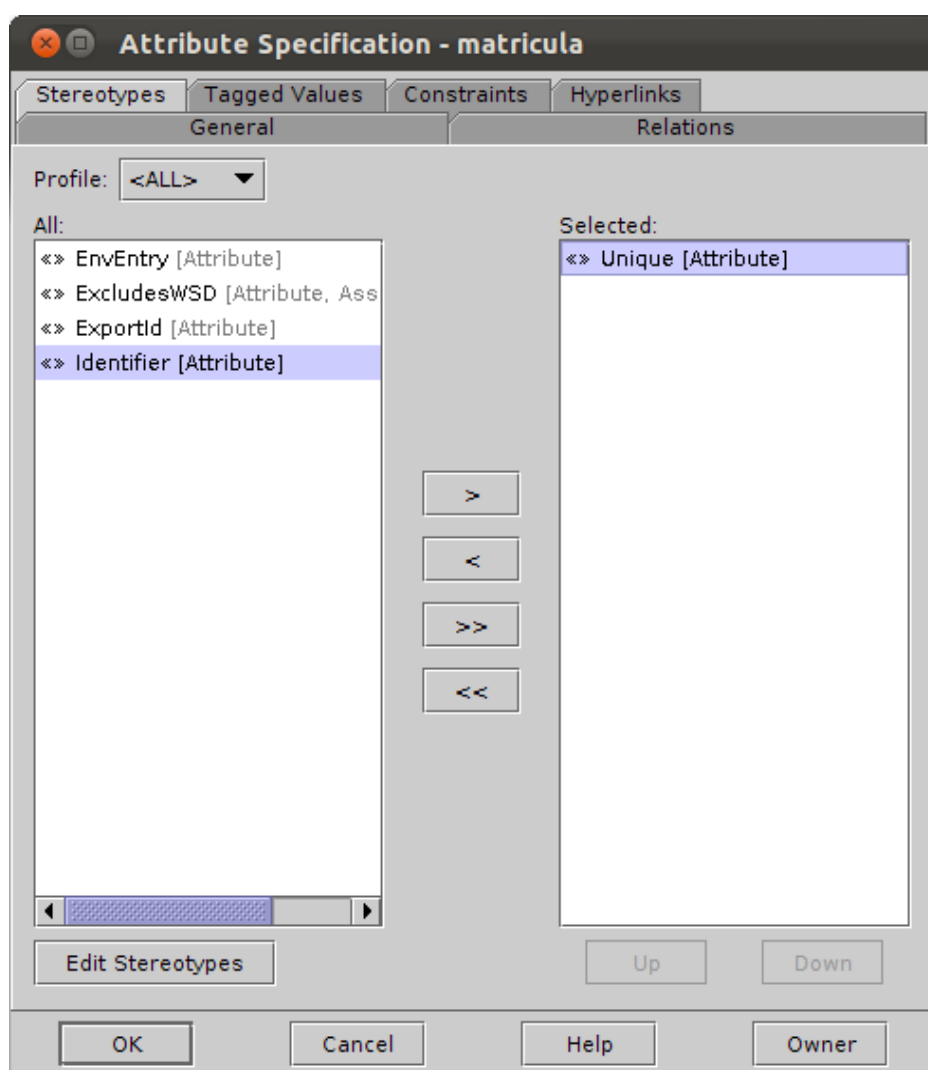


Figure 3.4: Adição de estereótipo no atributo matrícula.

