

# Relatório DBM

## MILESTONE 2 CRUDAPP

DBM – João Ventura

Tiago Monteiro 140221001

# Relatório DBM

## MILESTONE 2 CRUDAPP

### Introdução

Este relatório tem como objetivo explicar o projeto desenvolvido para a CRUDApps para a geração de código automático, neste documento irão ser abordados temas como os modelos e meta-modelos utilizados, o workflow da aplicação, melhorias relativamente á primeira entrega, a arquitetura da aplicação, e por fim os extras da aplicação que consistem em validações web, geração de uma aplicação react native, relações e layout.

### *Contexto do projeto*

...

O projeto consiste na aplicação dos conceitos aprendidos na cadeira de DBM para a geração de software orientada a modelos. O Projeto pretende desenvolver um software para a CRUDApps onde o processo de desenvolvimento de software era estático, optando-se então por se desenvolver um mecanismo de geração de código automática através de modelos.

## Modelos e Meta-modelos

Neste projeto foi necessário criar modelos e meta modelos de modo a fazer transformação de M2M e M2T. Os meta-modelos criados são “Class”, “Attribute” e “Relation” e “Model”.

### Class

A classe tem como objetivo representar uma Classe de uma linguagem com paradigma Objetos. Esta é definida por um conjunto de parâmetros essenciais para a geração do código.

### Parâmetros

Name – O nome da classe

Attributes – Os atributos da classe (meta-modelo Atributo)

Relations – As relações que esta classe tem com outras classes

Package – O Package onde a classe se encontra inserida

Extend – Usado quando a classe herda de outra classe

SuperClass – Usando quando é superclasse de outra classe

ProtectedMethods – Os métodos protegidos que são para manter após nova geração

```
private String name;  
private List<Attribute> attributes;  
private List<Relation> relations;  
private String pkg;  
private Class extend;  
private boolean superClass;  
private String protectedMethods;
```

Figura 1 - Parâmetros da Classe

## Attribute

Este meta-modelo serve para armazenar toda a informação sobre um atributo desde o seu nome e tipo às suas *constraints*.

## Parâmetros

Name – O nome do atributo

Type – O tipo do atributo

Required – Constrain required, se é um atributo que não pode ser nulo

Max – O valor máximo (Apenas para atributos numéricos)

Min – O valor mínimo (Apenas para atributos numéricos)

Contains – Se contém uma cadeia de caracteres (Apenas para String)

notEmpty – Se não é vazia (Apenas para String)

maxLength – O tamanho máximo da String

minLength – O tamanho mínimo da String

```
private String name;  
private String type;  
private boolean required;  
private Integer max;  
private Integer min;  
private String contains;  
private boolean notEmpty;  
private Integer maxLength;  
private Integer minLength;
```

Figura 2 - Parâmetros do meta-modelo Atributo

### Relation

Este meta-modelo tem como objetivo representar as relações que podem existir entre as classes. Estas relações podem ser de 4 tipos sendo dois idênticos:

- OneToOne
- OneToMany
- ManyToOne
- ManyToMany

### Parâmetros

Base – A classe base da relação

Target – A classe com quem a classe base tem uma relação

RelationType – Um dos 4 tipos de relação possíveis

```
private Class base;  
private Class target;  
private RelationType type;
```

Figura 3 - Parâmetros do meta-modelo Relation

### Model

Este é o meta-model responsável por agrupar todos os outros, é para este Model que são feitas as transformações M2M. Este para além de ter como parâmetros os meta-modelos Class e Relation possui também o nome do modelo.

## Workflow

Neste projeto foram encontradas quatro fases muito importantes, a transformação de M2M, a transformação de M2T a nível de geração de código de linguagens orientadas a objetos (ex: Java), a fase de geração da base de dados também esta M2T que permite existir um ORM fazendo a ponte entre o código gerado em Java e os dados e a geração da interface de administração que permite um utilizador comum manipular os dados. De modo a que o projeto seja extensível e escalável estas quatro fases do workflow foram divididas em interfaces. Assim sempre que for necessária uma nova maneira de fazer uma das quatro fases basta criar uma nova classe que use a interface. Então o workflow para a geração de código passa por transformação M2M para obter o nosso modelo, depois a partir do modelo obtido faz-se uma transformação M2T para gerar o código para base de dados que é executado no momento, é gerado código da linguagem de programação com o acesso á base de dados e por fim é gerado o código da interface de administração e também o código da geração da aplicação React Native.

