Toward automatic generation of mvc2 web applications

Samir MBARKI[1], Mohammed ERRAMDANI[2]

[1]Department of Mathematics and Computer Science, Faculty of science
Ibn Tofail University, Kenitra, BP 133, Morocco
mbarkisamir@hotmail.com

[2]Department of Management, EST
Mohammed1 University, Oujda, Morocco
mramdani69@yahoo.co.uk

Abstract. Most of server-side development technologies have emerged, since the invention of CGI in 1993. Facing this diversity and incessant improvement of Web technology, we have the feeling that we are in need of developing a tool which is able to produce the code from UML conception model. This paper examines the application of MDA approach in the engineering of web applications. Two meta-models were designed: The first one for managing UML source models, the seco2 web application models. The transformation rules and mapping algorithm were developed to generate an xml file containing all actions, forms, and forward JSP pages from class diagram that can be used to generate the required code of web application.

Keywords: Software Engineering, Model Driven Architecture, Model Driven Engineering, Meta-models, rules transformation, Web Application development

1. Introduction

The quality of web applications depends on the content, the graphical interface and the different entity structure wealth. Problems of this quality can be technical: navigability, visual presentation,…etc. In [15], we have specified that frameworks can attenuate the technical problems. Among these frameworks we find: Struts [4] [11], Cocoon [3], Turbine [2]. Other Frameworks can help in the achievement of the graphical user interface [13].

In the majority of web applications, we note that operations recur, namely create, remove, update and display of the different objects. We are especially interested in the display of objects of a given class, basing on information of another object of another class provided that the two classes are connected via associations on the class diagram.

Our work is very useful for users called to manipulate some information linked to each other in an arborescent structure where the display of information depends on another (display, as part of a web application, employees of a service, of a division, of a direction, etc.).

In this sense, we present in this paper a tool that transforms a class diagram to a MVC2 web application (particularly Struts). The result of this transformation is an XML file that can be used to generate the required code for Web application fulfilling the needs cited above.

This paper is organized as follow: we begin in the first section with an introduction. The second section permits to develop MDA as architecture. The point is particularly focused on the different models CIM, PIM and PSM (In the setting of our work, the model PIM is represented by the source classes diagram, and the PSM model is represented by the model MVC target. The third section presents the MVC paradigm and the setting up of MVC under the shape of a framework, such as: struts. In the fourth section, we elaborate the UML and MVC meta-models. The transformation rules of UML source model to the MVC target model, the transformation algorithm and the results of this transformation are presented in the fifth section. The final section concludes this paper.

2. Model Driven Architecture (MDA)

In November 2000, the OMG (consortium of more than 1000 enterprises), has introduced the MDA approach. Its aim was to provide a new unified vision to conceive applications by separating the business logic of the enterprise from all technical platforms. Indeed, the business logic is stable and is subject to a few changes over time, in opposition to the technical architecture. It is therefore obvious to separate the two in order to face the complexity of information systems and excessive costs of migration technology. The MDA standard should provide an opportunity to stop the stacking of technology that requires special skills to keep varied and diverse systems cohabit.

The MDA architecture is divided into four layers. In the first layer, we find the standard UML (Unified Modelling Language), MOF (Meta-Object Facility) and CWM (Common Warehouse Meta-model). In the second layer, we find a standard XMI (XML Metadata Interchange), which enables the dialogue between middlewares (Java, CORBA, .NET and web services). The third layer contains the services that manage events, security, directories and transactions. The last layer provides frameworks which are adaptable to different types of applications namely Finance, Telecommunications, Transport, medicine, E-commerce and Manufacture, etc.) [7].

The principle key of MDA consists in the use of models in different phases of the development cycle of an application. Specifically, MDA recommends the development of the CIM (Computation Independent Model), PIM (Platform Independent Model) and the PSM (Platform Specific Model). The main objective of MDA is to develop perennial models independent of technical details of execution platforms (J2EE, .NET, PHP or otherwise) in order to enable the automatic generation of all application code and to obtain a significant gain in productivity [6] [9] [10].

