International Journal of Engineering and Applied Computer Science (IJEACS)
Volume: 02, Issue: 07, ISBN: 978-0-9957075-8-0, July 2017

Model-Driven Generation of MVC2 Web Applications: From Models to Code

M'hamed RAHMOUNI
Department of Computer Science, Faculty of Science
Ibn Tofail University
Kenitra, BP 133, Morocco

Samir MBARKI
Department of Computer Science, Faculty of Science
Ibn Tofail University
Kenitra, BP 133, Morocco

***Abstract*—Computer systems engineering is based, increasingly, on models. These models permit to describe the systems under development and their environment at different abstraction levels. These abstractions allow us to conceive applications independently of target platforms. For a long time, models have only constituted a help for human users, allow to manually develop the final code of computer applications. The Model-Driven Engineering approach (MDE) consists of programming at the level of models, represented as an instance of a meta-model, and using them for generating the end code of applications. The MDA (Model-Driven Architecture) is a typical model-driven engineering approach to application design. MDA is based on the UML standard to define models and on the meta-modeling environment (MOF) [1] for model-level programming and code generation. The code generation operation is the subject of this paper. Thus, in this work, we explain the code generation of MVC2 Web application by using the M2M transformation (ATL transformation language) then the M2T transformation. To implement this latter we use the Acceleo generator which is a generator language. In the M2T transformation, we use the PSM model of Struts2 already generated by M2M transformation as an input model of Acceleo generator. This transformation is validated by a case study. The main goal of this paper is to achieve the end-to-end code generation.**

Keywords- MDA; PSM; Code Generation; Acceleo; Struts; ATL transformation; MVC2 architecture

I. INTRODUCTION

In recent years, development of Internet, distributed, embedded or self-managed applications has considerably complicated the development of computer systems. These are used in increasingly varied and constrained environments (mobile terminals, cars, robots) and in increasingly critical contexts (aerospace, medical, military, nuclear). In addition, during their lifetime, software is subject to constant evolution to satisfy user needs and adapt to new platforms. Academic and industrial world are thus led to rethink the software production processes in order to adapt them to these new stakes.

A pragmatic vision to adapt to these changes is to allow the software production in automatic way in order to take into account the inevitable evolutions which these are subjected to. The models have always been used in computer science as a basis for reflection. However, they have often been relegated to the rank of simple documentation. Model-driven engineering aims to place models at the heart of the development process to make them the main elements through which applications are generated. Taking model change into account and automatically generating systems by generation from these models seems to be a promising outcome to meet changing constraints.

At the heart of model-engineering are placed the model transformations. They realize the automation of essential stages of software production: refinement and models composition, reverse engineering, code generation and documentation, etc. These transformations are intended to be used repeatedly by development teams, for example in software workshops. Therefore, it is crucial that they be validated and tested so that the developer can have full confidence in them.

The presented work is the continuation of the work presented in [39]. This work allows generate automatically a MVC2 web application that is a source code of an application based on Struts2 framework [1]. To realize this operation of code generation, we used ACCELEO [37] as a code generation language. To achieve this code generation, we begin by implementing the different templates corresponding to each meta-class which is an element of the PSM model already generated in [39]. After establishing the different templates (Action classes and Jsp pages), we validate the automatic code generation with a case study.

The remaining part of this paper is structured as follows: section 2 explains the process and methodology of this work. Section 3 is devoted to the architecture of UML and Struts2 meta-models. Section 4 is devoted to the transformation rules implementation. Section 5 is dedicated to the transformation rules execution and the result of execution process. Section 6 presents the Model-to-Text transformation implementation. The Model-to-Text transformation result is dedicated to the section 7. Section 8 presents the evaluation of this work. Section 9 discusses the main related work, while section 10 wraps up the conclusions and future works.

II. PROCESS AND METHODOLOGY

In this work, we begin our process by the presentation of the different meta-models (CIM, PIM and PSM). CIM model is represented by an UML class diagram of a case study of an Employee management. This UML class diagram is represented by an Ecore model. The PIM meta-model is an extract of UML class diagram meta-model. The PSM meta-model refers to the Struts2 meta-model. After models and meta-models, we define the different ATL transformation rules. Then, we implement the KM3 models corresponding to each PIM and PSM meta-model then the different Ecore models corresponding to these KM3. The next step is devoted to establish the traceability links between different elements of source and target meta-models and thereafter we define the different transformation rules in ATL transformation language. The ATL transformation result is a MVC2 web model represented in EMF model. In the end step, we use the generated PSM model as an input model of Acceleo generator then we generate the application code of the presented case study by applied a M2T transformation based on the cited ACCELEO generator. In the case of M2T transformation, we begin by the implementation of different templates corresponding to the Action classes and the JSP pages. The work of this transformation is validated and exemplified by a case study.

The tools support of this work is the UML [14], ATL transformation language [22]-[23]-[24], MOF [15], XMI [16], KM3 [6], OCL [17], EMF Project [15] and Acceleo [37].

The following section is dedicated to the M2M transformation based on MDA approach and ATL transformation language [25]. In this section, we begin by presenting the different meta-models.

III. UML AND STRUTS2 META-MODELS

In this section, we explain the different meta-classes that form the UML source and the target meta-models.

A. *UML Source Meta-model*

Figure 1 presents the UML source meta-model. This meta-model is a simplified UML model composed essentially of packages containing the data types and classes. The main elements of this metamodel are:

- UmlPackage: This is an UML package concept.
- Classify: Represents the generalization concept of meta-class. This latter represents both the concept of UML class as well as the data type concept.
- Class: Shows the UML class concept.
- DataType: Explains the concept of UML data types.
- Operation: Represents the concept of UML class methods.
- Parameter: Expresses the method parameters concept.
- Property: Represents the concept of UML class properties.

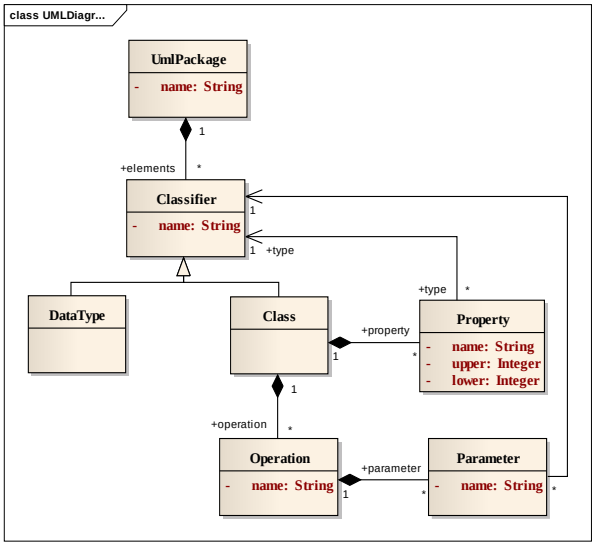

Figure 1. An Extract of UML Class Diagram Metamodel.

B. *Struts 2 Target Meta-model*

In this section, we present the Struts2 target meta-model. This meta-model is explained in first time in [39]. The different elements that form the Struts 2 meta-model are as follows:

- ModelPackage: Presents the concept of UML package. This package designates the notion of Model in the MVC2 architecture.
- ControllerPackage: Expresses the Controller concept in the MVC2 architecture.
- ViewPackage: Indicates the concept of Views package.
- ActionMapper: Represents the ActionMapper class concept.
- ActionProxy: Expresses the ActionProxy class concept.
- ActionInvocation: Expresses the ActionInvocation class concept.
- Action: Indicates the concept of action in the controller package.
- JspPages: Represents the Jsp package concept.
- Result: Expresses the generated element through an Action class.
- The Interceptors: Represents the concept of an Interceptor package.
- Interceptor: Indicates the interceptor classconcept.
- HttpRequest: Expresses the concept of HttpServletRequest class.
- HttpResponse: Designates the concept of HttpServletResponse class.