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

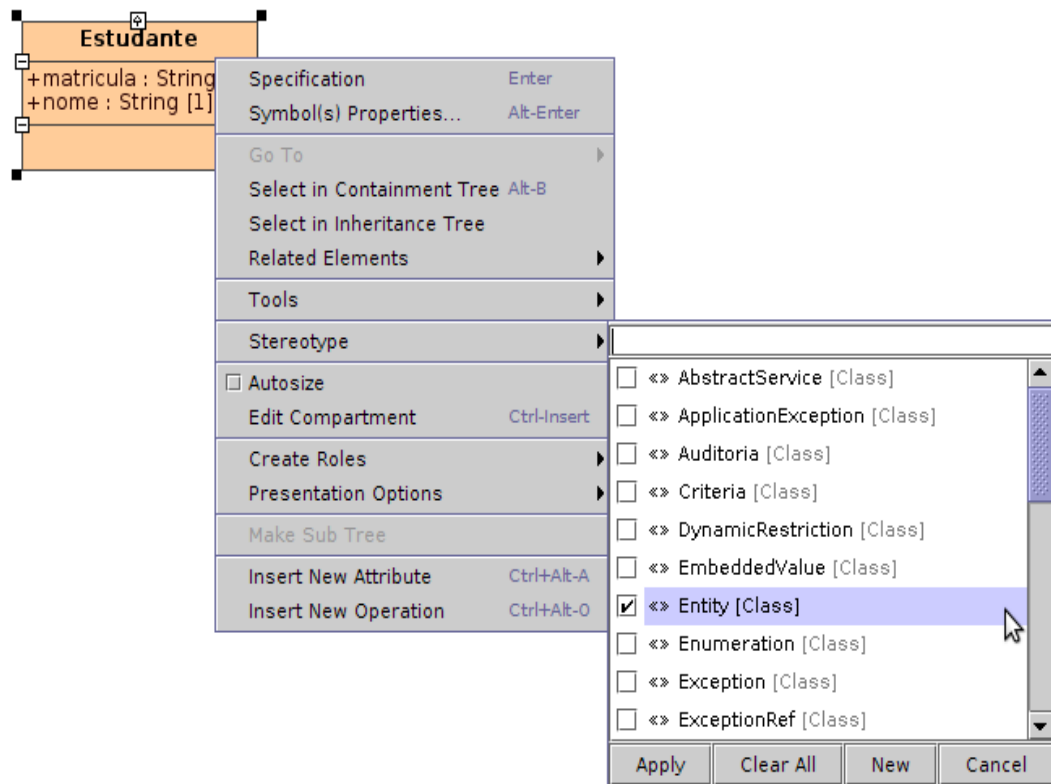
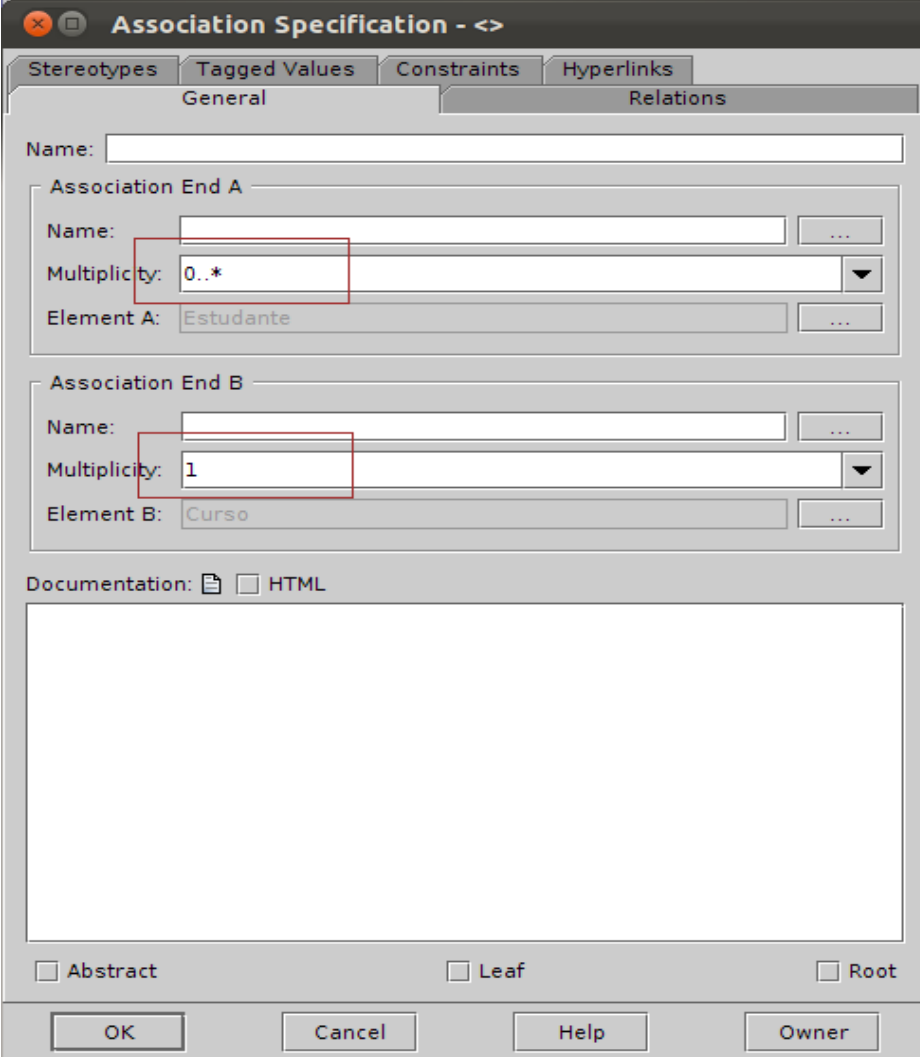


Figure 3.5: Adição dos estereótipos Entity e Manageable na classe estudante.

8. No mesmo diagrama de classes, crie outra classe. Clique com o botão direito 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.



The image shows a 'UML Association Specification' dialog box with the following fields and options:

- General Tab:**
 - Name:** (empty text field)
 - Association End A:**
 - Name:** (empty text field)
 - Multiplicity:** 0..* (highlighted with a red box)
 - Element A:** Estudante
 - Association End B:**
 - Name:** (empty text field)
 - Multiplicity:** 1 (highlighted with a red box)
 - Element B:** Curso
 - Documentation:** ☐ HTML
 - Options:** ☐ Abstract, ☐ Leaf, ☐ Root
- Buttons:** OK, Cancel, Help, Owner

Figure 3.6: Definindo multiplicidade da associação.