## Build

O build consiste na junção das quatro fases mencionadas acima, uma parte importante neste build é a abstração criada devido ao uso de Interfaces, sendo estas:

M2M – Interface responsável pela transformação de um modelo no desejado

M2TClass – A Interface responsável por gerar código para paradigma Objetos

M2TBd – A interface responsável por gerar código para bases de dados

WEB – A interface responsável por gerar o painel de administração web.

Assim para gerar o código só tem de se indicar classes que implementem estas interfaces, no exemplo da figura 4 temos como M2M o XMLToModel que converte xml no modelo, temos para M2TBd MySql que vai gerar código SQL apropriado para bases de dados Sqlite3 e temos como M2TClass Java que irá gerar código em java tendo noção qual a base de dados que usara para alimentar os objetos e PureHTML para gerar a interface gráfica de CRUDs. Através desta solução é possível estender o programa para várias bases de dados ou várias linguagens sem comprometer nenhum dos outros processos.

```
Build build = new Build(new Java(), new XMLTOModel( filename: "src/model/person.xml"), new Sqlite3(), new PureHTML());
build.build(new File( pathname: "C:\\Users\\pctm\\Desktop\\my2"));
```

Figura 4 - Código para fazer build

## M2M

Esta fase corresponde á transformação de modelos importados para o modelo desejado que segue os meta-modelos criados neste projeto (página 2). Neste projeto é possível converter dois tipos de modelos para o modelo desejados, sendo estes XML e XML. O XML terá de cumprir o DTD da figura-5 de modo a ser possível a sua transformação, um exemplo e um XML que cumpre este DTD é o XML da figura-6.

```
<?xml encoding="UTF-8"?>

<!--ELEMENT model (class*,relation*)-->
<!--ATTLIST model
    name CDATA #REQUIRED-->

<!--ELEMENT class (attribute)+-->
<!--ATTLIST class
    name CDATA #REQUIRED
    extends CDATA #IMPLIED-->

<!--ELEMENT relation EMPTY-->
<!--ATTLIST relation
    base CDATA #REQUIRED
    target CDATA #REQUIRED
    type (ManyToMany|OneToMany|ManyToOne|OneToOne) #REQUIRED-->

<!--ELEMENT attribute EMPTY-->
<!--ATTLIST attribute
    name CDATA #REQUIRED
    type CDATA #REQUIRED
    required (true|false) #IMPLIED
    max CDATA #IMPLIED
    min CDATA #IMPLIED
    isAfter CDATA #IMPLIED
    isBefore CDATA #IMPLIED
    contains CDATA #IMPLIED
    notEmpty (true|false) #IMPLIED
    maxLength CDATA #IMPLIED
    minLength CDATA #IMPLIED-->
```

Figura 5 - DTD para o XML

```

<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE model SYSTEM "model.dtd">

<model name="Person">
  <class name="Super">
    <attribute name="ok" type="int" min="0" max="5"/>
    <attribute name="da" type="Date"/>
  </class>
  <class name="Person" extends="Super">
    <attribute name="name" type="String" required="true"/>
    <attribute name="age" type="int" required="true"/>
  </class>
  <class name="Manel" extends="Person">
    <attribute name="coi" type="String" contains="a" notEmpty="true" minLength="4" maxLength="9"/>
    <attribute name="man" type="int" />
  </class>
  <relation base="Person" target="Manel" type="ManyToMany"/>
</model>

```

Figura 6 - XML válido para o DTD criado

No projeto esta transformação é feita a partir de uma classe que implemente a interface M2M que possui o método getModel(). Para demonstração foram feitas duas conversões, a partir de xml e a partir de xmi, mas o programa não está limitado e pode-se expandir para outras conversões.