3. The MVC pattern

Formerly, web applications were very simple and a technology was used so as to develop them: Common Gateway Interface (CGI). As soon as the applications become more complex, the defects and limits of this technology have emerged: slowness and considerable consumption of memory.

Besides, they are not correctly coded: The code used for the presentation is always mixed with the code representing the business logic. So as to face such constraints, the editors have proposed solutions like Application servers. This term gathers web servers and enterprise application servers.

A servlet is a Java program that be activated at the browser's request, and interacts with the database or other programs to provide an HTTP reply to the browser. Contrary to the CGI, a servlet is loaded only once in the memory [12]. The servlet receives an Http request and handles it by interrogating the information system and returning an HTML response, and that's where the problem resides. Therefore, the J2EE platform applies the architecture Model, View, Control (MVC) [1]. In this paradigm, the model represents the information system consisting of java beans. The view represents the HTML pages returned to the user. The view consists of JavaServerPage (JSP). Control is the glue between the two and it is composed of servlets. In short, appearing from the early 80 with smalltalk, MVC was widespread in the field of object development.

It was generalized to Web development. This model provides the IHM Layer with all the guarantees of maintainability and reusability of expected software [14] [16] [17].

3.1. The struts framework

Is the framework for writing web applications that supports the MVC architecture. Struts structures all the components of the application into a unified whole. These components are servlets, javaserver pages, javabeans, business logic. To make the necessary connections between the components, we need to configure and set up the struts-config.xml and web.xml files, the two principal configuration files for each struts application [11]. The components of struts are:

- **Business model**: This model corresponds to the application business logic. It is often described as JavaBeans or EJB components. Struts doesn't provide any business classes [11].
- **Controller**: The control is implemented as a component servlet via the org.apache.struts.action. ActionServlet class which extends the javax.http.HttpServlet class [5] [12]. The controller performs the following tasks when receiving a request:
  - Matches URI of the request to the appropriate ActionMapping class (an ActionMapping object provides the "ActionServlet" servlet with a description of a particular action instance).
  - Maps the request towards the name of the Action class owing to the "ActionMapping" information.
  - Creates or finds an ActionForm instance and decorates properties of this instance from requested parameters.
  - Calls the method execute() on the suitable class and declared Action in the "ActionMapping" class and transmitting to it the ActionForms, ActionMapping, request and response objects (the latter two parameters are respectively instances of HtppServletRequest and HttpServletResponse).
  - Transmits the returned value to the resource specified in the "ActionForward".
- **View**: The view component corresponds to the tier presentation of a struts application.

4. Meta-models UML source and MVC2 target

To develop the transformation algorithm between source and target model, we present in this section, the various meta-classes forming the UML source meta-model and the MVC meta-model. The source meta-model structures a simplified UML model based on packages containing data types and classes. Those classes contain typed properties and they are characterized by multiplicities (upper and lower). The classes are composed of operations with typed parameters. Figure 1 illustrates the source meta-model:

- ***UmlPackage***: is the concept of UML package. This meta-class is connected to the meta-class ***Classifier***.
- ***Classifier***: This is an abstract meta-class representing both the concept of UML class and the concept of data type.
- ***Class***: is the concept of UML class.
- ***DataType***: represents UML data type.
- ***Operation***: is used to express the concept of operations of a UML class.
- ***Parameter***: expresses the concept of parameters of an operation. These are of two types, ***Class*** or ***DataType***. It explains the link between ***Parameter*** meta-class and ***Classifier*** meta-class.
- ***Property***: expresses the concept of properties of a UML class. These properties are represented by the multiplicity and meta-attributes upper and lower. A UML class contains properties that may be of primitive type (***DataType***) or references to other classes. This explains the link between ***Property*** meta-class and ***Classifier*** meta-class. The association between ***Class*** meta-class and ***Property*** meta-class expresses the fact that a class is composed of properties.