14. A associação deve ser dupla, tanto `Estudante` 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.

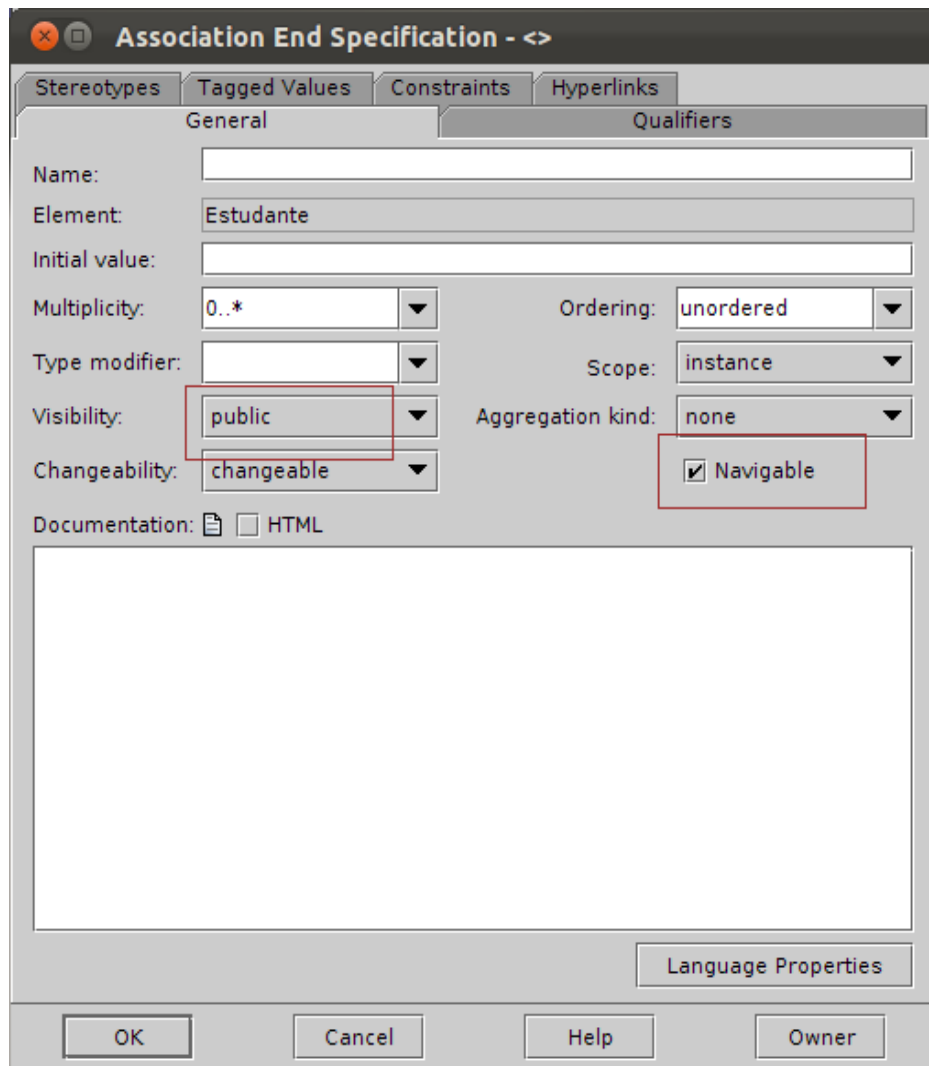


Figure 3.7: Configuração da navegabilidade da associação.

O resultado final pode ser visto na imagem [3.8](#):

