Transformation by Modeling MOF 2.0 QVT: From UML to MVC2 Web model

REDOUANE ESBAI[1]
MOHAMMED ERRAMDANI[2]
SAMIR MBARKI[3]
IBTISSAM ARRASSEN[4]
ABDELOUAFI MEZIANE[5]
MIMOUN MOUSSAOUI[6]

MATSI Laboratory, EST
Mohammed First University, Oujda, Morocco
[1]es.redouane@gmail.com
[6]moussaoui@est.univ-oujda.ac.ma

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

Department of Computer Sciences, Faculty of Science
Ibn Tofail University, Kenitra, BP 133, Morocco
[3]mbarkisamir@hotmail.com

Department of Mathematics and Computer Sciences, Faculty of Science
Mohammed First University, Oujda, Morocco
[4]arrassen@yahoo.com
[5]abdelouafi_meziane@yahoo.fr

Abstract. The continuing evolution of business needs and technology makes Web applications more demanding in terms of development, maintenance, and management. To cope with this complexity, several frameworks have emerged. Given this diversity of solutions, the generation of a code based on UML models has become important. This paper presents the application of the MDA (Model Driven Architecture) to generate, from the UML model, the Code following the MVC2 pattern (Model-View-Controller) using the standard MOF 2.0 QVT (Meta-Object Facility 2.0 Query-View-Transformation) as a transformation language. This standard defines the meta-model for the development of model transformation. The transformation rules defined in this paper can generate, from the class diagram, an XML file containing the Actions, the Forms, and JSP pages. This file can be used to generate the necessary code of a web application.

Keywords: MDA, model transformation, MVC 2, transformation rules, MOF 2.0 QVT, meta-model, OCL.

(Received January 11st, 2011 / Accepted July 16th, 2011)

1 Introduction

In recent years many organizations have begun to consider MDA as an approach to design and implement enterprise applications. The key principle of MDA is the use of models at different phases of application development by implementing many transformations. These changes are present in MDA, and help transform a CIM (Computation Independent Model) into a PIM (Platform Independent Model) or to obtain a PSM (Platform Specific Model) from a PIM.

MVC2 is a programming scheme that takes into account the entire architecture of a program. It categorizes the different types of objects that make up the application into three categories: The model manages the behavior and data of the application domain, the view corresponding to the interface with which users interact, and the Controller that supports event management synchronization to update the model. This pattern saves time for maintenance as well as upgrading and greater flexibility to organize the development of different developers (independent data, display and actions). Many frameworks that implement the MVC2 pattern have emerged; for instance: Struts [1], PureMVC [4], Gwittir [3], SpringMVC [5], Zend [6], ASP.NET MVC2 [2]. Struts remains the most mature and highly trusted solution among developers.

Mbarki and Erramdani [26, 27], both source and target meta-models have been developed. The first corresponds to a specific PIM meta-model class diagram, and the second is a PSM meta-model for MVC2 web application. The development was done via RSM (Rational Software Modeler) based on a programming approach. This means that programming transformations models was done in the same way as programming computer applications. This paper aims to rethink the work presented in [26, 27]. However, we develop the transformation rules using the MOF 2.0 QVT standard to generate an XML file which contains actions, forms and JSP pages used to produce the code for the target application. The advantage of this standard is the bidirectional execution of transformation rules.

This paper is organized as follows: related works are presented in the second section, the third section defines the MDA approach, and the fourth section presents the MVC2 model and its implementation as a framework, Struts in this case. The transformation language MOF 2.0 QVT and the language of OCL constraints are the subject of the fifth section. In the sixth section, we present the UML and MVC2 meta-models. In the seventh section, we present the transformation rules using MOF 2.0 QVT from UML source model to the MVC2 target model. The last section concludes this paper and presents some perspectives.

2 Related Work

A much relevant work on meta-modeling was completed in 2007 [22] in which the authors have developed a meta-model for web needs. This meta-model takes into account concepts such as usage cases. The authors have developed transformation rules, but the main purpose of this work was the use of this meta-model as a CIM to turn it into a PIM and then to a PSM.

Two other works followed the same logic and have been the subject of two works [20, 24]. A meta-model for Ajax was defined using AndroMDA tool. The generation of Ajax code has been illustrated by an application CRUD (Create, Read, Update, and Delete) that manages people.