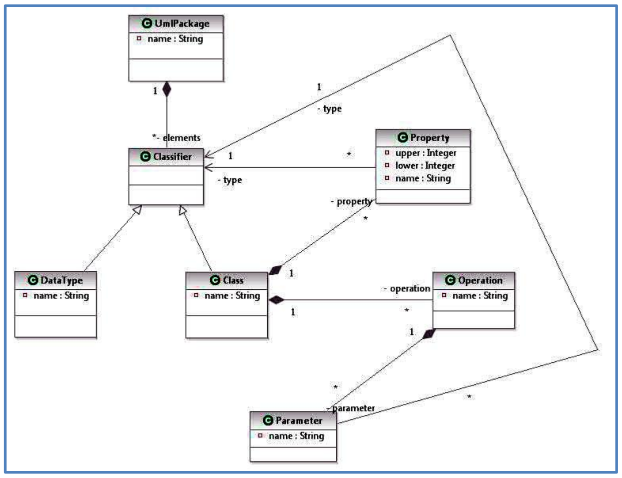


Figure 1: Simplified UML meta-model

Figure 2 illustrates the first part of the target meta-model. This meta-model represents a simplified version of the schemas of relational databases. We present here the different meta-classes to express the concept of tables of a relational database:

- ***Schema***: represents the concept of relational databases schema.
- ***Table***: is the concept of table in the relational databases. It contains a meta-attribute ***name*** which represents the table name in the database. The meta-class is connected by a meta-association to the meta-class ***Column.***
- ***Column***: is the concept of column in the database tables. In order to express the fact that column can be a primary key, or a foreign key, we decided to build a meta-class by the type of column.
- ***PrimaryKey***: expresses a column as a primary key.
- ***ForeignKey***: expresses a column as a foreign key. The meta-class is connected by a meta-association to the meta-class ***Table*** to express the fact that the foreign key references the corresponding table.

This meta-model structures models representing relational databases. The latter consists of several tables, themselves made up of typed columns.

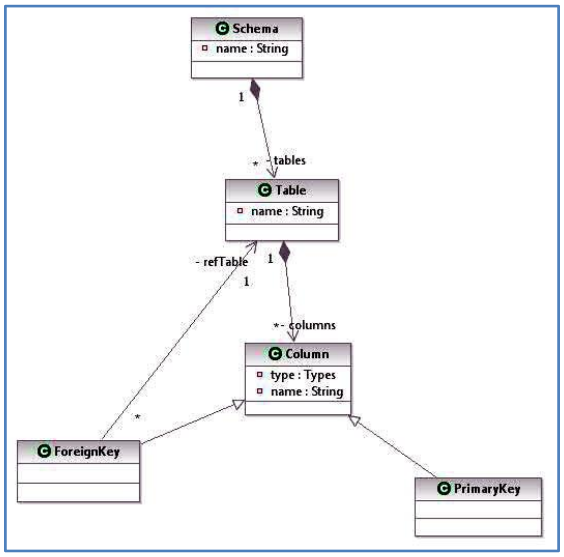


Figure 2: A simplified meta-model of a relational database

Figure 3 illustrates the second part of target meta-model. The meta-model is the business model of the application to be processed. In our case, we opted for components such as Bean. We recall that struts doesn't provide specific business classes. We present here the different meta-classes to express the concept of bean contained in the ***PackageModel*** package.

- ***Bean***: represents the concept of bean. The latter extends the meta-class ***Class***. The beans represent objects in the area of application. These objects communicate with the tables of relational database, which explains the meta-association with meta-class ***Table***.
- ***Vector***: Business class in the field of application has an operation that returns a list of all items of this bean stored in the database. We use the ***Vector*** meta-class to represent the list.
- ***Action***: is the concept of action (See section 3). Class Action contains its own processing of the application, hence it should be linked to the various beans.
- ***HttpRequest***: is the concept of ***HttpServletRequest*** classes (See section 3).
- ***HttpResponse*** represents the concept of ***HttpServletResponse*** classes (See section 3).

This meta-model structures the models representing the business logic of the target application. This logic is essentially made up of Bean components.