## M2T

Esta é a fase do workflow onde é gerado código. Esta parte está dividida em geração de código orientado a objetos, geração de bases de dados e geração das interfaces gráficas.

Para gerar código orientado a objetos deve-se usar uma classe que implemente a interface M2TClass, pois é utilizada pelo build para gerar o código. Este código passa pela geração de atributos, gets e sets até ao código ORM (save, all, where, delete) e ainda código da parte das relações, sendo este bidirecional, ou seja, se a classe C1 tiver relação com a classe C2, através da classe C1 posso obter as instancias C2 com quem se relaciona C1 e a partir da classe C2 posso obter as instancias de C1 com quem C2 se relaciona. Um aspeto importante aqui é o tipo de base de dados que irá ser usado para alimentar os objetos com dados, por este motivo o método Convert() da interface M2TClass recebe como parâmetros o model e também uma classe da interface M2TBd, esta informação é inserida no template através de parâmetros partilhados como se pode observar na figura 7, sendo possível gerar o código usando das classes usando as mesmas propriedades partilhadas que irão ser usadas para incluir ficheiros específicos no template.

```

Model2Text model2Text = new Model2Text( pathname: "src/templates");
System.out.println(m2TBd.getBdTyp().toString());
model2Text.addShared("bdname", m2TBd.getBdTyp().toString());
model2Text.addShared("bdinclude", m2TBd.getIncludeClassName());
model2Text.addShared("bdincludefunctions", m2TBd.getImportFunctionsFileName());

```

Figura 7 - Parametros partilhados no template



Para gerar código que crie as bases de dados deve-se utilizar uma classe que implemente a interface M2TBd, pois é esta que contém informação importante como por exemplo o template a utilizar.

```
public class Mysql implements M2TBd {
    @Override
    public void convert(Model model) {
        Model2Text model2Text = new Model2Text( pathname: "src/templates");
        MySqlConnection sq = new MySqlConnection();
        String sqlTables = model2Text.render(model, name: "mysql/mysql_create.ftl");
        System.out.println(sqlTables);
        sq.executeAll(sqlTables);
    }

    @Override
    public BD_Type getBdTyp() { return BD_Type.MySql; }

    @Override
    public String getIncludeClassFileName() { return "java_sqlite3.ftl"; }

    @Override
    public String getImportFunctionsFileName() { return "java_mysql_functions.ftl"; }
}
```

Figura 8 - Classe MySql que é responsável por gerar código SQL para bases de dados MySql

## WEB(M2T)

Nesta fase o objetivo é gerar a interface gráfica bem como os ficheiros de suporte á mesma, para que seja possível fazer os CRUDs bem como a web API.

```
package Web;

import metamodels.Model;

/**
 * Created by pctm on 01/06/2017.
 */
public interface Web {

    public final String PATH_DEFAULT = "src/proj/";

    public void build(Model model);

    public void build(Model model, String path);
}
```

## Arquitetura

Este projeto é dividido em 8 packages sendo estes M2M, M2T, M2TBd, Web, metamodels, model, templates e utils.

### M2M

Package responsável por guardar classes de transformações M2M

#### Classes

XMLToModel – Converte xml para Model

M2M (Interface) – Interface para conversores M2M como o XMLToModel

### M2T

Package responsável por guardar classes de transformações M2T para linguagens de programação

#### Classes

M2TClass (Interface) – Interface para conversores M2T para linguagens de programação como java

Java – Converte o Model para código Java

### WEB

Package responsável por guardar classes de transformações M2T para web interfaces

#### Classes

WEB (interface) – Interface para conversores M2T para web

PUREHtml – Classe responsável por gerar código web

### M2TBd

Package responsável por guardar classes de transformações M2T para bases de dados

#### Classes

M2TBd (Interface) – Interface para conversores M2T para bases de dados

MySQL – Classe responsável por gerar código para bases de dados MySQL

Sqlite3 – Classe responsável por gerar código para bases de dados Sqlite3

### *Metamodels*

Package responsável pelos meta-modelos

### *Model*

Package responsável pelos modelos (xml, xmi)