Kraus, Knapp and Koch [25] show how to build JSP pages and JavaBeans using the UWE [31], (UML-based Web Engineering) and the ATL transformation language [15].

Nasir, Hamid and Hassan [30] have presented an approach to generate a code for the .Net application `Student Nomination Management System`. The method used is WebML and the code was generated by applying the MDA approach, but the creation was not done according to the .Net MVC2 logic.

The work presented by Amen, Abdelaziz and Samir [13] aims at providing a generic approach to automate the translation of conceptual models' integrity constraints to the relational context of the MDA approach. To do this, the authors proposed a transformational model based on the UML meta-model. The rules of that transformation are described by the graphical notation of QVT-Relations language.

Oberortner, Vasko and Dustdar [32] have examined the safety aspects. A meta-model was developed to integrate the roles of users to access various pages of the Web application. Each page contains navigation rules and each rule contains a decision (if, else if, else).

Recently, Mbarki, Rahmouni and Erramdani [28] were conducted to model Web MVC2 generation using the ATL transformation language.

This paper aims to rethink the work presented by Mbarki and Erramdani [26, 27], by applying the standard MOF 2.0 QVT to develop the transformation rules aiming at generating the MVC2 target model. It is actually the only work for reaching this goal.

3 Model Driven Architecture (MDA)

In November 2000, OMG, a consortium of over 1 000 companies, initiated the MDA approach. The key principle of MDA is the use of models at different phases of application development. Specifically, MDA advocates the development of requirements models (CIM), analysis and design (PIM) and code (PSM).

The MDA architecture [29] 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.).

The major objective of MDA [16] is to develop sustainable models; those models are independent from the technical details of platforms implementation (Java EE, .Net, PHP or other), in order to enable the automatic generation of all codes and applications leading to a significant gain in productivity. MDA includes the definition of several standards, including UML [7], MOF [8] and XMI [9].

4 The MVC2 Pattern

The Model-View-Controller (MVC) architectural pattern is a widely used software and was created in 1980 by Xerox PARC for Smalltalk-80. Lately it has been recommended as a model for Java EE by Sun. The model also won strong popularity among PHP developers. The MVC pattern is a useful addition to developer tools, whatever the language used is.

The MVC pattern is a type of Design Patterns in the `Architectural Patterns` category. It is simple and very useful, and can essentially build an application using three levels: model, view and controller.

Figure 1 shows the architecture of the MVC2 pattern. The main feature of this pattern is to be composed of a single Servlet control. This pattern distinguishes the business logic, server-side processing and the display. Each component is reusable and replaceable.

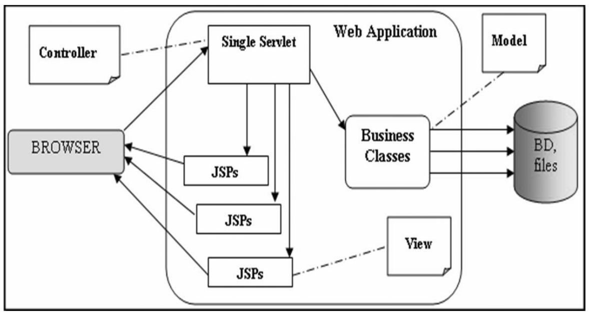


Figure 1: MVC2 Architecture

Based on this model many frameworks are designed to help developers build the presentation layer of their web applications. In the Java community, the Jakarta Struts project is one of the best examples.

4.1 The Struts framework

The Struts project [1] is managed within the community of Apache Software Foundation among the Jakarta projects. The motivation of this project is to provide the Java community with a framework based on the MVC2 architectural pattern while using Java EE technologies standard [19]: JSP / Servlet, JavaBeans, XML.

However, Struts is not the only framework for managing the presentation layer. Indeed, other frameworks have been designed for the same goal, but Struts is the most mature. The main advantage of Struts is the reduced complexity compared to other frameworks of the same degree of power [27], for instance, PureMVC, Gwittir and WebWork.

4.2 Architecture and functioning of Struts framework

The structure of the Struts framework derives from the MVC2 model (see Figure 2). In this model, there is a controller, views and access to the model.

Controller: The controller of the Struts framework is responsible for making the link between the view and model. It receives all client requests and forwards them to specific actions. These correspondences (mapping) are described in a configuration file called `struts-config.xml`.