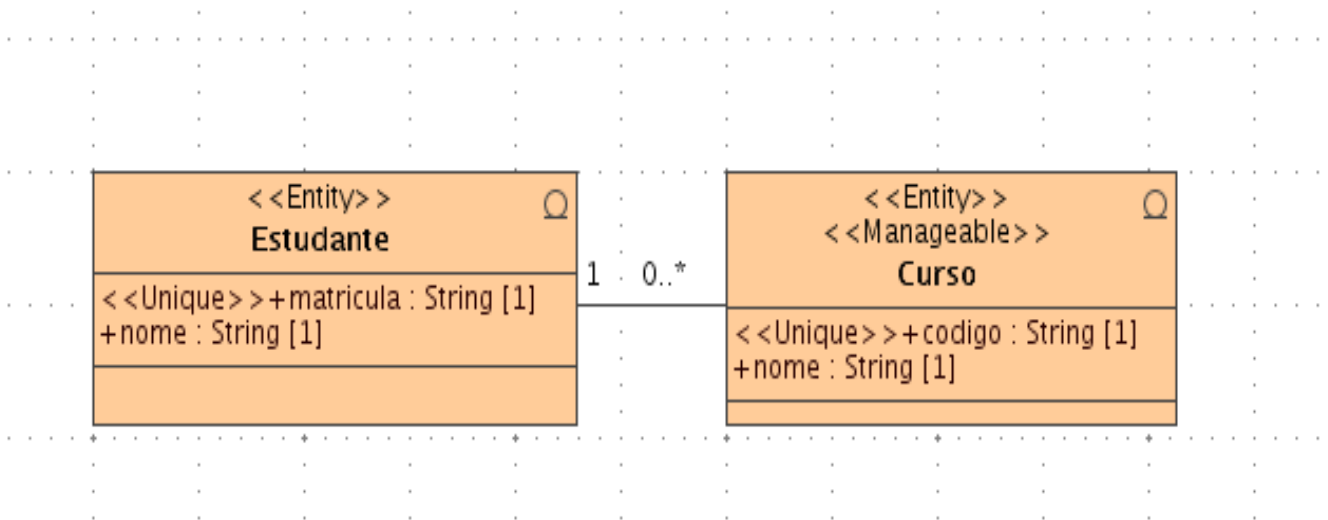


Figure 3.8: Resultado final da modelagem da camada de domínio.

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:

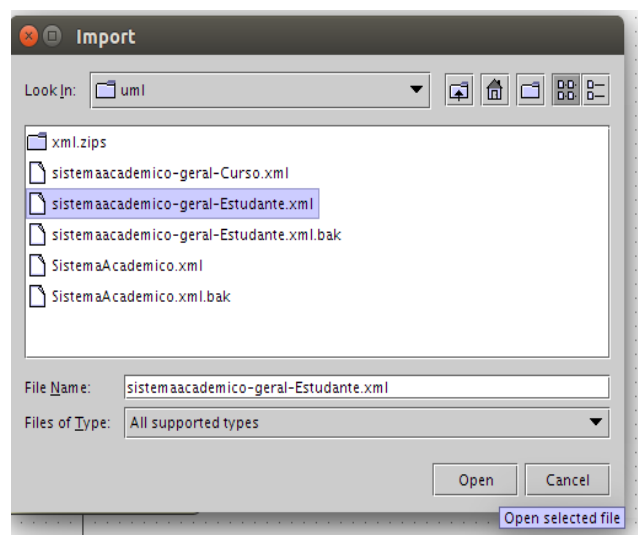


Figure 3.9: Criação do pacote para a camada de serviço.

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

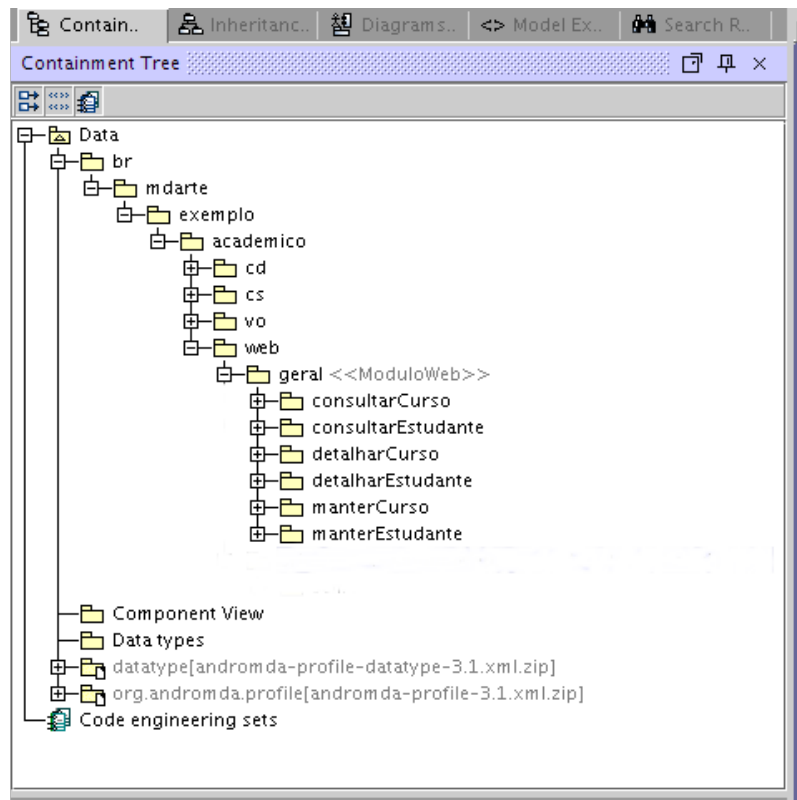


Figure 3.10: Arvore de diretórios do projeto após a importação dos CRUD gerados automaticamente.