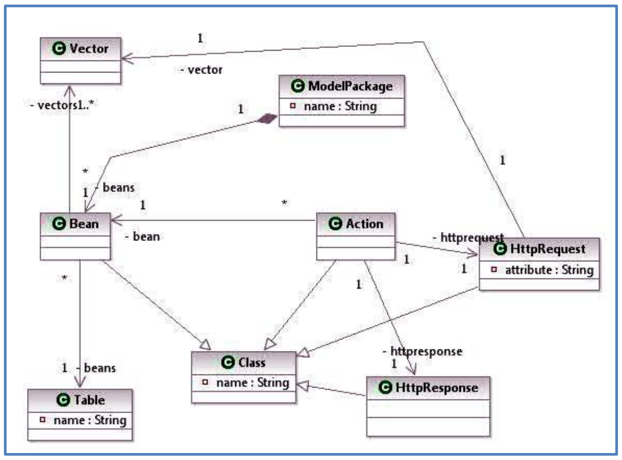


Figure 3: A simplified meta-model of the model package

Figure 4 illustrates the third part of the target meta-model. This meta-model represents the view package (see section 3):

- ***JspPage***: depicts a Jsp page (See section 3). An action class may be called from a hyperlink in a Jsp. This explains the link between the Jsp page and Action class. The link between ActionForward and Jsp page is trivial. ActionForm is linked to Jsp page because it contains the information that would be transmitted in the request and then filled in the actionForm (See section 3). The link between Jsp page and HttpRequest expresses the fact that the Jsp page can use the information contained in an HttpRequest object.

This meta-model structures the models representing the view application. In this model, the Servlet invokes the execute( ) method on the instance of the action class. This method completes its processing and then calls the mapping.findforward( ) method with a return to a specified Jsp page.

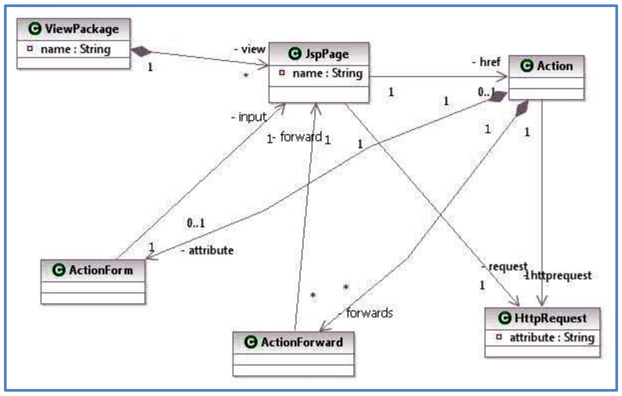


Figure 4: A simplified meta-model of the view package

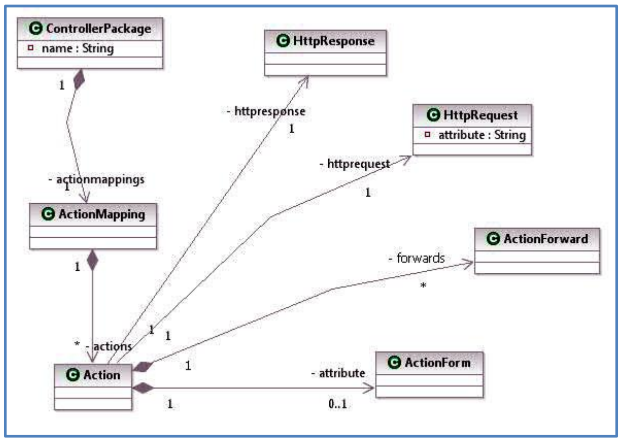


Figure 5: A simplified meta-model of the controller package

Figure 5 illustrates the fourth part of target meta-model. This package represents the controller meta-model (see section 3):

- ***ActionMapping***: Represents the concept of ActionMapping classes (See section 3). An ActionMapping class contains information to deploy of a class Action. This explains the connection with the meta-class ***Action***.
- ***Action***: (already seen at the ModelPackage meta-model).
- ***ActionForm***: represents the concept of ActionForm classes (see section 3). An ActionForm represents a form containing the parameters of the request from the view (ViewPackage). This object is used by Action Class (This is particularly one of the four parameters of the operation execute()), which explains the link with the metaclass Action.