View: The view is a set of JSP pages. To facilitate construction, the Struts framework provides several tag libraries.

Model: According to the MVC2 pattern, the model is independent from the controller. The Struts framework does not impose any; instead, technological choice is up to the developer (JDBC, EJB, JDO, and XML), etc according to his needs.

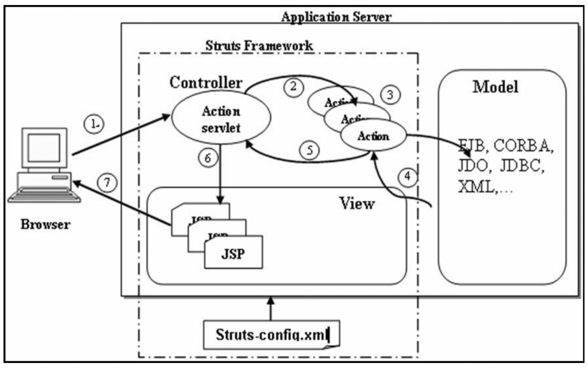


Figure 2: Principle of operation of the Struts framework

The interaction between the three components is managed by the main controller. In order to better to understand the working of the framework, we retail the life cycle of a HTTP request, schematized in figure 2:

1- The customer sends his HTTP request to the application. This request is taken in charge by the main controller, in the ActionServlet case;

2- The request is redirected towards the adequate controller;

3- The chosen controller handle the request. A dialogue with business logic is started when necessary;

4- The model provides the requested data;

5- The main controller is notified about the result of the treatment. In case of success, data are encapsulated in the JavaBeans (ActionForm) and then transmitted to the JSP selected by the controller;

6- The JSP constructs the answer according to the transmitted data;

7- The answer is sent to the browser.

5 The transformations of MDA models

MDA establishes the links of traceability between the CIM, PIM and PSM models through to the execution of the models' transformations.

The models' transformations recommended by MDA are essentially the CIM transformations to PIM and PIM transformations to PSM.

5.1 Approach by modeling

Currently, the models' transformations can be written according to three approaches: The approach by Programming, the approach by Template and the approach by Modeling.

The approach by Modeling is the one used in the present paper. It consists of applying concepts from model engineering to models' transformations themselves. The objective is modeling a transformation, to reach perennial and productive transformation models, and to express their independence towards the platforms of execution. Consequently, OMG elaborated a standard transformation language called MOF 2.0 QVT [11]. The advantage of the approach by modeling is the bidirectional execution of transformation rules. This aspect is useful for the synchronization, the consistency and the models reverse engineering [18].

Figure 3 illustrates the approach by modeling. Models transformation is defined as a model structured according to MOF 2.0 QVT meta-model. The MOF 2.0 QVT meta-model expresses some structural correspondence rules between the source and target meta-model of a transformation. This model is a perennial and productive model that is necessary to transform in order to execute the transformation on an execution platform.

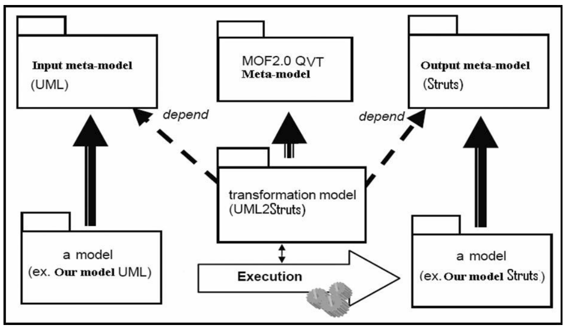


Figure 3: Approach by Modeling

5.2 MOF 2.0 QVT

Transformations models are at the heart of MDA, a standard known as MOF 2.0 QVT being established to model these changes. This standard defines the metamodel for the development of transformation model.

The QVT standard has a hybrid character (declarative / imperative) in the sense that it is composed of three different transformation languages (see Figure 4).

The declarative part of QVT is defined by `Relations` and `Core` languages, with different levels of abstraction. Relations are a user-oriented language for defining transformations in a high level of abstraction. It has a syntax text and graphics. Core language forms the basic infrastructure for the declaration part; this is a technical language of lower level determined by textual syntax. It is used to specify the semantics of Relations language in the form of a Relations2Core transformation. The declarative vision comes through a combination of patterns, source and target side to express the transformation.