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

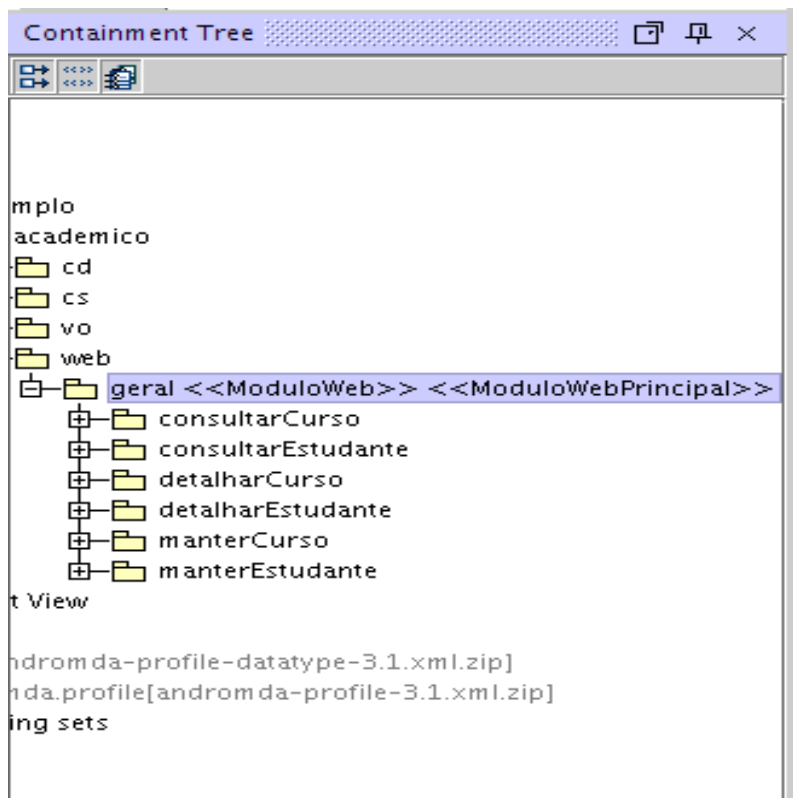


Figure 3.11: Arvore de diretórios do projeto após a importação dos CRUD gerados automaticamente.

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 criação 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 através 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 chamamos `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](#).

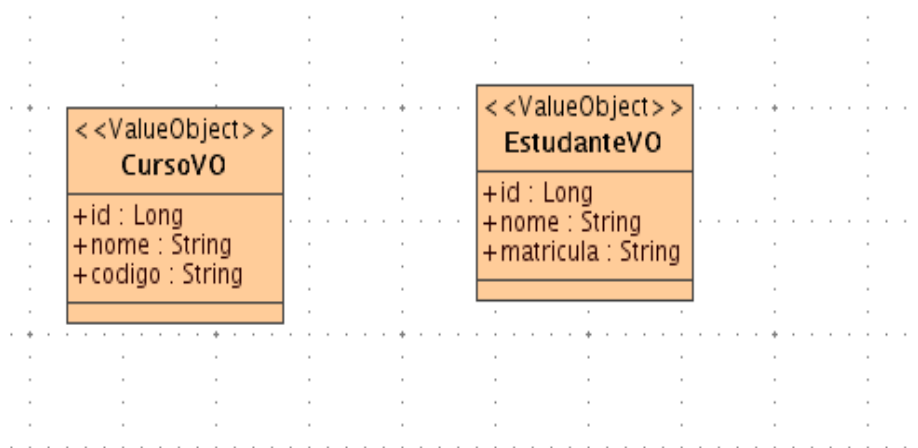


Figure 3.12: Criação dos ValueObject.

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 seu projeto deve estar como na imagem [3.13](#).

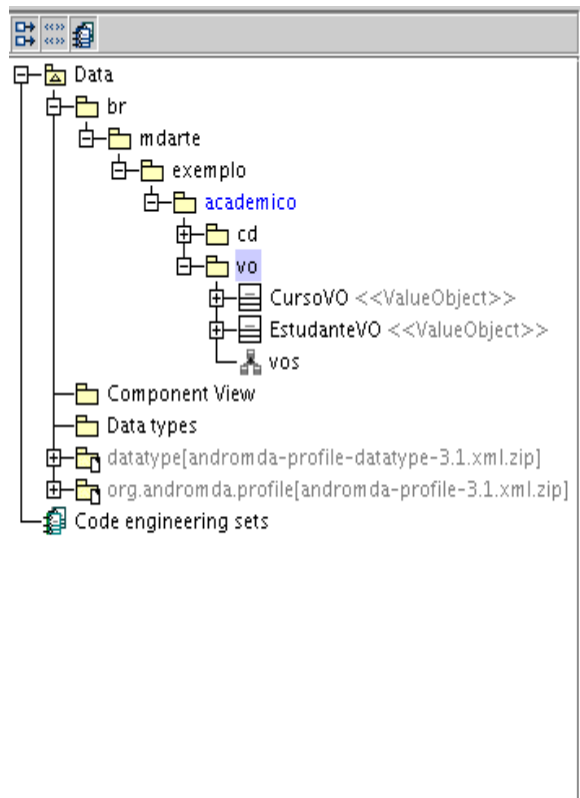


Figure 3.13: Estrutura de pacotes após a criação dos ValueObject.