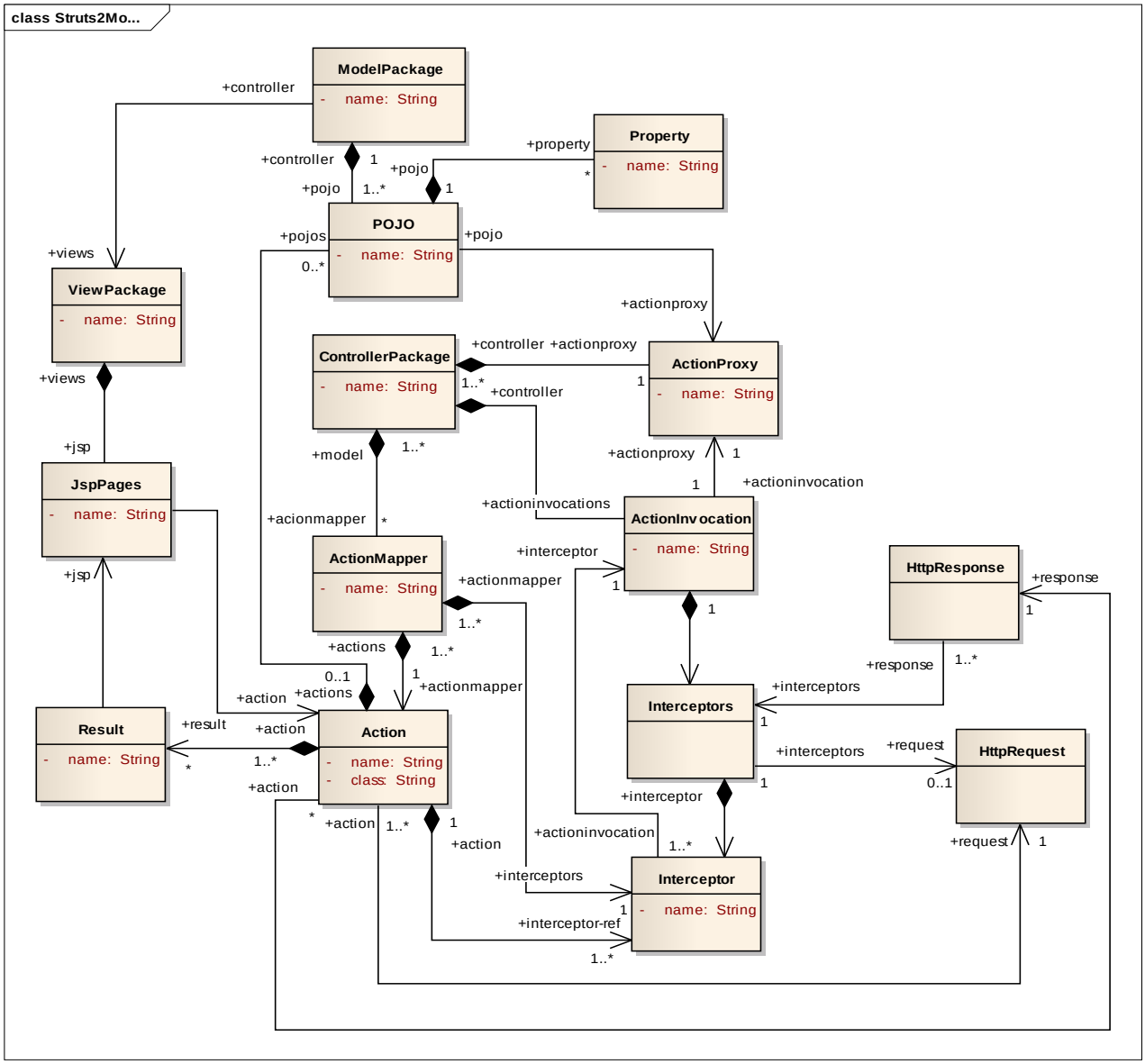

Figure 2. PSM Struts2 Meta-model.

IV. TRANSFORMATION RULES IMPLEMENTATION

This section is dedicated to the transformation rules implementation. It explains the different steps from implementation to execution of different transformation rules. In first step, we implement the different meta-models like: *struts2.km3*, *struts2.ecore*, *UML.km3* and *UML.ecore*. In second step, we establish the rules specification. Then, we define the different transformation rules based on these specification rules by ATL language in a file named *UML2Struts2.atl*. Finally, we implement the source model. This latter is an UML class diagram of Employee management represented in XMI language.

The different tools permits to achieve this work are: ATL plug-in integrated in Eclipse, XMI, OCL, KM3, UML, MOF and EMF project.

The next section is dedicated to the implementation of rules specification and then the definition and the execution of ATL transformation rules. Ecore and *km3* meta-models are not presented in this paper for letting it quite understandable and clear.

A. Rules Specification

This section is devoted to the presentation of the main rules to transform an UML Class Diagram into Struts2 Web model. The different specification rules are:

- An UML package generates a Struts2 package.
- The Struts package is composed of a Controller package and a View package.
- Each Controller Package is composed of a set of Action classes.
- An Action class is composed of a set of Result classes.
- An Operation generates an Action and a JSP pages.

B. *Rules-Based transformation written in ATL*

This section expresses the different rules which transform the UML class diagram into MVC2 web model. The different rules are:

Rule 1: From Operation to Struts 2 Action

This rule represents the main rule of this transformation. Thus, this rule allows generate the different action classes and Jsp pages of each Action. The name of each jsp page is composed from the name of the operation concatenated with the name of the class and followed by the extension ".jsp". The different rules that compose this main rule are as follow:

- Rule 1: An Operation can generate an Action.
- Rule 2: An Operation can generate a Result.
- Rule 3: Each Result class is composed of a set of Jsp pages.

The main rule is shown in figure 3. These rules are implemented by ATL language.

```
rule Operation2Action{

    from
        c : UML!Operation
    to
        js : NTiers!JspPage (
                name<- if c.name<>'Delete'then
                            c.name+c.class.name+'.jsp'
                        else 'Retrieve'+c.class.name+'.jsp'
                       endif
             ),
        frm : NTiers!Action(
              name<- c.name+c.class.name+'Action',
              method <- c.name+c.class.name,
              class <- 'com.web.struts2.'+c.name+c.class.name+'Action',
              result <- Sequence{fr}
             ),
         fr : NTiers!Result(
              name <- 'Success',
              type <- 'redirect',
              jsp <- js
         )
    }
```

Figure 3. Main Rule : From Operation to Action.

V. TRANSFORMATION RULES EXECUTION

In this section, we present a case study to demonstrate and exemplify our proposition. The UML class diagram of this case study represents the source model of our ATL transformation. The execution algorithm of ATL transformation allows browsing all transformation rules and thereafter generates the MVC2 web model. This latter is represented in figure 6.

A. *Case study*

The current case study is a system of a three classes. This system makes it possible to manage an employee of a given department. The different classes that compose this system are: City class, Department class and Employee class. In this case study, we elaborate only the CRUD operations (Create, Retrieve, Update, and Delete). These operations are most often implemented in all systems. Figure 4 presents the Ecore model equivalent to the UML class diagram of a department's employees.

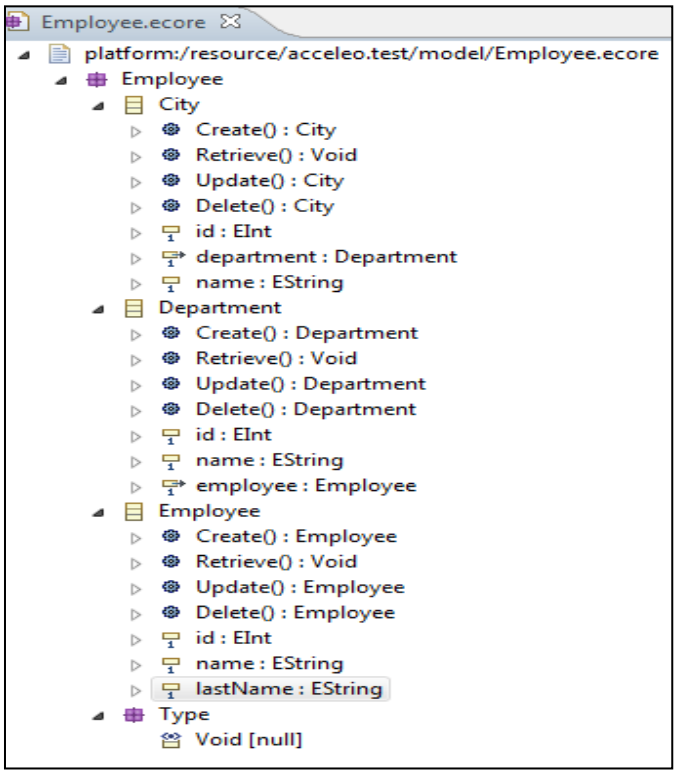

Figure 4. Ecore model of a department's employees.

B. *ATL Transformation Result*

Figure 5 shows the generated MVC2 Web model of Struts2. This model contains the different ingredients for implementing a presentation layer respecting the architecture of MVC2 pattern.

The generated model is composed of a Struts package that is composed of a set of Action classes and the View package that is composed of a set of Jsp pages.