The imperative QVT component is supported by Operational Mappings language. The vision requires an explicit imperative navigation as well as an explicit creation of target model elements. The Operational Mappings language extends the two declarative languages of QVT, adding imperative constructs (sequence, selection, repetition), etc and constructs in OCL edge effect.

The imperative style languages are better suited for complex transformations including a significant algorithm component. Compared to the declarative style, they have the advantage of optional case management in a transformation. For this reason, we chose to use an imperative style language in this paper.

Finally, QVT suggests a second extension mechanism for specifying transformations invoking the functionality of transformations implemented in an external language `Black Box`.

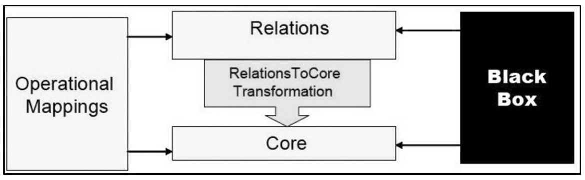


Figure 4: The QVT Structure

This work uses the QVT-Operational mappings language implemented by SmartQVT [23]. SmartQVT is the first open source implementation of the QVT-Operational language. The tool comes as an Eclipse plug-in under EPL license running on top of EMF framework. This tool is developed by France Telecom R & D project partially funded by the European IST Model Ware.
SmartQVT is composed of 3 components:

- **QVT Editor:** helps end users to write QVT specifications.
- **QVT Parser:** converts the QVT concrete textual syntax into its corresponding representation in terms of the QVT metamodel.
- **QVT Compiler:** produces, from a QVT model, a Java program on top of EMF generated APIs for executing the transformation. The input format is a QVT specification provided in XMI 2.0 in conformance with the QVT meta-model.

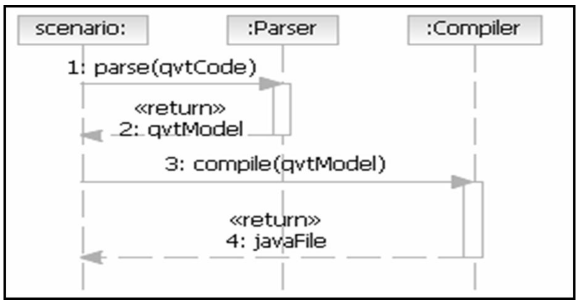


Figure 5: Transformation Scenario with SmartQVT tool

In Figure 5, presents a scenario of minimal processing:

- The parser is called and gets as input a text file containing a QVT code (qvtCode).
- The parser returns the model conforming to the QVT metamodel.
- Then the returned model is passed to the compiler.
- Finally, we get a Java file implementing the transformation (javaFile).

5.3 OCL (Object Constraint Language)

Object Constraint Language (OCL) is a formal language used to describe expressions on UML models. These expressions typically specify invariant conditions that must hold for the system being modeled or queries over objects described in a model. Note that when the OCL expressions are evaluated, they do not have side effects. OCL expressions can be used to specify operations / actions that, when executed, do alter the state of the system. UML modelers can use OCL to specify application-specific constraints in their models.

Currently, several tools of OCL exist, including ATL [14] Dresden OCL Toolkit [21], Eclipse MDT OCL [10] KMF [12], Ocle [17], etc.

In MOF 2.0 QVT, OCL is extended to Imperative OCL as part of `QVT Operational Mappings`. Imperative OCL added services to manipulate the system states (for example, to create and edit objects, links and variables) and some constructions of imperative programming languages (for example, loops and conditional execution). It is used in QVT Operational Mappings to specify the transformations.

QVT defines two ways of expressing model transformations: declarative and operational approaches.
The declarative approach is the `Relations` language where transformations between models are specified as a set of relationships that must hold for successful transformation.

The operational approach allows either defining transformations using a complete imperative approach or complementing the relational transformations with imperative operations, by implementing relationships. Imperative OCL adds imperative elements of OCL, which are commonly found in programming languages like Java. Its semantics are defined in [11] by a model of abstract syntax. The complete abstract syntax ImperativeOCL is shown in Figure 6.

The most important aspect of the abstract syntax is that all expression classes must inherit OclExpression. OclExpression is the base class for all the conventional expressions of OCL. Therefore, Imperative Expressions can be used wherever there is OclExpressions.