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 mantém 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:

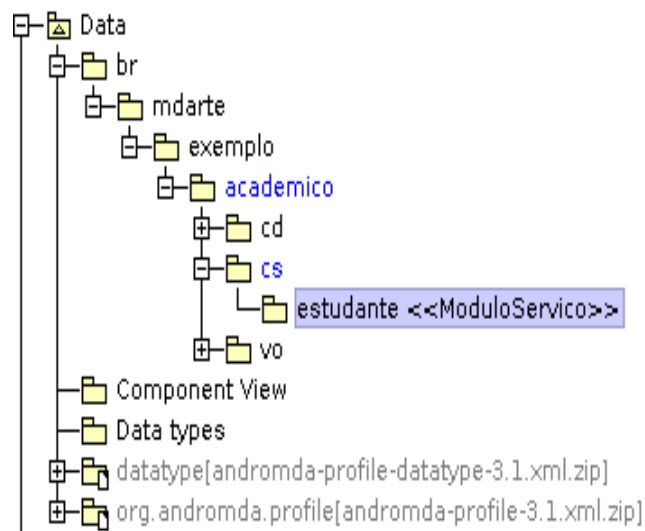


Figure 3.14: Criação do pacote para a camada de serviço.

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.

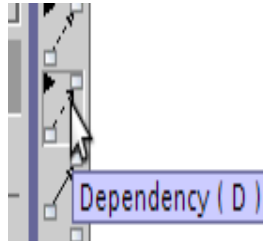


Figure 3.17: Criação da dependência entre as classes do serviço.

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

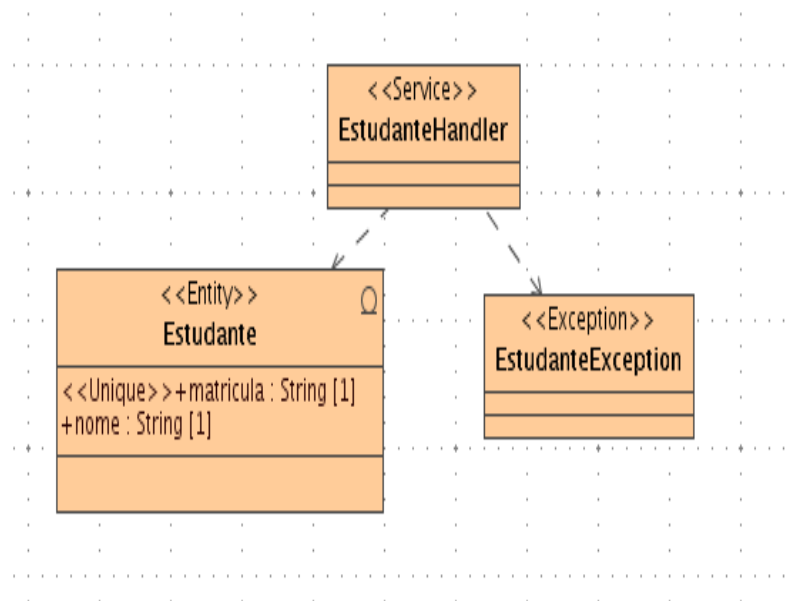


Figure 3.18: Diagrama de classe completo do módulo de serviço para a classe `Estudante`.

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 SUCCESSFUL".

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 código 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(Constants.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.cdCurso;
import br.mdarte.exemplo.academico.cdCursoImpl;
import br.mdarte.exemplo.academico.toCursoTO;
import br.mdarte.exemplo.academico.toCursoTOImpl;
import br.mdarte.exemplo.academico.action
    .DefaultFilterAction;
import java.util.Collection;
import java.util.ArrayList;
import br.mdarte.exemplo.academico.voCursoVO;
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(Constants.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.cdCursoImpl;
import br.mdarte.exemplo.academico.cdCurso;
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.cdCurso;
import br.mdarte.exemplo.academico.cdCursoImpl;
import br.mdarte.exemplo.academico.voCursoVO;

/**
 * @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 testando 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](#).