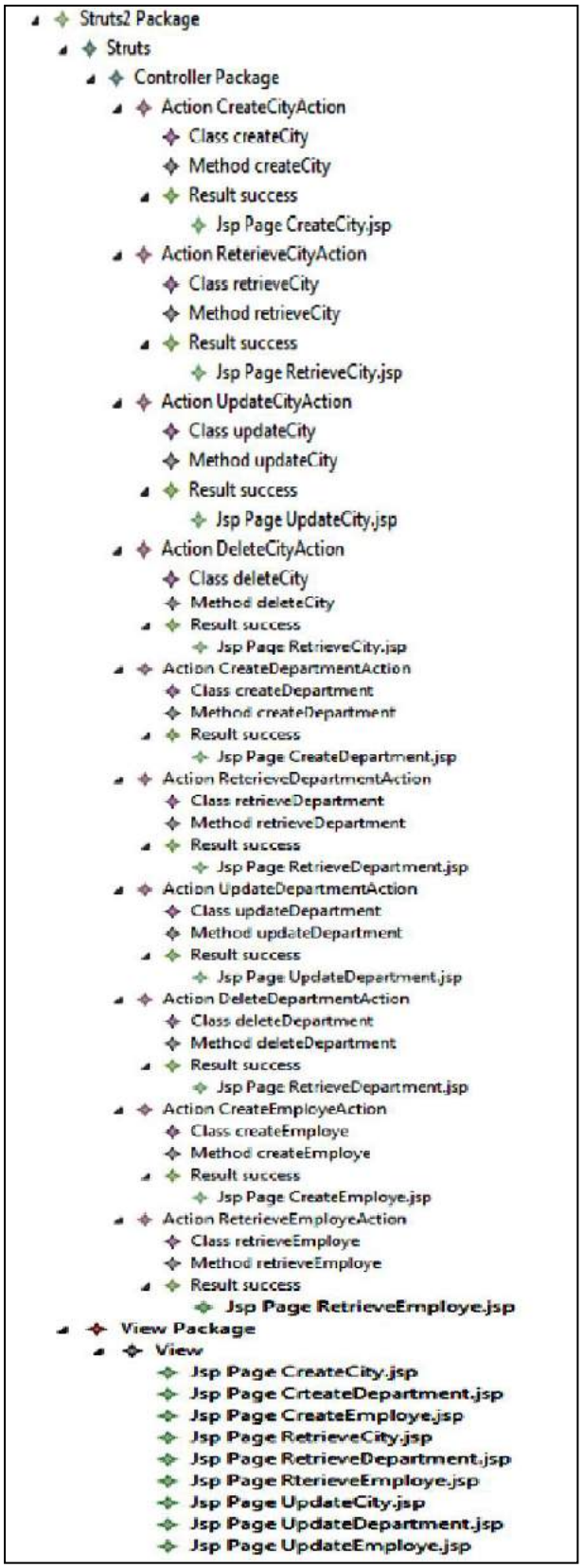

Figure 5. The Generated MVC2 Web model.

VI. Model-To-Text (M2T) Transformation Implementation

In this M2T transformation, we use the Acceleo [37] code generator which is a code generator integrated into Eclipse. The following section gives an idea of this generator then a brief history of this latter.

A. *ACCELEO Generator*

Acceleo [37] is a source code generator of Eclipse foundation that enables the implementation of MDA (Model driven architecture) approach to realize applications from models based on EMF. This is an implementation of the Object Management Group (OMG) standard for model-to-text (M2T) transformations.

The famous Acceleo project was born in 2006 around the Acceleo.org website by Obeo. In its first versions, Acceleo 1.0 and 1.1 were at the time under GPL licence and compatible with Eclipse 3.2 and many modelers based on EMF or UML. In 2009, for his passage to version 3, the project was admitted to the Eclipse Foundation. From this transition, Acceleo changed the language used to define generators to use the standard OMG language for model-to-text transformation. This language is an implementation of the MOFM2T standard. ACCELEO is based on the concept of template approach.

B. *Structure of ACCELEO project*

According to the model already generated shown in figure 6, the ACCELEO structure project is composed of the following principles elements:

- The "*GenerateStruts2"* package 1 presents the different generated classes (Bean and Action classes).
- The "org.eclipse.acceleo.ecore2java.files" package 2 contains the main template explained in figure 8.
- 3 Represents the main template named "*genStruts2.mtl*".
- Represents the main template named "*genStruts2.mtl*".
- 4 Represents the template of Jsp pages.
- The "*GenerateJsp*" folder 5 contains the different JSP pages generated by code generation.

Figure 7 presents the created ACCELEO project and the different packages and templates that constitute this project.

In the following section, we present the different templates necessary to implement the automatic code generation by ACCELEO and then to respond to our case study.

C. *Code Generation Implementation*

In this paper, we present the different templates permits to generate the source code from PSM model (Figure 5) already generated by applying M2M transformation in [39].

The generated PSM model implements only a CRUD application that performs the operations of listing, adding, modifying and deleting on a given entity. This use case is so common in software development that it is rare to find an application that does not do CRUD.

The aim of this paper is to implement the automatic code generation of a Java web application allowing to make CRUD on an entity model (UML class diagram) using Struts2 as a presentation framework.

The automatic code generation by ACCELEO is based necessary on the implementation of templates. In this paper, we implement the templates correspond to the Action classes and Jsp pages. The different templates needed to realize this project are listed in the following UML class diagram shown in figure 7.

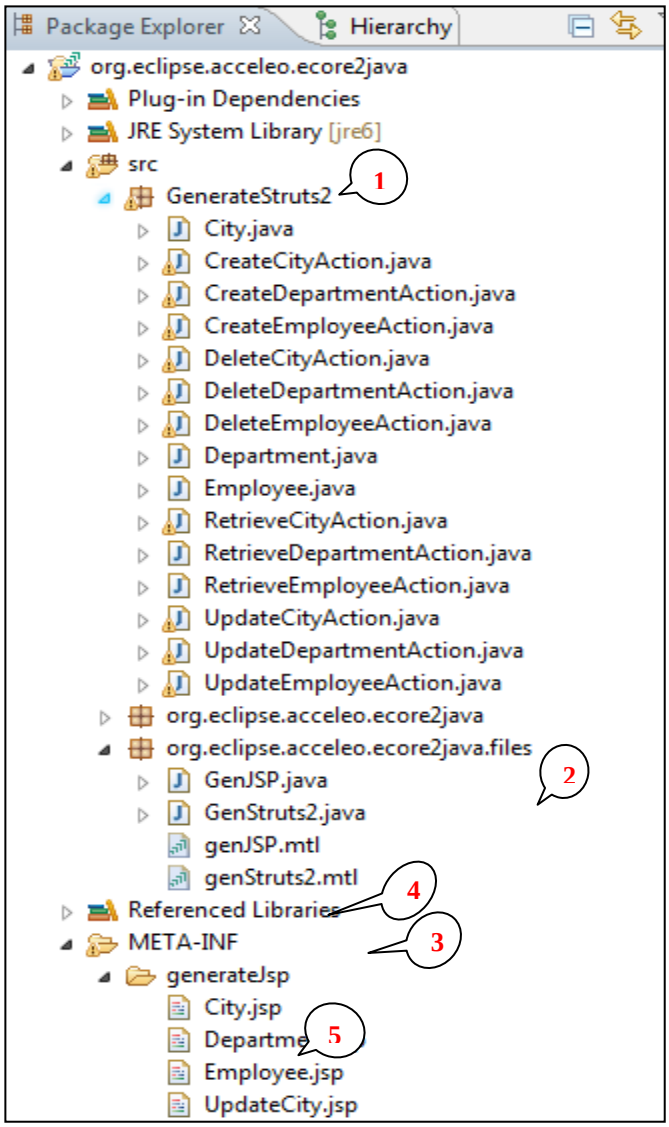

Figure 6. The Structure of Acceleo project.