This meta-model structures the models representing the application's controller. The controller is responsible for the receipt of requests transmitted by the client, with the invocation of the Action class, and therefore interacts with business model and coordinates with a view by returning it to the client.

5. The mapping rules

5.1. The context

The purpose of this formalism is to develop a mathematical presentation of a class diagram with its constituents: the classes, their members as well as interactions between different classes. Such formalism will be very useful for the development of an incremental and easily scalable transformation algorithm.

We rely on the fact that it is possible to know all the attributes and all basic methods for a given problem. This axiom is not always checked. We must remember that most of the object design methods presuppose the knowledge of the attributes to construct the initial class diagram and the basic methods [8]. This situation remains largely valid in the case of a problem of re-engineering.

Consider all business classes and let us call them C_i with $1 \leq i \leq p$. p is the total number of such classes.

Each class has:

- Properties $P_{i,j}$ with $1 \leq j \leq n_i$. n_i is the number of C_i's properties.
- Operations $M_{i,j}$ with $1 \leq j \leq m_i$. m_i is the number of C_i's operations.

Let $M_{i,j}$, $1 \leq j \leq m_i$ a C_i's basic operation . We can classify it in one of the following cases:

Case 1: $\exists$ $P_{k,j}$ (k may be different from i) such that $P_{k,j}$ is updated by $M_{i,j}$.

Case 2: $\exists$ $P_{k,j}$ (k may be different from i) such that $P_{k,j}$ is consulted by $M_{i,j}$.

Case 3: $\exists$ $P_{k,j}$ (k may be different from i) such that $P_{k,j}$ is updated and consulted by $M_{i,j}$.

In all three cases, the class C_i is related to the class C_k on the class diagram.

As for the database, we would like to clarify that for one class, we can add other operations that have the connection to the database. We also find that basic CRUD (Create, Remove, Update, and Display) operations, in the database necessarily to belong to some category 1, 2 or 3.

In this work, we limit ourselves to the operations of category 2. Formally, we can fill in the matrix of $M_{i,j}$ consultation operations compared to the properties $P_{i,j}$, thereby putting the value 1 at the intersection of the column $j + \sum_{r=1}^{i-1}(m_r)$ and the row $l + \sum_{r=1}^{k-1}(n_r)$. Note that at the same time the number of columns of the matrix is $\sum_{r=1}^{p}(m_r)$ and the number of lines is $\sum_{r=1}^{p}(n_r)$.

Example : Consider the following class diagram:

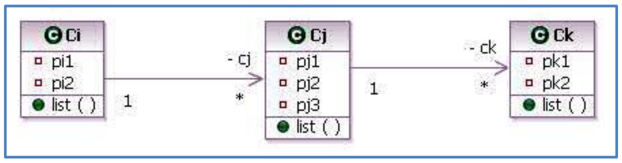


Figure 6: An example of class diagram

Compared to this example, each class has a single operation of category 2 (list), i.e. a consultation operation. The matrix on these operations is shown below. In this matrix, in addition to the representation of classes and its members (properties and operations), there is also a representation of associations between classes. The first rectangle in the matrix shows the link between instances of classes C_i and C_j.

$$
\begin{array}{cccc}
 & list\_C_i & list\_C_j & list\_C_k \\
P_{i,1} & 1 & 1 & 0 \\
P_{i,2} & 1 & 0 & 0 \\
P_{j,1} & 0 & 1 & 1 \\
P_{j,2} & 0 & 1 & 0 \\
P_{j,3} & 0 & 1 & 0 \\
P_{k,1} & 0 & 0 & 1 \\
P_{k,2} & 0 & 0 & 1
\end{array}
$$

The matrix representation is established, and below we present the model of view as well as the relationship with the business model. Either Page $ListC_iPage.jsp$ displaying various objects of Class C_i, stored in a database. If k_i represents the number of objects of class C_i in the database then the page $ListC_iPage.jsp$ will be displayed as follow:

$$
\begin{array}{ccc}
P_{i,1_1} & P_{i,2_1} & \underline{List\ C_j} \\
P_{i,1_2} & P_{i,2_2} & \underline{List\ C_j} \\
\cdots & \cdots & \cdots \\
P_{i,1_{k_i}} & P_{i,2_{k_i}} & \underline{List\ C_j}
\end{array}
$$

- Using the button $ListC_j$ at the line L $(1 \leq L \leq k_i)$ causes the invocation of the URL represented by the action attribute i.e. $ListC_j$.
- The controller processes the request by accessing the object $ListC_jForm$. It affects to its attributes members the data present in the request and then place this object $ListC_jForm$ in the session under the name

$ListC_j Form$ (It is a single information to be sent in the request namely $P_{L,1}$, which is unique because it is associated with the primary key of the class C_i).