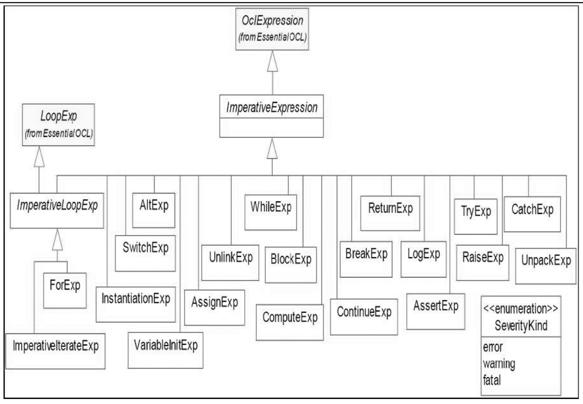


Figure 6: Imperative Expressions of ImperativeOCL

6 The UML and MVC2 meta-models

To develop the algorithm of transformation between the source and target model, we present in this section, the different meta-classes forming the UML source meta-model and the MVC2 target meta-model. The meta-model source structure simplified UML model based on a package containing the data types and classes. These classes contain properties typed and characterized by multiplicities (upper and lower). The classes contain operations with typed parameters. Figure 7 shows the source meta-model:

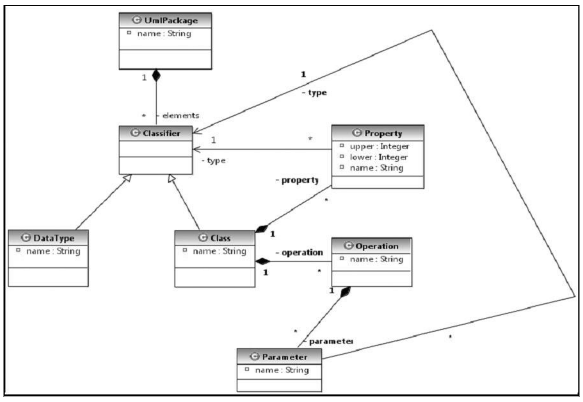


Figure 7: Simplified UML Meta-model

Figure 8 illustrates the first part of the target meta-model. This meta-model is a simplified diagram of relational databases. It consists of several tables, themselves composed of typed columns.

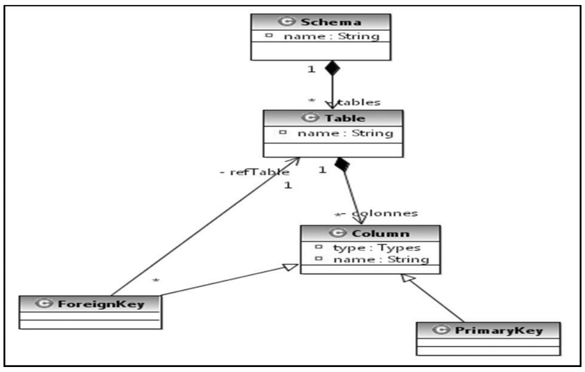


Figure 8: Simplified meta-model of a relational database

Figure 9 illustrates the second part of the target meta-model. This is the business model of the application to be processed. In our case, we opted for components such as Beans. We recall that Struts does not provide specific classes.

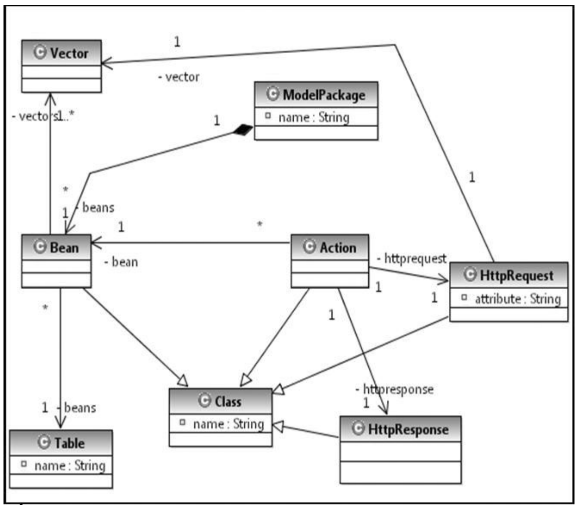


Figure 9: Simplified meta-model of a modelPackage

Figure 10 illustrates the third part of the target meta-model. This meta-model illustrates the models that represent the display of the application. In this model, the servlet calls the `execute()` method on the instance of the class action. It performs its processing and then calls the `mapping.findForward()` method with a return to the JSP page specified.