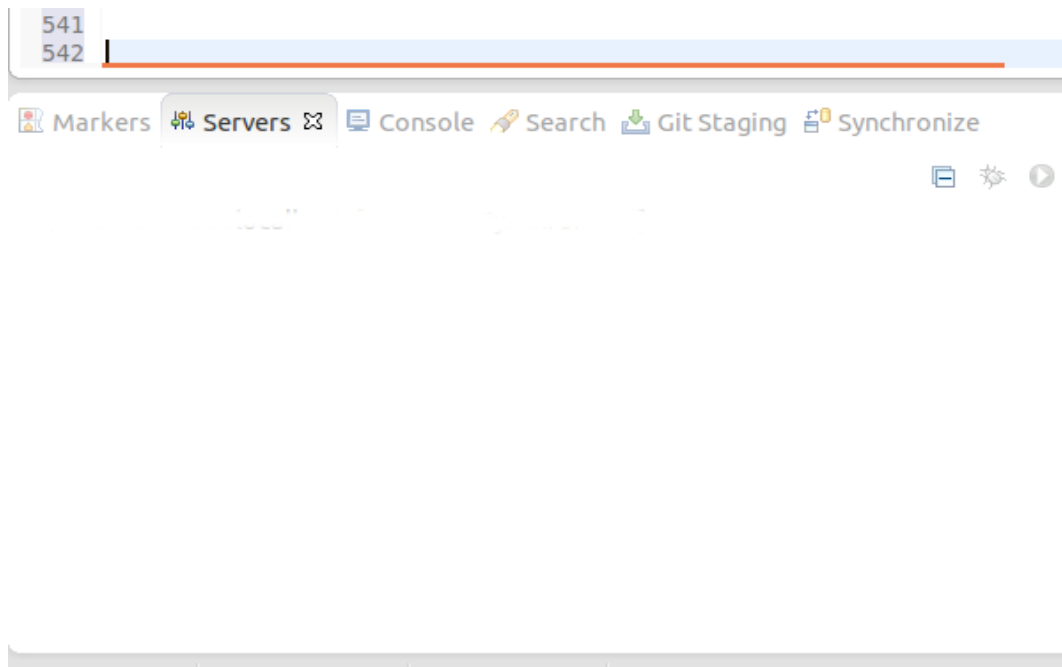


Figure 3.19: Inicializando o servidor JBoss.

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](#).

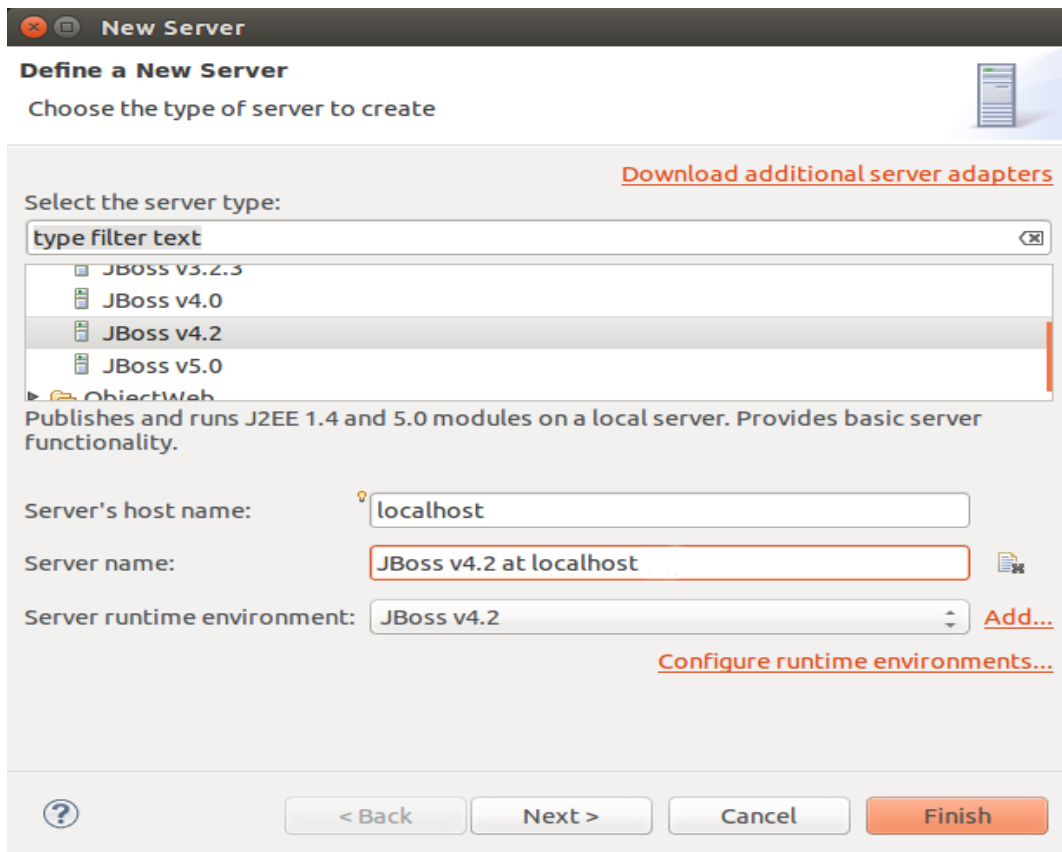


Figure 3.20: Inicializando o servidor JBoss.