- The controller searches on the file $struts-config.xml$ for an entry $< action-mappings >$ with an attribute $path = ListC_j$ ($ListC_j$ is the name of the URL cited above)
- The controller creates an instance of $ListC_j Action$ class corresponding to the value of the attribute type and an instance of the class $AcionMapping$ containing <action> data.
- The controller invokes the method $ListC_j Action.execute()$ with four arguments (see section 3). This method completes its processing and then calls the method $mapping.findforward()$.
- $ListC_j Action$ returns the object $ActionForward$ to the controller, which in turn transmits the request to the view selected for the presentation, this is the $ListC_j Page.jsp$ page.

This process can be extended not only for two classes C_i and C_j, but also for a series of n classes $(C_i)_1^n$. We present this extension in the following algorithm:

5.2. Algorithm transformation

By source model, we mean model containing the various classes of our business model. The elements of this model are primarily classes. The following list describes the various objects handled in the algorithm:

$V\acute{}$: Vector. We put in this vector, the operations names of the current class.

V": Vector. We put in this vector, the classes which are transformed.

M : We put in the matrix, the action class and jsp page mapping the success forward.

e : This variable is used to browse the different classes of source model.

m_i : This variable is used to browse the different operations of element e.

$acmp$: an object of type ActionMapping

cp : an object of type ControllerPackage

vp : an object of type ViewPackage

Algorithm

```
input up:UmlPackage
output sp StrutsProjectPackage
begin
   sp=transformationRuleOne(up)
   for each e ∊ source model
      if e is class
         put operations of e in V′
         if V′ ∍ list and e ∉ V″
            x=transformationRuleTwoList(e)
            link x to acmp
         end if
         for each property p in e
            if p is class
               empty V′
               put operations of p in V′
               if V′ ∍ list
   x=transformationRuleThreeList(e)
                  link x to acmp
               end if
            end if
         end for
      end if
   end for
end
function
transformationRuleOne(up:UmlPackage):Struc
tsProjectPackage
begin
   create StrutsProjectPackage sp
   create ModelPackage mp
   create cp
   create vp
   link acmp to cp
   link cp to sp
   link vp to sp
   link mp to sp
   return sp
end
function
transformationRuleTwoList(e:Class):Action
begin
   create Action action
   put e in V″
   create ActionForward actionForward
   create PageJsp page
   put e and page in M
   link page to vp
   link page to actionForward
   link actionForward to action
   return action
end
function
transformationRuleThreeList(prop:Property)
:Action
begin
   create Action action
   $C_i$,$C_j$:Class
   $C_j$ is the prop type
   $C_i$ is the prop class
   Create ActionForm actionForm
   There  exists  a  unique  k  such  as
M[k,1]=$C_i$
   page=M[k,2]
   actionForm.input=page
   actionForm.attribute=action
```

```
    create ActionForward actionFwd
    page2:PageJsp
    put $C_j$ and page2 in M
    link page2 to vp
    link page2 to actionFwd
    link actionFwd to action
    put $C_j$ in V”
    return action
end
```

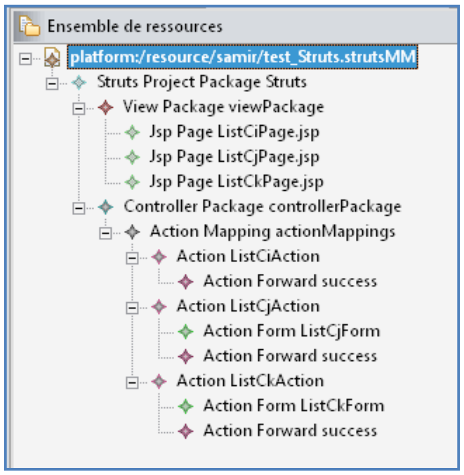
5.3. Results

The implementation of mapping rules can be performed in three manners: by programming, by template or by modelling [6]. The programming approach is the most widely used because it’s best supported by development tools. In this present work, we opt for programming approach using the Eclipse EMF framework. The advantage of EMF is the ability to develop any model transformation using Java as programming language and interfaces manipulating models.