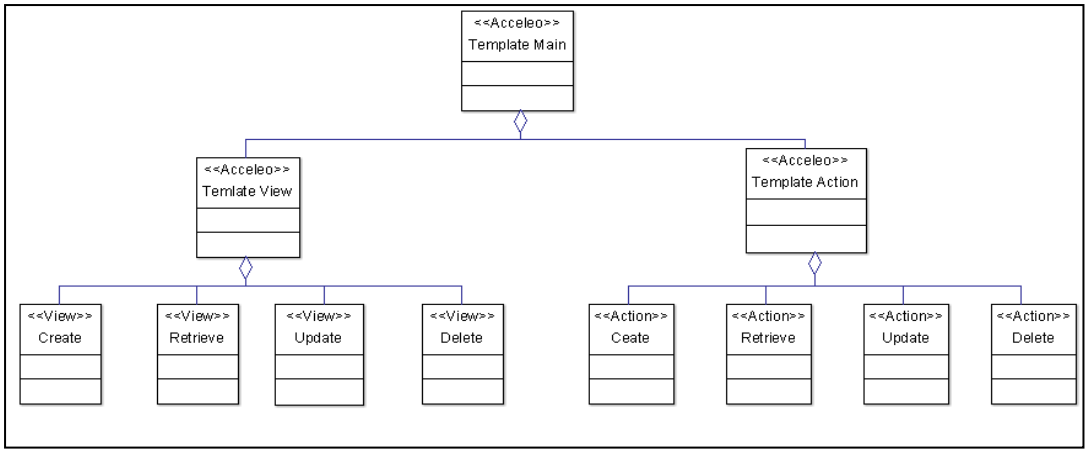

Figure 7. UML class diagram of Main template corresponding to Struts2 Model.

According to the figure 7, the principle templates are as follow: "Template Action" and "Template View". In this section, we begin by implementing the "*Template Action*" then in second hand we implement the "*Template View*".

1) Template of Actions

The different templates which constitute the templates of Actions, according to the figure 7, are as follow:

a) Template of Create Action class

This template permits to generate a Create class Action. This latter allows insert or create a new employee, a new city or a new department. The figure 8 shown below presents this template.

```
[file ('Create'.concat(aEclass.name).concat('Action').concat('.java'), false, 'UTF-8')]
[comment]//CreateAction Class[/comment]

package GenerateStruts2;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import com.opensymphony.xwork2.ActionSupport;

public class Create[aEclass.name.toUpperFirst()/]Action extends ActionSupport{
    private static final long serialVersionUID = 1L;

    [aEclass.name/]Bean mb=new [aEclass.name/]Bean();

    public [aEclass.name/]Bean getMb() {
        return mb;
    }
    public void setMb([aEclass.name/]Bean mb) {
        this.mb = mb;
    }


    public String execute()
    {

    try{
    Class.forName("oracle.jdbc.driver.OracleDriver");
    java.sql.Connection con
=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","system","admin");

    String s = "insert into [aEclass.name/] values(?,?,?)";
    PreparedStatement ps=con.prepareStatement(s);
    [for (aEAttribute : EAttribute | aEclass.eAllAttributes)]
                ps.set[aEAttribute.eType.instanceClassName.toUpperFirst()/]([i/],
mb.get[aEAttribute.name.toUpperFirst()/]());

    [/for]
    [for (aEReference : EReference | aEclass.eAllReferences)]
                ps.set[aEReference.eType.instanceClassName.toUpperFirst()/]([i/],
mb.get[aEReference.name.toUpperFirst()/]());

    [/for]

    ps.executeUpdate();
    con.commit();

        ps.close();
        con.close();

            }
        catch(Exception e){
            e.printStackTrace();
        }


            return SUCCESS;

    }

}
[/file]
```

Figure 8. Template of Create Action Class.

b) Template of Retrieve Action Class

This template permits to generate a retrieve class Action. This latter allows display all employees, departments and cities existing in the BD. The figure 9 shown below presents this template.

```
[file ('Retrieve'.concat(aEclass.name).concat('Action').concat('.java'), false,
'UTF-8')]
    [comment]//RetrieveAction Class[/comment]
    package GenerateStruts2;
    import java.sql.DriverManager;
    import java.sql.ResultSet;
    import java.sql.Statement;
    import java.util.ArrayList;
    import java.util.List;
    import javax.servlet.http.HttpServletRequest;
    import org.apache.struts2.interceptor.ServletRequestAware;
    import com.opensymphony.xwork2.ActionSupport;

    public class Retrieve[aEclass.name.toUpperFirst()/]Action extends ActionSupport
implements ServletRequestAware{
        private static final long serialVersionUID = 1L;
        HttpServletRequest request;

    public String execute()
    {
    try{
    Class.forName("oracle.jdbc.driver.OracleDriver");
    java.sql.Connection con
=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","system","admin");
    Statement st=con.createStatement();
    ResultSet rs = st.executeQuery("select * from [aEclass.name/]");

        List<[aEclass.name.toUpperFirst()/]Bean> li = null;
        li = new ArrayList<[aEclass.name.toUpperFirst()/]Bean>();
        [aEclass.name.toUpperFirst()/]Bean mb = null;

        while(rs.next())
            {
                mb = new [aEclass.name.toUpperFirst()/]Bean();
                [for (aEAttribute : EAttribute | aEclass.eAllAttributes)]

    mb.set[aEAttribute.name.toUpperFirst()/](rs.get[aEAttribute.eType.instanceClassName
toUpperFirst()/]("[aEAttribute.name/]"));
                [/for]
                [for (aEReference : EReference | aEclass.eAllReferences)]
                mb.set[aEReference.name.toUpperFirst()/](rs.get[aEReference.eType.insta
nceClassName.toUpperFirst()/]("[aEReference.name/]"));
                [/for]
              li.add(mb);
            }

        request.setAttribute("disp", li);
        rs.close();
        st.close();
        con.close();
            }
        catch(Exception e){
            e.printStackTrace();
        }
        return SUCCESS;
    }
    public void setServletRequest(HttpServletRequest request) {
        this.request = request;
    }
    public HttpServletRequest getServletRequest() {
        return request;
    }
    }
[/file]
```

Figure 9. Template of Retrieve Action Class.

c) Template of Update Action Class.

This template allows generate an Update class Action. This latter allows modify an existing employee, an existing department or an existing city. The figure 10 shown below presents this template.

```
[file ('Update'.concat(aEclass.name).concat('Action').concat('.java'), false, 'UTF-8')]
[comment]//Update Action Class[/comment]
package GenerateStruts2;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import com.opensymphony.xwork2.ActionSupport;

public class Update[aEclass.name.toUpperFirst()/]Action extends ActionSupport{
    private static final long serialVersionUID = 1L;

    [aEclass.name/] mb=new [aEclass.name/]();

    public [aEclass.name/] getMb() {
        return mb;
    }
    public void setMb([aEclass.name/] mb) {
        this.mb = mb;
    }

    public String execute()
    {

    try{
    Class.forName("oracle.jdbc.driver.OracleDriver");
    java.sql.Connection con
=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","system","admin");

    String s = "update [aEclass.name/] set [for (aEAttribute : EAttribute |
aEclass.eAllAttributes)]
        [if (i<>1)] [aEAttribute.name/]=?, [/if][/for]
            [for (aEReference : EReference |
aEclass.eAllReferences)][aEReference.name/]=?,[/for] where id=?";

    PreparedStatement ps=con.prepareStatement(s);
    [for (aEAttribute : EAttribute | aEclass.eAllAttributes)]
        [if (i<>1)] ps.set[aEAttribute.eType.instanceClassName.toUpperFirst()/]([i-1/],
rb.get[aEAttribute.name.toUpperFirst()/]());[/if][/for]

    [for (aEAttribute : EAttribute | aEclass.eAllAttributes)]
        [if (i=1)]
ps.set[aEAttribute.eType.instanceClassName.toUpperFirst()/]([aEclass.eAllAttributes->s
ize()/], rb.get[aEAttribute.name.toUpperFirst()/]());[/if][/for]

    [for (aEReference : EReference | aEclass.eAllReferences)]
        [if (i<>1)]ps.set[aEReference.eReferenceType.name.toUpperFirst()/]([i-1/],
rb.get[aEReference.name.toUpperFirst()/]());[/if][/for]

    [for (aEReference : EReference | aEclass.eAllReferences)]
        [if
(i=1)]ps.set[aEReference.eReferenceType.name.toUpperFirst()/]([aEclass.eAllReferences-
size()/], mb.get[aEReference.name.toUpperFirst()/]());[/if][/for]

    ps.executeUpdate();
    con.commit();
    ps.close();
    con.close();
            }
        catch(Exception e){
            e.printStackTrace();
        }
            return SUCCESS;
    }
}
[/file]
```

Figure 10. Template of Update Action Class.

d) Template of Delete Action Class.