### *Templates*

Package responsável por guardar os templates

#### *Pastas*

Java – templates de java

Sqlite3 – templates de sqlite3

MySQL – templates de MySQL

WEB – templates web

RN – templates React Native

### *Utils*

Package responsável por guardar utilitários tais como classe de acesso a bases de dados

O ORM já era uma aplicação com muitas funcionalidades, na defesa da 1ª milestone foi aconselhado como melhoria para esta milestone a prevenção de sql injection. Assim a única melhoria face á 1ª milestone é a prevenção de sql injection.

```
11      'DROP TABLE PERSON;aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
```

## Extras

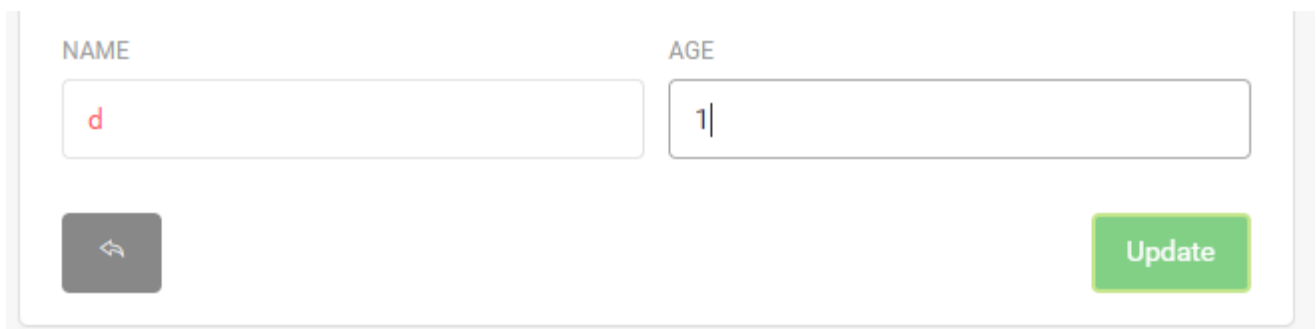
### Validações web

A validações web são importantes pois poupam os recursos do servidor , assim este foi um dos principais extras a implementar.

Através do modelo são geradas validações web para as seguintes costraints:

- Require
- Max
- Min
- maxLength
- minLength

Estão são inseridas na interface gráfica web através de javascript.



The screenshot shows a web form with two input fields: 'NAME' and 'AGE'. The 'NAME' field contains the letter 'd' and has a red error message 'd' below it. The 'AGE' field contains the number '1'. There is a grey button with a left arrow icon and a green 'Update' button.

Figura 9 - Exemplo de validação no Name

## React Native App

Visto que uma das gerações feitas é uma RESTful API, de modo a validá-la e a demonstrar o seu uso foi desenvolvida a geração de uma aplicação mobile, em React native, que permite através da API fazer crud.

Esta app é genérica e é apenas gerado um json de suporte, nesta app é possível ver os seguintes ecrãs:

- Menu
- List
- Create
- Edit

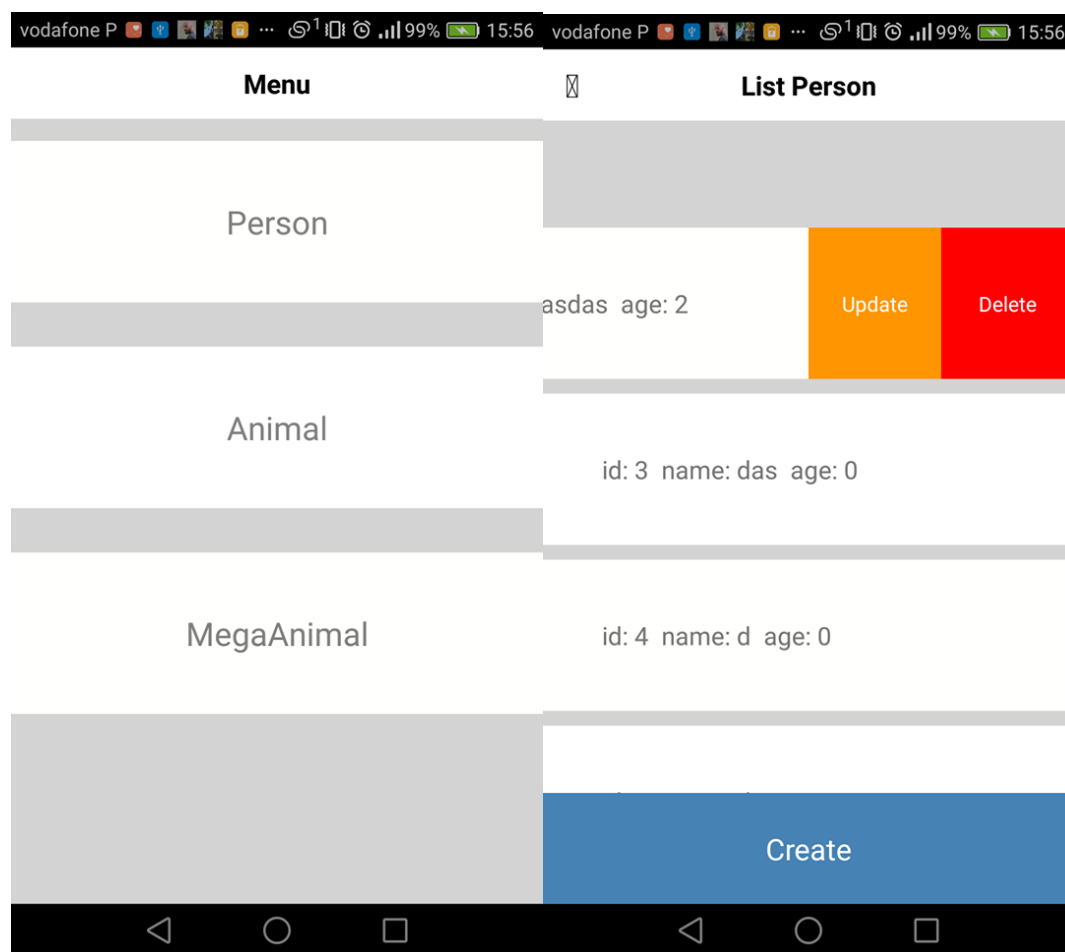


Figura 10 – Menu

Figura 11 - List

## Relações

Este extra consiste em o administrador conseguir manipular as relações, estas relações podem ser de 1 para 1, M para N e N para 1. Usou-se tabelas para quando são muitas relações para que seja mais fácil a edição das relações.

The screenshot shows the DBM interface. On the left is the 'Edit' form with fields for 'NAME' (containing 'ddsadasdas') and 'AGE' (containing '2'). There is a 'Update' button. On the right is the 'Relations' section, which contains two tables: 'Animal' and 'MegaAnimal'.

ID	NAME	TYPE	ACTIONS
1	Rato	Mamífero	<button>add</button>
2	Cão	Mamífero	<button>add</button>
3	Manel	Réptil	<button>add</button>

ID	NAME	SUPERTYPE	ACTIONS
1	CAOZAO	MEGA-CAO	<button>add</button>

Figura 11 - Painel de quando são N relações (lado direito)

The screenshot shows the DBM interface. On the left is the 'Edit' form with fields for 'NAME' (containing 'Rato') and 'TYPE' (containing 'Mamifero'). There is a 'PERSON' dropdown menu with a list of options: 'none', 'Person{ name=ddsadasdas age=2}', 'Person{ name=das age=0}', 'Person{ name=d age=0}', 'Person{ name=d age=0}', 'Person{ name=ee age=0}', and 'Person{ name=sd age=0}'. There is a 'Update' button.

Figura 12 - Painel de quando é só uma relação

### Layout

Foi também implementado um layout mais atrativo e responsivo utilizando bibliotecas como bootstrap e jquery. Este extra é importante pois cada vez mais a internet é utilizada por dispositivos moveis com diferentes resoluções de ecrã.