We first develop ECORE models matching our two meta-models source and target. Then we implement the transformation algorithm (see sub-section 5.2) using interfaces manipulating models generated from ECORE models.

To validate our transformation algorithm, we have conducted several tests. For illustration, we consider the class diagram in figure 6. After applying the transformation on the UML model, composed by Classes C_i, C_j and C_k, we generate the target in figure 7 and presented by the XML file below:

```
<?xml version="1.0" encoding="ASCII"?>
<strutsMM:StrutsProjectPackage
xmi:version="2.0"
xmlns:xmi="http://www.omg.org/XMI"
xmlns:strutsMM="http:///strutsMM.ecore"
name="Struts">
  <vPack name="viewPackage">
    <view name="ListCiPage.jsp"/>
    <view name="ListCjPage.jsp"/>
    <view name="ListCkPage.jsp"/>
  </vPack>
  <cPack name="controllerPackage">
    <actionmappings name="actionMappings">
      <actions name="ListCiAction">
        <forwards name="success"
forward="//@vPack/@view.0"/>
      </actions>
      <actions name="ListCjAction">
        <attribute name="ListCjForm"
input="//@vPack/@view.0"/>
        <forwards name="success"
forward="//@vPack/@view.1"/>
      </actions>
      <actions name="ListCkAction">
        <attribute name="ListCkForm"
input="//@vPack/@view.1"/>
        <forwards name="success"
forward="//@vPack/@view.2"/>
      </actions>
    </actionmappings>
  </cPack>
</strutsMM:StrutsProjectPackage>
```



```
Ensemble de ressources
platform:/resource/samir/test_Struts.strutsMM
  Struts Project Package Struts
    View Package viewPackage
      Jsp Page ListCiPage.jsp
      Jsp Page ListCjPage.jsp
      Jsp Page ListCkPage.jsp
    Controller Package controllerPackage
      Action Mapping actionMappings
        Action ListCiAction
          Action Forward success
        Action ListCjAction
          Action Form ListCjForm
          Action Forward success
        Action ListCkAction
          Action Form ListCkForm
          Action Forward success
```

Figure 7: Generated struts model

The first element in the XML file is: vPack which includes the three jsp pages, namely ListCiPage.jsp, ListCjPage.jsp and ListCkPage.jsp. Then comes the element cPack which contains a single element ActionMapping. It also contains three action elements whose names are successively: ListCiAction, ListCjAction and ListCkAction.

The element action: ListCiAction contains only one element forward with the attribute forward = = / / @ vPack / @ view.0, i.e. jsp page: ListCiPage.jsp. This is because in the element vPack, there are three elements:

```
<vPack name="viewPackage">
    <view name="ListCiPage.jsp"/>
    <view name="ListCjPage.jsp"/>
    <view name="ListCkPage.jsp"/>
</vPack>
```

Thus //@ vPack /@view.1 means ListCjPage.jsp and //@vPack/@view.2 means ListCkPage.jsp.

The element action: ListCjAction contains two elements: the element attribute to indicate the entry form for this action, it is the form ListCjForm. This form is topped by reports of page ListCiPage.jsp because the attribute is set to input the value //@vPack/@view.0. The sub element is put forward to the value //@vPack/@ view.1, ie jsp page: ListCjPage.

The element action: ListCkAction contains two elements: the element attribute to indicate the entry form for this action, it is the form ListCkForm. This form encapsulates the ListCjPage informations because the input attribute is set to the value //@vPack/@view.01. The forward attribute is set to the value //@vPack/@view.2, ie jsp page: ListCkPage.