This template permits to generate a Delete class Action. This latter allows delete a selected employee from the existing list of employees, departments or cities in database. The figure 11 shown below presents this template.

```
   [file ('Delete'.concat(aEclass.name).concat('Action').concat('.java'), false,
'UTF-8')]
   [comment]//DeleteAction Class[/comment]
   package GenerateStruts2;
   import java.sql.DriverManager;
   import java.sql.PreparedStatement;
   import javax.servlet.http.HttpServletRequest;
   import org.apache.struts2.interceptor.ServletRequestAware;
   import com.opensymphony.xwork2.ActionSupport;

   public class Delete[aEclass.name.toUpperFirst()/]Action extends ActionSupport
implements ServletRequestAware{
   private static final long serialVersionUID = 1L;

   HttpServletRequest request;

   public String execute()
   {

   try{
   Class.forName("oracle.jdbc.driver.OracleDriver");
   java.sql.Connection con
=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","system","admin");
   PreparedStatement ps=null;

   String cv [ '[' /][ ']' /]  =request.getParameterValues("rdel");

   for(int i=0;i<cv.length;i++)
   {
       ps=con.prepareStatement("delete from [aEclass.name/] where SNO=(?)");
       int k = Integer.parseInt(cv[ '[' /]i[ ']' /]);
       System.out.println("this is" +k);
       ps.setInt(1,k);
       ps.executeUpdate();
       con.commit();
   }

       ps.close();
       con.close();

           }
       catch(Exception e){
           e.printStackTrace();
       }
           return SUCCESS;
   }

   public void setServletRequest(HttpServletRequest request) {
       this.request = request;
   }

   public HttpServletRequest getServletRequest() {
       return request;
   }

}

   [/file]
```

Figure 11. Template of Delete Action Class.

2) Template of JSP pages

According to the figure 8, the different templates which constitute the super template "Template View" are as follow: "Template of Create Jsp Page", "Template of Retrieve Jsp page", "Template of Update Jsp page" and "Template of Delete Jsp page".

a) Template of Create Jsp page

This template permits to generate a "*Create JSP page"*. This page allows insert or create a new employee, a new city or a new department in the database. It represents the layer presentation of the application. The figure 12 shown below presents this template.

```
[comment @main /]
[file ('Create'.concat(aEclass.name).concat('.jsp'), false, 'UTF-8')]
 package generateJsp;
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<%@ taglib prefix ="s" uri="/struts-tags"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
 "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title> Create [aEclass.name.toUpperFirst()/]</title>
</head>
<body>
<center><h2>Welcome to Create [aEclass.name.toUpperFirst()/]</h2>

    <div id="formulaire">
        <s:form method ="post" action="Save_[aEclass.name.toUpperFirst()/]">
        [for (aEAttribute : EAttribute | aEclass.eAllAttributes)]

            <s:textfield name="[aEAttribute.name/]" id="[aEAttribute.name/]"
                label="[aEAttribute.name.toUpperFirst()/]" labelposition="left">
            </s:textfield>
        [/for]
        [for (aEReference : EReference | aEclass.eAllReferences)]
            <s:textfield name="[aEReference.name/]" id="[aEReference.name/]"
                label="[aEReference.name.toUpperFirst()/]" labelposition="left">
            </s:textfield>
        [/for]

            <s:submit value = "Envoyer"></s:submit>

        </s:form>
    </div>
</center>
</body>
</html>
[/file]
```

Figure 12. Template of Create JSP page.

b) Template of Retrieve Jsp page

This template permits to generate a Retrieve JSP page. This page allows display all employees, all departments or cities existing in database. The figure 13 shown below presents this template.

```
[comment @main /]
[file ('Retrieve'.concat(aEclass.name).concat('.jsp'), false, 'UTF-8')]
 package generateJsp;
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<%@ taglib prefix="s" uri="/struts-tags"  %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
 "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Retrieve [aEclass.name.toUpperFirst()/]</title>
</head>
<body>
<center><div>
    <center><h2>[aEclass.name.toUpperFirst()/] List </h2></center>

      <s:if test="%{[aEclass.name.toUpperFirst()/].size()>0}">
            <s:iterator value="list[aEclass.name.toUpperFirst()/]"><br/>

                [for (aEAttribute : EAttribute | aEclass.eAllAttributes)]
                [aEAttribute.name.toUpperFirst()/] : <s:property
 value="[aEAttribute.name/]"/><br/>
                [/for]

                [for (aEReference : EReference | aEclass.eAllReferences)]
                [aEReference.name.toUpperFirst()/] : <s:property
 value="[aEReference.name/]"/><br/>
                [/for]
        </s:if>
        <s:else>
                No items in the list
        </s:else>

    </div>
   </center>
</body>
</html>
[/file]
```

Figure 13. Template of Retrieve JSP page.

c) Template of Update Jsp page

This template permits to generate an Update JSP page. This page allows modify or update an employee, information of a department or city existing in database. The figure 14 shown below presents this template.

```
[comment @main /]
[file ('Update'.concat(aEclass.name).concat('.jsp'), false, 'UTF-8')]
 package generateJsp;
<%@ taglib prefix="s" uri="/struts-tags" %>
<%@ page import="java.util.*;" %>

<html>

<head>
<link rel="stylesheet" type="text/css" href="css/java4s.css" />
</head>

<body>
<a href="<s:url action="view.action"/>">Display [aEclass.name/]</a>
<br><br>

<b><font color="#5d8122" face="verdana">Update [aEclass.name/]</font></b>

    <s:form action="updates">

        [for (aEAttribute : EAttribute | aEclass.eAllAttributes)]

            <s:textfield label="[aEAttribute.name.toUpperFirst()/]"
 value="%{#application.a[i/]"}" name="mb.[aEAttribute.name/]"
                [if (i=1)] readonly="true" [/if] cssClass="bord">
            </s:textfield>
        [/for]

        [for (aEReference : EReference | aEclass.eAllReferences)]

            <s:textfield label="[aEReference.name.toUpperFirst()/]"
 value="%{#application.a[i/]"}" name="mb.[aEReference.name/]"
                [if (i=1)] readonly="true" [/if] cssClass="bord">
            </s:textfield>
        [/for]

    <s:submit value="Update"/>

</s:form>

</body>
</html>
[/file]
```

Figure 14. Template of Update JSP page.

d) Template of Delete Jsp page

This template permits to generate Delete JSP page. This page allows delete or destruct an employee, a department or city existing in database. The figure 15 shown below presents this template.

```
[comment @main /]
[file ('Delete'.concat(aEclass.name).concat('.jsp'), false, 'UTF-8')]
 package generateJsp;
<%@ taglib prefix="s" uri="/struts-tags" %>
<%@ page import="java.util.*;" %>

<head>
<link rel="stylesheet" type="text/css" href="css/java4s.css" />
<script type="text/javascript">
function deleteRecord()
    {
        document.fom.action="del.action";
        document.fom.submit();
    }
function editr(val)
{
    document.fom.action="update.action?fid="+val;
    document.fom.submit();
}
</script>
</head>

<a href="<s:url action="savelink.action"/>">Insert</a>
<br><br>

<table class="bordt">
<form name="fom" method="post">
<%
List l=(List)request.getAttribute("disp");
if(l!=null)
{
    Iterator it=l.iterator();

    while(it.hasNext())
    {
        GenerateStruts2.[aEclass.name/] b=(GenerateStruts2.[aEclass.name/])it.next();
        [for (aEAttribute : EAttribute | aEclass.eAllAttributes)]
        [aEAttribute.eType.instanceClassName/]
 temp[aEAttribute.name.toUpperFirst()/]=b.get[aEAttribute.name.toUpperFirst()/]();
        [/for]
        [for (aEReference : EReference | aEclass.eAllReferences)]
        [aEReference.eType.instanceClassName/]
 temp[aEReference.name.toUpperFirst()/]=b.get[aEReference.name.toUpperFirst()/]();
        [/for]
    %>
        <tr>
        <td class="bord"><input type="checkbox" value="<%= tempId %>" name="rdel"></td>
        [for (aEAttribute : EAttribute | aEclass.eAllAttributes)]
        <td class="bord"><%= temp[aEAttribute.name.toUpperFirst()/] %></td>
        [/for]

        [for (aEReference : EReference | aEclass.eAllReferences)]
        <td class="bord"><%= temp[aEReference.name.toUpperFirst()/] %></td>
        [/for]
        <td class="bord"><a href="javascript:editr('<%= tempId %>')">Edit</a></td>
        </tr>
    <%
    }
}
%>
<input type="button" value="delete" onclick="deleteRecord();">
</table>
</form>
[/file]
```

Figure 15. Template of Update JSP page.

VII. MODEL-TO-TEXT TRANSFORMATION RESULT

According to the generated PSM model (Figure 5), the different classes and JSP pages generated in this paper are listed in the figure 16 for the Action classes and in the figure 21 for the JSP pages.

In the next section, we present the different Action classes that are the CRUD Action classes.