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:

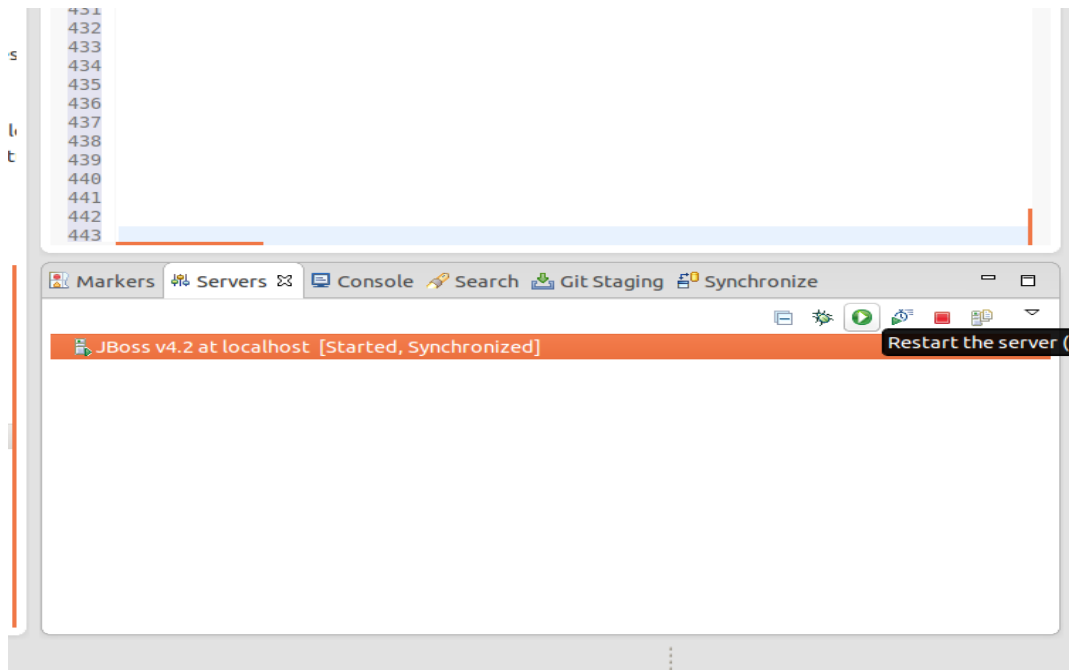


Figure 3.21: Inicializando o servidor JBoss.

Para acessar o sistema, abriremos o navegador e acessaremos a url `http://localhost:8080/sistemaacademico`. 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'.

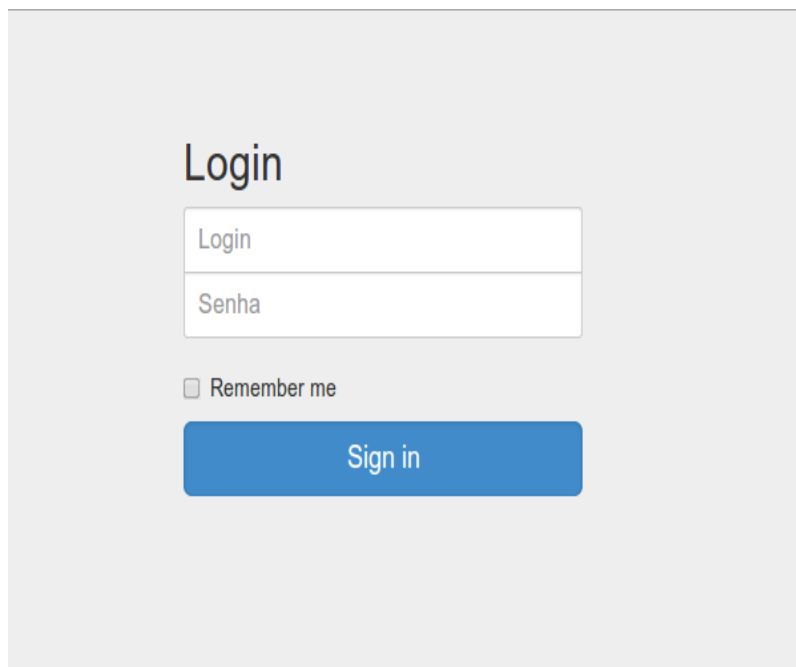


Figure 3.22: Página de login do Sistema Acadêmico.

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.

Preencha Campos

Matricula

Nome

Consulta Estudante

© MDArte

Figure 3.23: Página inicial de Consulta Estudante.

Se pesquisarmos sem nenhum filtro veremos uma lista completa dos estudantes registrados como na imagem [3.24](#).

Resultado Consulta

[Nova Consulta](#)

Estudantes

4 itens encontrados, exibindo todos itens.1

Matricula	Nome	
0001	Estudante 1	Detalha Estudante
0002	Estudante 2	Detalha Estudante
0003	Estudante 3	Detalha Estudante
0004	Estudante 4	Detalha Estudante

Opções de exportação: [CSV](#) | [Excel](#) | [XML](#) | [PDF](#)

© MDArte

Figure 3.24: Resultado da busca sem filtro pelos estudantes.

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 MDaTe durante a sua execução, este, na raiz do projeto, especificamente 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/conf`. Alteraremos o arquivo adicionando uma tag `<application-policy 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.

Configurando repositório externo do Maven

Bibliography

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