6. Conclusion and perspectives

We applied the MDA approach in web applications engineering. This particularly generates the makings of a web application on the basis of a UML class diagram. The latter is built on the basis of different attributes of the information system. The generation process will provide an opportunity for the user to add, edit, delete, and especially display the various objects he needs. He must be able to display objects of a given class, based on information from another object of another class provided the two classes are connected via associations at the class diagram mentioned above.

To achieve this, we first develop the source metamodel managing UML class diagrams. At the target metamodel, we have developed all metaclasses needed to be able to generate an application respecting a MVC2 architecture.

The mapping rules were developed and put together in a transformation algorithm. This algorithm makes it possible to browse the source class diagram and generate through these rules, an XML file containing all actions, forms, and then forwards jsp pages that can be used to generate the necessary code of the target application.

This is very useful when dealing with related information between them in a tree structure and the display of information depends on another. This work can be extended to support advanced aspects of the content of Web pages to produce a web application from start to finish, i.e. providing the user's interface part at a will and appropriate treatment responding to requests.

In perspective, this work should be extended to allow the generation, in addition to the configuration files, of other components of the Web application: model, view, controller and their constituents. Emphasis should be placed on the support of other CRUD methods such as create, remove and update. After we can consider integrating other execution platforms like PHP and .NET.

References

[1] D. Alur, J. Crupi, D. Malks, Core J2EE Patterns: Best Practices and Design Strategies, Prentice Hall, 2003

[2] Apache Jakarta Project: Jakarta Turbine Web Application Framework http://jakarta.apache.org/turbine

[3] Apache Jakarta Project: The Apache Cocoon Project. http://cocoon.apache.org

[4] Apache Software Foundation: The Apache Struts Web Application Software Framework. http://struts.apache.org

[5] H. Bergsten, JavaServer Pages, 3rd edition, O'Reilly, 2004

[6] X. Blanc, MDA en action : Ingénierie logicielle guidée par les modèles. 1st edition, 270 pages, 2005

[7] P-A. Caron, Spécialisation d'un environnement de conception de systèmes flexibles aux Environnements Informatiques pour l'Apprentissage Humain, Mémoire de DEA, Université des Sciences et Technologies de LILLE, 2 july 2003, 53 p. http://noce.univ-lille1.fr/cms/uploaddocs/ RapportdeStageDEAInformatiquever8.pdf

[8] J-L. Cavarero, R. Lecat. La conception orientée objet, évidence ou fatalité, Ellipses, 2000

[9] S. Cook, Domain-Specific Modeling and Model Driven Architecture. MDA Journal, pages 1-10, January 2004.

[10] J. M. Favre. Towards a Basic Theory to Model Driven Engineering. UML 2004 - Workshop in Software Model Engineering (WISME 2004), 2004.

[11] J. Goodwill, Mastering Jakarta Struts, Wiley edition, 2002

[12] J. Hunter, W. Crawford, Java Servlet Programming, 2nd edition, O'Reilly, 2001

[13] Java Server Faces Home Page. http://java.sun.com /j2ee/javaserverfaces/index.jsp

[14] Y-P. Kontogiannis, K. Lau, Transforming legacy Web applications to the MVC architecture, Software Technology and Engineering Practice, 2003. Eleventh Annual International Workshop on Volume Issue: 19-21 Sept. 2003 Page(s): 133 – 142

[15] A. J. Offutt, Quality Attributes of Web Software Applications. IEEE Software, Special Issue on Software Engineering of Internet Software, 19(2): 25-32, 2002

[16] Y. Ping et al., Migration of legacy web applications to enterprise Java™ environments net.data® to JSP™ transformation, Proceedings of the 2003 conference of the Centre for Advanced Studies on Collaborative Research, Toronto, Ontario, Canada, Pages: 223 - 237

[17] Y. Ping et al., Refactoring Web sites to the controller-centric architecture, Software Maintenance and Reengineering, 2004. CSMR 2004. Proceedings. Eighth European Conference, 24-26 March 2004 Page(s): 204 - 213