A. *Action classes Result*

The Action class is one of Struts 2 novelties, the developer is no longer constrained to extend a given class and frees up J2EE objects to facilitate unit tests. It leaves the possibility of creating a mother class for its controllers, in which you can write all their common code (factorization) and which will implement the "*ActionSupport*" interface. The different Action classes generated in this paper are shown in the following figure.

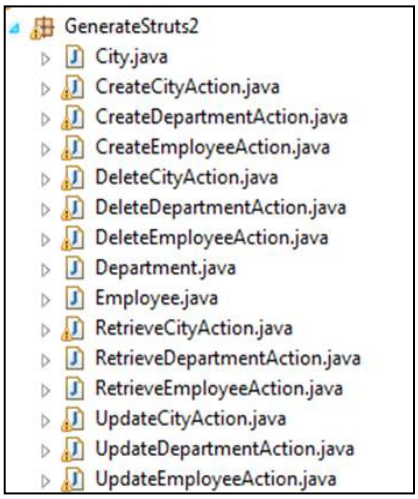

Figure 16. The different generated classes.

In the next section, we present the different classes listed in the above figure (Figure 16).

For letting this paper quite understandable and clear, we present only one example of each Action class. We note that in the application we arrived to generate all classes and all JSP pages.

1) *Create Action Class*

This class is the result of M2T transformation by using the ACCELEO generation language. The Create Action Class "*CreateActionClass*" allows insert or create a new employee, a new city or a new department. In this case, we present the "*CreateEmployeeAction.java" class*. Figure 17 shows this class.

CreateEmployeeAction.java

```
package GenerateStruts2;

import java.sql.DriverManager;

public class CreateEmployeeAction extends ActionSupport{
    private static final long serialVersionUID = 1L;

    Employee mb=new Employee();

    public Employee getMb() {
        return mb;
    }
    public void setMb(Employee mb) {
        this.mb = mb;
    }


    public String execute()
    {

    try{
    Class.forName("oracle.jdbc.driver.OracleDriver");
    java.sql.Connection con
=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","system","admin");

    String s = "insert into Employee values(?,?,?)";
    PreparedStatement ps=con.prepareStatement(s);
                ps.setInt(1, mb.getId());

                ps.setString(2, mb.getName());


    ps.executeUpdate();
    con.commit();

        ps.close();
        con.close();

            }
        catch(Exception e){
            e.printStackTrace();
        }

            return SUCCESS;

    }
}
```

Figure 17. Create Class Action:"CreateEmployeeAction.java".

2) Retrieve Action class

This class is the result of M2T transformation by using the ACCELEO generation language. The Retrieve Action Class "*RetrieveActionClass.java*" allows retrieve or lists all employees, all cities or all departments. In this case, we present the "*RetrieveDepartmentAction.java" class.* Figure 18 presents this class.

RetrieveDepartmentAction.java

```
package GenerateStruts2;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;
import javax.servlet.http.HttpServletRequest;
import org.apache.struts2.interceptor.ServletRequestAware;
import com.opensymphony.xwork2.ActionSupport;

public class RetrieveDepartmentAction extends ActionSupport{
    private static final long serialVersionUID = 1L;
    HttpServletRequest request;

public String execute()
{
try{
Class.forName("oracle.jdbc.driver.OracleDriver");
java.sql.Connection con
=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","system","admin");
Statement st=con.createStatement();
ResultSet rs = st.executeQuery("select * from Department");

    List<Department> l1 = null;
    l1 = new ArrayList<Department>();
    Department mb = null;

    while(rs.next())
        {
            mb = new Department();
            mb.setId(rs.getInt("id"));
            mb.setName(rs.getString("name"));
            //mb.setEmployee(rs.getEmployee("employee"));
        l1.add(mb);
        }

    //request.setAttribute("disp", l1);
    rs.close();
    st.close();
    con.close();
        }
    catch(Exception e){
        e.printStackTrace();
    }
    return SUCCESS;
}

}
```

Figure 18. Retrieve Class Action:"RetrieveDepartmentAction.java".

3) Update class

This class is the result of a M2T transformation by using e ACCELEO generation language. The Update Action Class allows modify or update an employee, a city or a department. In this case, we present the "*UpdateDepartmentAction"* that permits to update the information of a given Department. The figure 19 shown below presents this class.

UpdateDepartmentAction.java

```
package GenerateStruts2;

import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import com.opensymphony.xwork2.ActionSupport;

public class UpdateDepartmentAction extends ActionSupport{
    private static final long serialVersionUID = 1L;

    Department mb=new Department();

    public Department getMb() {
        return mb;
    }
    public void setMb(Department mb) {
        this.mb = mb;
    }

    public String execute()
    {

    try{
    Class.forName("oracle.jdbc.driver.OracleDriver");
    java.sql.Connection con
=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","system","admin");

    String s = "update Department set  name=?, employee=?, where id=?";

    PreparedStatement ps=con.prepareStatement(s);
    ps.setString(1, mb.getName());
    ps.setInt(2, mb.getId());
    //ps.setEmployee(1, mb.getEmployee());
    ps.executeUpdate();
    con.commit();
    ps.close();
    con.close();
            }
        catch(Exception e){
            e.printStackTrace();
        }
            return SUCCESS;
    }
}
```

Figure 19. Update class Action: "UpdateDepartmentAction.jsp".

4) Delete class Action

This class is the result of M2T transformation by using the ACCELEO generation language. The Delete Action Class "*DeleteActionClass*" allows delete an employee, a city or a department from database. In this case, we present the "*DeleteDepartmentAction.java".* Figure 20 shows this class.

DeleteDepartmentAction.java

```
package GenerateStruts2;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import javax.servlet.http.HttpServletRequest;
import org.apache.struts2.interceptor.ServletRequestAware;
import com.opensymphony.xwork2.ActionSupport;

public class DeleteDepartmentAction extends ActionSupport {
private static final long serialVersionUID = 1L;

HttpServletRequest request;

public String execute()
{

try{
Class.forName("oracle.jdbc.driver.OracleDriver");
java.sql.Connection con
=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","system","admin");
PreparedStatement ps=null;

String cv []=null;//  =request.getParameterValues("rdel");

for(int i=0;i<cv.length;i++)
{
    ps=con.prepareStatement("delete from Department where SNO=(?)");
    int k = Integer.parseInt(cv[i]);
    System.out.println("this is" +k);
    ps.setInt(1,k);
    ps.executeUpdate();
    con.commit();
}

    ps.close();
    con.close();

        }
    catch(Exception e){
        e.printStackTrace();
    }
        return SUCCESS;
}

}
```

Figure 20. Delete class Action: "DeleteDepartmentAction.jsp".

B. *Generated JSP pages*

The different JSP pages generated by ACCELEO generator is shown in figure 21.

For letting this paper quite understandable and clear, we present only one example of each JSP page. We note that we have been able to generate all the requested JSP pages according to the generated Struts2 model presented in figure 5.

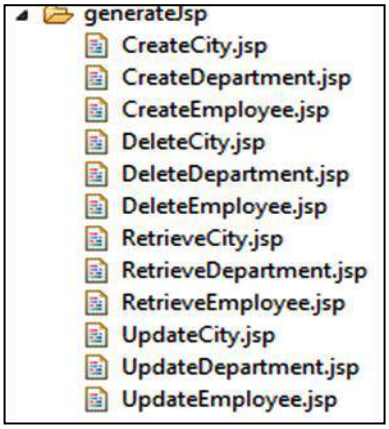

Figure 21. The generated Jsp page

5) *Create Jsp page*

This jsp page is the M2T transformation result by ACCELEO generation language. The Create jsp page "*CreateClass*" allows insert or create a new employee, a new city or a new department in database. In this case, we present the "*CreateCity.jsp" jsp page*. Figure 22 presents this jsp page.

CreateCity.jsp

```
package generateJsp;
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<%@ taglib prefix ="s" uri="/struts-tags"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
 "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title> Create City</title>
</head>
<body>
<center><h2>Welcome to Create City</h2>

    <div id="formulaire">
        <s:form method ="post" action="Save_City">

            <s:textfield name="id" id="id" label="Id" labelposition="left">
            </s:textfield>

            <s:textfield name="name" id="name" label="Name" labelposition="left">
            </s:textfield>
            <s:textfield name="department" id="department" label="Department"
 labelposition="left">
            </s:textfield>

            <s:submit value = "Envoyer"></s:submit>

        </s:form>
    </div>
</center>
</body>
</html>
```

Figure 22. Create JSP page:"CreateCity.jsp".

6) *Retrieve Jsp Page*

This JSP page is the M2T transformation result by using ACCELEO generation language. The Retrieve JSP page "*RetrieveClass.jsp*" allows retrieve or lists all employees, all cities or all departments. In this case, we present the "*RetrieveEmployee.jsp" class*. Figure 23 shows this class.