Animal List				
<a href="#">Create a animal</a>				
ID	NAME	TYPE	PERSONID	ACTIONS
1	Rato	Mamifero	2	<button>Update</button> <button>Remove</button>
2	Cão	Mamifero		<button>Update</button> <button>Remove</button>
3	Manel	Réptil		<button>Update</button> <button>Remove</button>

Figura 13 - Tabela de List

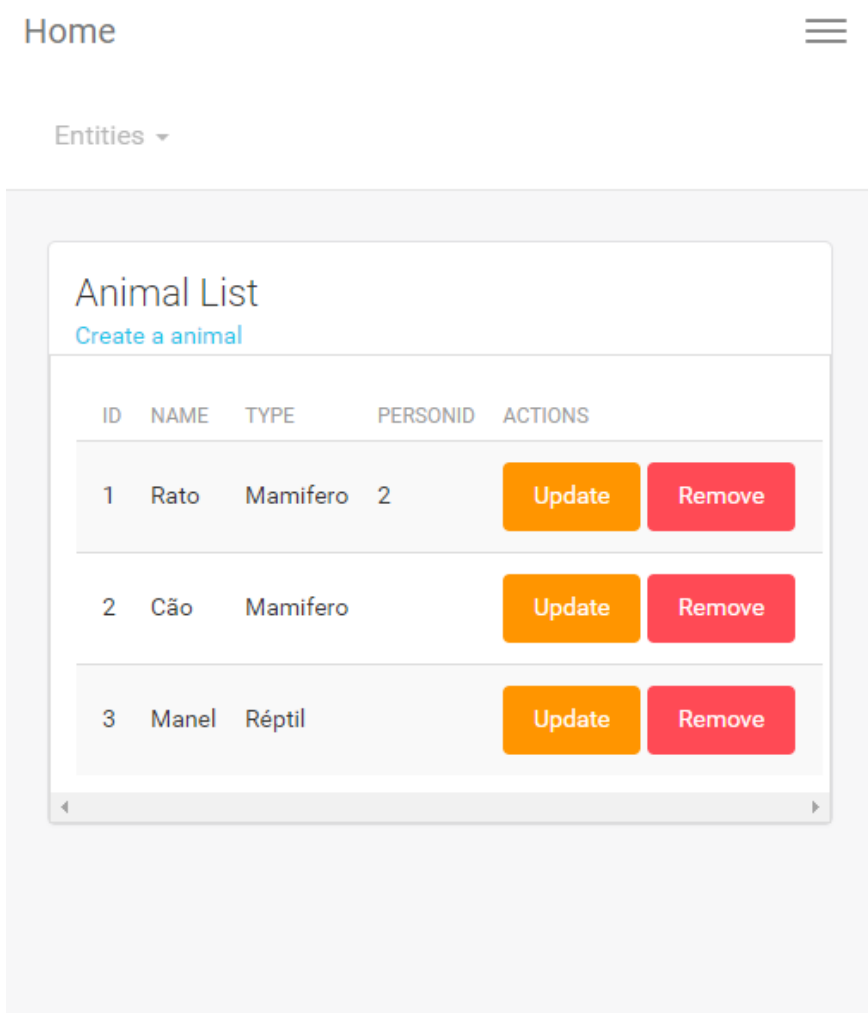


Figura 14 - Tabela em Mobile



## Relatório DBM



### Edit

NAME

Rato

TYPE

Mamifero

PERSON

Person{ name=ddsadasdas age=2}

Update

### Relations

MegaAnimal

ID	NAME	SUPERTYPE	ACTIONS
1	CAOZAO	MEGA-CAO	<div>add</div>

Figura 15 - Painel de Edit

### Edit

NAME

Rato

TYPE

Mamifero

PERSON

Person{ name=ddsadasdas age=2}

Update

### Relations

MegaAnimal

Figura 16 - Painel de edit em mobile

## Conclusão

Com o desenvolvimento deste projeto e com os conceitos adquiridos na cadeira de DB conclui-se que a geração de código cada vez tem mais impacto na criação de softwares, os próprios IDE's geram código e diria que cada vez mais a geração de código estará mais presente devido ao aumento de abstração e á necessidade de software rápido.