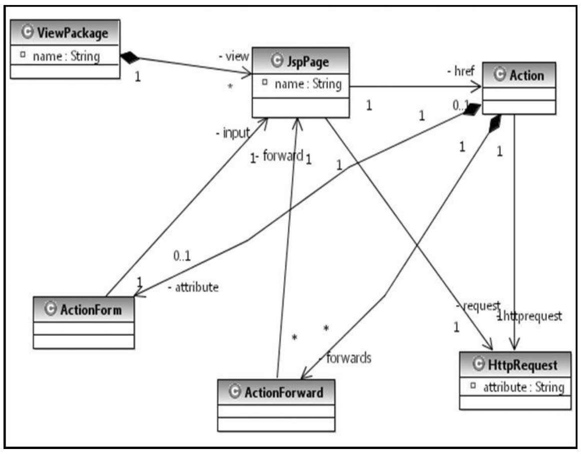


Figure 10: Simplified meta-model of a viewPackage

Figure 11 shows the fourth part of the target meta-model. This meta-model is the package controller. This meta-model illustrates models that represent the controller application. The controller is responsible for receiving applications sent by the client, with the invocation of the class action. It, thus, interacts with the business model and coordinates with the display by sending it to the client.

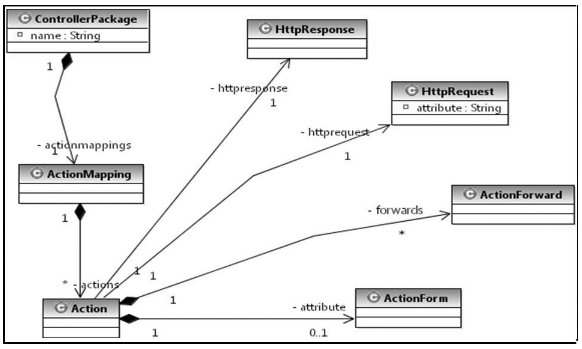


Figure 11: Simplified meta-model of a controllerPackage

The works of Mbarki and Erramdani [26, 27] contain more details related to this section topic.

7 The process of transforming UML source model to MVC2 target model (Struts)

CRUD operations (Create, Read, Update, and Delete) are most commonly implemented in all systems. That is why we have taken into account in our transformation rules these types of transactions. In [26], it was implemented that read operation, however, our work aims to implement all CRUD operations.

We first developed ECORE models corresponding to our source and target meta-models, and then we implemented the algorithm (see sub-section 7.1) using the transformation language QVT Operational Mappings. To validate our transformation rules, we conducted several tests. For example, we considered the class diagram (see Figure 12). After applying the transformation on the UML model, composed by the classes Department, Employee and City (ville), we generated the target model (see Figure 16).

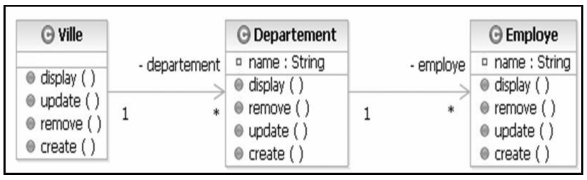


Figure 12: UML instance model

7.1 The transformation rules

By source model, we mean model containing the various classes of our business model. The elements of this model are primarily classes.

Main algorithm:

```
input umlModel:UmlPackage
output strutsModel
:StrutsProjectPackage
begin
create StrutsProjectPackage struts
create ViewPackage vp
vp = transformationRuleOne(e)
create ControllerPackage cp
cp = transformationRuleTwo(e)
link vp to struts
link cp to struts
return struts
end
```

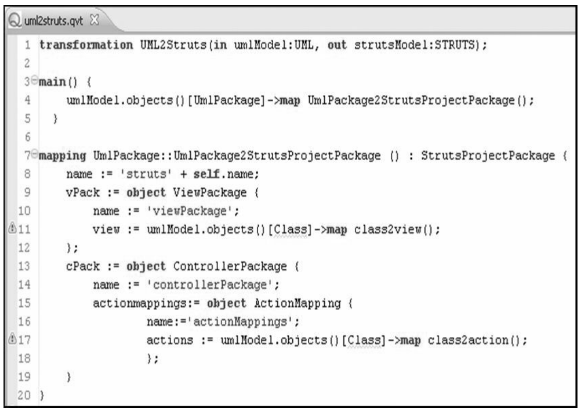
function

```
transformationRuleOne(e:Class)
:ViewPackage
begin
```

```
create ViewPackage vp
for each e ?  source model
if e.methods.name ?  ’remove’
create JspPage page
link page to vp
end if
end for
return vp
end
```