RetrieveEmployee.jsp

```
package generateJsp;
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<%@ taglib prefix="s" uri="/struts-tags"  %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
 "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Retrieve Employee</title>
</head>
<body>
<center><div>
    <center><h2>Employee List </h2></center>

      <s:if test="%{Employee.size()>0}">
            <s:iterator value="listEmployee"><br/>

                Id : <s:property value="id"/><br/>
                Name : <s:property value="name"/><br/>
                LastName : <s:property value="lastName"/><br/>

                </s:if>
                <s:else>
                    No items in the list
                </s:else>


   </div>
  </center>

</body>
</html>
```

Figure 23. Retrieve JSP page:"RetrieveEmployee.jsp".

7) Update Jsp page

This jsp page is the M2T transformation result by using ACCELEO generation language. The Update jsp page allows modify or update an employee, a city or a department. In this case, we present the "*UpdateCity.jsp"* that permits to update the information of a given Department in database. Figure 24 shows this jsp page.

UpdateCity.jsp

```
package generateJsp;
<%@ taglib prefix="s" uri="/struts-tags" %>
<%@ page import="java.util.*;" %>

<html>

<head>
<link rel="stylesheet" type="text/css" href="css/java4s.css" />
</head>

<body>
<a href="<s:url action="view.action"/>">Display City</a>
<br><br>

<b><font color="#5d8122" face="verdana">Update City</font></b>

    <s:form action="updates">


            <s:textfield label="Id" value="%{#application.a1}" name="mb.id"
 readonly="true"  cssClass="bord">
            </s:textfield>

            <s:textfield label="Name" value="%{#application.a2}" name="mb.name"
 cssClass="bord">
            </s:textfield>


            <s:textfield label="Department" value="%{#application.a1}"
name="mb.department" readonly="true"  cssClass="bord">
            </s:textfield>
    <s:submit value="Update"/>

</s:form>

</body>
</html>
```

Figure 24. Update JSP page:"UpdateCity.jsp".

8) Delete Jsp page

This page is the result of M2T transformation by using ACCELEO generation language. The Delete jsp page "*DeleteClass*" allows delete an employee, a city or a department from database. In this case, we present the "DeleteDepartment.jsp*"*. Figure 25 shows this jsp page.

DeleteDepartment.jsp

```
package generateJsp;
<%@ taglib prefix="s" uri="/struts-tags" %>
<%@ page import="java.util.*;" %>

<head>
<link rel="stylesheet" type="text/css" href="css/java4s.css" />
<script type="text/javascript">
function deleteRecord()
    {
        document.fom.action="del.action";
        document.fom.submit();
    }
function editr(val)
{
    document.fom.action="update.action?Eid="+val;
    document.fom.submit()
}
</script>
</head>

<a href="<s:url action="savelink.action"/>">Insert</a>

<br><br>

<table class="bordt">
<form name="fom" method="post">
<%
List l=(List)request.getAttribute("disp");
if(l!=null)
{

    Iterator it=l.iterator();

    while(it.hasNext())
    {

        GenerateStruts2.Department b=(GenerateStruts2.Department)it.next();
        int tempId=b.getId();
        String tempName=b.getName();

         Employee tempEmployee=b.getEmployee();

    %>

        <tr>
        <td class="bord"><input type="checkbox" value="<%= tempId %>" name="rdel"></td>
        <td class="bord"><%= tempId %></td>
        <td class="bord"><%= tempName %></td>

        <td class="bord"><%= tempEmployee %></td>
        <td class="bord"><a href="" javascript:editr('<%= tempId %>')">Edit</a></td>
        </tr>

    <%

    }
}
%>
<input type="button" value="delete" onclick="deleteRecord();">
</table>

</form>
```

Figure 25. Delete JSP page:"DeleteDepartment.jsp".

VIII. EVALUATION

In this paper, we was conducted a process of code generation by using Acceleo generator. After generating this code, we want to know the percentage of the generated code with respect to the total application code. In this case study, we obtained a very respectful percentage of code generation occurred. The generated code has reached into 100% of total code.

IX. RELATED WORK

After examining the related work concerning the automatic code generation by applying MDA approach, we can cite several works that are conducted in this domain such as: [26]-[11]-[27]-[28]-[29]-[30]-[31]-[32]-[33]-[34]-[36].

The work presented in [26] permits to generate only JSP pages and JavaBeans by applying the UWE [11] and ATL as a transformation language [24].

The work cited in [27] permits to transform the PIMs models implemented by Enterprise Distributed Object into PSMs for different services platforms. The transformation rules of this work are defined by ATL language. This work generates only the PSM model.

In [28], Billing et al., explains the different transformation rules permits to transform PIM into PSM in the EJB context. The different transformation rules of this work are defined by the approach by modeling based on QVT.

The work presented in [29] considers that MDA is a software industrialization pattern (or a software factory). The idea of this work is illustrated by a real case study in an IT services company. The main objective is to create MDA tools founded on XMI, XSLT and Visitor pattern. .It is a proposal to create MDA tools taking as base XMI, XSLT and the Visitor pattern.

In [30], the objective of this work is the model-driven development approach for E-Learning platform. Thereby, the authors implement the CIM model by analyzing business logic. And thereafter, they establish the system diagram and the robustness analysis. Thus, the authors define a transformation method from PIM to PSM layer by layer.

In the work [31] the objective is to integrate a new framework for secure Data Warehouses design by applying the approach by modeling based on QVT.

In [32], the author presents the AndroMDA approach of the community of web-based MDA [32]. This work permits to transform a PIM schemes to model by integrating a wide variety of scenarios and comes with a set of plug-ins, called cartridge.

In [33], the authors arrived to generate a MVC 2 web model from Struts framework. The meta-model of Struts is realized in first time in this work. The different transformation rules of this work are defined by ATL language in view to generate the CRUD operations from three classes Ci, Cj and Ck.

The subject of the work [34] is to generate the MVC 2 web model from the combination of UML class diagram and the UML activity diagram. The work presented in [35] is the continuation of [34], the objective of this work is to generate the code from the MVC2 web model already generated in [34] by using the JET2 generator.

In [38], the authors arrived to generate the MVC2 web model, but in this case, from the combination of UML class diagram and UML sequence diagram. This work is considering as an end-to-end code generation by using JET2 generator and based in the MDA approach.

The objective of the work presented in [36] is to generate the N-tiers PSM model by integrating Struts2, Spring IoC and Hibernate DAO frameworks. The different transformation rules of this work are defined by ATL language.

The objective of the paper [37] is to generate the Struts2 PSM model and to validate the ATL transformation rules presented in this work which was not possible in [33]-[34]-[35]-[36]-[38]. This paper describes a new validation method of ATL transformation rules and the test of this method by a case study.

Thus, we can say that the main contributions of our approach compared to others are: the use of ACCELEO generator to generate the code of Struts2 framework which was not possible in the case of [34]-[35]-[38]. This work is realized for the first time in this paper.

X. Conclusion and Future Work

Model-driven architecture (MDA) is a software development approach proposed and supported by the OMG foundation. This is a particular variant of model-driven engineering (MDE). A typical example of MDA approach is the automatic generation of source code from UML modeling, which involves combining: The UML standard, the modeling tool that implements it and the UML generation templates. The passage from the PSM to the code generation is the logical continuation of this treatment. It can be realized by generators such as these in order to produce any type of technological targets.

In this paper, we generate a MVC2 web code from Struts2 model already generated by applying an ATL transformation. This is the M2M transformation. After that, we implement the model-to-text transformation based on the MDA approach in view to obtain the code of this application from the generated model. Thereby, this transformation is started by the M2M transformation to obtain the PSM Struts2 model. The generated PSM model is an EMF model. This latter is used, in the M2T transformation, as an input model of Acceleo generator to produce automatically the necessary target application code. Finally, the code generation result was demonstrated and exemplified by a case study.

Furthermore, we plan to generate an e-commerce web code from a PSM model result of the integration of struts2, Spring IoC and Hibernate DAO. In other hand, we can extend this method for considering other frameworks like: PHP, Zend and DotNet.

References

[1] Strsus2 homepage (2017). Retrieved June 4, 2017 from http: //www.struts.apache.org/2.x/.

[2] Hibernate homepage. Retrieved July 4, 2017 from http: //hibernate.org/.

[3] Spring homepage (2017). Retrieved June 22, 2017 from http: //projects.spring.io/spring-framework/.

[4] Arnaud Cogoluègnes, Thierry Templier, Julien Dubois , & Jean-Philippe Retaillé (2nd Ed.) . "Spring par la Pratique". 2009, Eyrolles.

[5] GertiKappel, Birgit Pröll, Siegfried Reich, & Werner Retschizegger. "Web Engineering". 2006, John Wiley.

[6] Blanc, X., "MDA en action : Ingénierie logicielle guidée par les modèles ". Eyrolles, 2005.

[7] Kleppe, A., Warmer, J., & Bast, W., "MDA Explained: The Model Driven Architecture: Practice and Promise". 2003, Addison-Wesley Professional.

[8] Stefano Ceri, Piero Fraternali, Aldo Bongio, Marco Brambilla, Sara Comai, & Maristella Matera. "Designing Data-Intensive Web Applications". 2003, Morgan Kaufman.

[9] Jaime Gómez, & Cristina Cachero. "OO-H: Extending UML to Model Web Interfaces". Information Modeling for Internet Applications. 2002. IGI Publishing.

[10] Pedro Valderas, Joan Fons, & Vicente Pelechano. "From Web Requirements to Navigational Design – A Transformational Approach". Proceedings of the 5th Int. Conference of Web Engineering (ICWE'05), 2005, LNCS 3579.

[11] Koch, N., "Transformations Techniques in the Model-Driven Development Process of UWE". Proceeding of the 2nd Workshopsh of Model-Driven Web Engineering (MDWE'06). 2006, Palo Alto.

[12] Santiago Meliá, & Jaime Gomez. "The WebSA Approach: Applying Model Driven Engineering to Web Applications". Journal of Web Engineering, Vol.5, No.2, 2006.

[13] Object Management Group (OMG). Model Driven Architecture. Document update/2012-06-27, Retrieved June 12, 2017 from http://www.omg.org/mda/.

[14] OMG. (2017, June). Unified Modeling Language. Available at : http://www.uml.org/

[15] Meta Object Facility (MOF), version 2.0, Retrieved June 20, 2017 from http://www.omg.org/spec/MOF/2.0/PDF/.

[16] XML Metadata Interchange (XMI), version 2.1.1. Retrieved June 28, 2017 from http://www.omg.org/spec/XMI/.

[17] Object Management Group (OMG) (2017). UML 2 Object Constraint Language (OCL). Retrieved June 30, 2017 from http://www.omg.org/cgibin/doc?ptc/2005-06-06.

[18] Pan Yuqing, & Wang Jiuli. "Design And Implementation of Accounting E-Business Platform for Source Documents". Proceeding in the International Conference on Computer Application and System Modeling (ICCASM 2010). 2010, DOI: 10.1109/ICCASM.2010.5622916.

[19] Spring AOP homepage. Retrieved June 8, 2017 from http://static.springsource.org/spring/docs/2.5.x/reference /aop.html.

[20] Praveen Gupta, & Govil, Prof. M.C., "Spring Web MVC Framework for rapid open source J2EE application development: a case study". International Journal of Engineering Science and Technology, Vol.2, No.6, 2010, pp.1684-1689.

[21] Praveen Gupta, & Govil, Prof. M.C., "MVC Design Pattern for the multi framework distributed applications using XML, spring and struts framework". International Journal on Computer Science and Engineering, Vol.2, No.4, 2010, pp.1047-1051.

[22] MENET, L., "Formalisation d'une Approche d'Ingénierie Dirigée par les Modèles Appliquée au Domaine de la Gestion des Données de Référence". PhD thesis, Université de Paris VIII, Laboratoire

d'Informatique Avancée de Saint-Denis (LIASD), école doctorale : Cognition Langage Interaction (CLI), 2010

[23] Frédéric Jouault, & Ivan Kurtev, "Transforming Models with ATL". Proceeding in MoDELS 2005 Workshops, LNCS 3844, 2006, pp. 128 – 138, © Springer-Verlag Berlin Heidelberg.

[24] Jouault, F., Allilaire, F., Bézivin, J., & Kurtev, I., "ATL: A Model Transformation Tool". Sciences of Computer Programming-Elseiver, Vol.72, No.1-2, 2008, pp.31–39.

[25] AtlanMod (2017). Atl transformation language home page. Retrieved January 20, 2017 from http://www.eclipse.org/m2m/atl/.

[26] Kraus, A., Knapp, & A., Koch N., "Model-Driven Generation of Web Applications in UWE". Proceeding in the 3rd International Workshop on Model-Driven Web Engineering, CEUR-WS, Vol. 261, 2007.

[27] Bezivin, J., Hammoudi, S., Lopes, D., & Jouault, F., "Applying MDA approach for web service platform". Proceedings of the 8th IEEE International Entreprise Distributed Object Computing Conference (EDOC'04), 2004, pp.58-70.

[28] Bezivin, J., Busse, S., Leicher, A., & Suss, J.G., "Platform Independent Model Transformation Based on TRIPLE". In Middleware'04: Proceedings of the 5th ACM/IFIP/USENIX International Conference on Middleware, 2004, pp. 493- 51.

[29] Ndie, T. D., Tangha, C., & Ekwoge, F. E., "MDA (Model- Driven Architecture) as a Software Industrialization Pattern: An Approach for a Pragmatic Software Factories". Journal of Software Engineering & Applications, Vol.3, No.6, 2010, pp. 561-571.

[30] Cong, X., Zhang, H., Zhou, D., Lu, P., & Qin, L., "A Model- Driven Architecture Approach for Developing E-Learning Platform", Entertainment for Education. Proceeding of Digital Techniques and Systems Lecture Notes in Computer Science, Vol. 6249/2010, 2010, pp. 111-122, DOI: 10.1007/978-3-642-14533-9-12.

[31] Soler, E., Trujillo, J., Blanco, C., & Fernandez-Medina, E., "Designing Secure Data Warehouses by Using MDA and QVT". Journal of Universal Computer Science, Vol.15, No.8, 2009, pp.1607-1641.

[32] AndroMDA (2017). Retrieved January 15, 2017 from http: //www.andromda.org/.

[33] RAHMOUNI, M., & Mbarki, S., "MDA-Based ATL Transformation to Generate MVC2 Web Models".International Journal of Computer Science & Information Technology (IJCSIT), Vol.3, No.4, 2011, pp.57-70.

[34] RAHMOUNI, M., & Mbarki, S., "Combining UML Class and Activity Diagrams for MDA Generation of MVC 2 Web Applications". International Review in Computers and Software (IRECOS), Vol.8, No.4, 2013, pp.949-957.

[35] RAHMOUNI, M., & MBARKI, S., "An End-to-End Code Generation from UML Diagrams to MVC2 Web Applications". International Review in Computers and Software (IRECOS), Vol.8, No.9, 2013, pp.2123-2135.

[36] RAHMOUNI, M., & MBARKI, S., "MDA-Based Modeling and Transformation to Generate N-Tiers Web Model". Journal Of Software (JSW), Vol.10, No.3, 2015, pp. 222-238, DOI: 10.17706/jsw.10.3.222-238.

[37] Acceleo : https://www.eclipse.org/acceleo/ (Retrieved June 22, 2017)

[38] RAHMOUNI, M., & MBARKI, S., "Model-Driven Generation: From Models to MVC2 Web Applications". International Journal of Software Engineering and Its Applications (IJSEIA), Vol.8, No.7, 2014, pp.73-94, http://dx.doi.org/10.14257/ijseia.2014.8.7,07.

[39] RAHMOUNI, M., & MBARKI., S.," Validation of ATL Transformation to Generate a Reliable Web Models". International Journal of Engineering and Applied Computer Science (IJEACS), Vol. 2, No. 3, April 2017, pp. 83-91, DOI: 10.24032/ijeacs/0203/01.

AUTHORS PROFILE

M'hamed RAHMOUNI received the Diploma of Higher Approfondie Studies in Computer Science and Telecommunication from the faculty of science, Ibn Tofail University, Morocco, 2007, and Doctorat of High Graduate Studies degrees in Computer Sciences from Ibn Tofail University, Morocco, January 10, 2015 . He participated in various international congresses in MDA (Model Driven Architecture) integrating new technologies XML, EJB, MVC, Web Services, etc. and he published many papers in the MDA domain.

Samir MBARKI received the B.S. degree in Applied Mathematics from Mohammed V University, Morocco, 1992, and Doctorat of High Graduate Studies degrees in Computer Sciences from Mohammed V University, Morocco, 1997. In 1995 he joined the faculty of science Ibn Tofail University, Morocco where he is currently a Professor in Department of Computer Science. His research interests include software engineering, model driven architecture, software metrics and software tests. He obtained an HDR in Computer Science from Ibn Tofail University in 2010.