function
```
transformationRuleTwo(e:Class) :
ControllerPackage
begin
create ControllerPackage cp
create ActionMapping am
for each page viewPackage
link page to actionForward
create actionForm
create Action action
create ActionForward actionForward
actionForm.input=page
actionForm.attribute=action
link page to actionForward
link actionForward to action
put action in am
end for
link am to cp
return cp
end
```

Figure 13 illustrates the first part of the transformation code of UML source model to the MVC2 target.

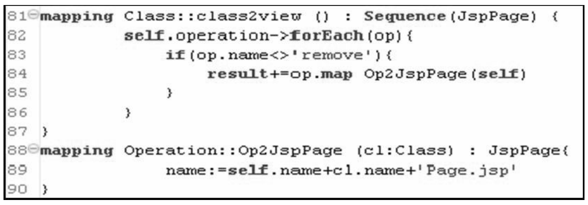


```
transformation UML2Struts(in umlModel:UML, out strutsModel:STRUTS);

main() {
    umlModel.objects()[UmlPackage]->map UmlPackage2StrutsProjectPackage();
  }

mapping UmlPackage::UmlPackage2StrutsProjectPackage () : StrutsProjectPackage {
    name := 'struts' + self.name;
    vPack := object ViewPackage {
        name := 'viewPackage';
        view := umlModel.objects()[Class]->map class2view();
    };
    cPack := object ControllerPackage {
        name := 'controllerPackage';
        actionmappings:= object ActionMapping {
                name:='actionMappings';
                actions := umlModel.objects()[Class]->map class2action();
                };
    }
}
```

Figure 13: The transformation code UML2Strut

The transformation uses as input a UML type model, named umlModel, and as output a STRUTS type model named strutsModel. The entry point of the transformation is the `main` method. This method makes the correspondence between all elements of type UmlPackage of the input model and the elements of type StrutsProjectPackage output model.

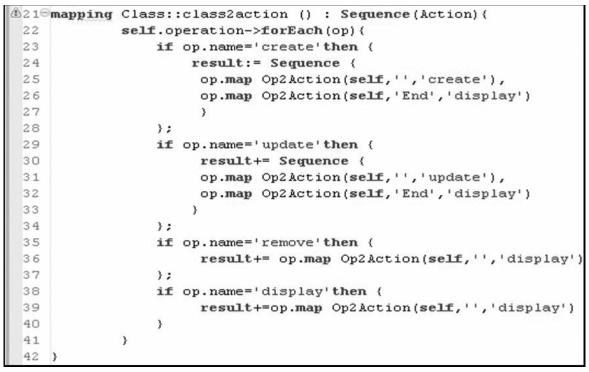
The objective of the second part of this code is to transform a UML package to Struts package, creating an item such `View` package and `Controller` package. It is to turn each class in UML package, into JSP in the View package, and into Action in the Controller package making sure to give names to different packages.



```
mapping Class::class2view () : Sequence(JspPage) {
        self.operation->forEach(op){
            if(op.name<>'remove'){
                result+=op.map Op2JspPage(self)
            }
        }
}
mapping Operation::Op2JspPage (cl:Class) : JspPage{
            name:=self.name+cl.name+'Page.jsp'
}
```

Figure 14: The mapping class2view and Operation2JspPage

The methods presented in Figure 14 means that each operation in a class corresponds to JSP page.



```
mapping Class::class2action () : Sequence(Action){
        self.operation->forEach(op){
            if op.name='create'then {
                result:= Sequence {
                 op.map Op2Action(self,'','create'),
                 op.map Op2Action(self,'End','display')
                 }
            };
            if op.name='update'then {
                 result+= Sequence {
                 op.map Op2Action(self,'','update'),
                 op.map Op2Action(self,'End','display')
                }
            };
            if op.name='remove'then {
                 result+= op.map Op2Action(self,'','display')
            };
            if op.name='display'then {
                 result+=op.map Op2Action(self,'','display')
            }
        }
}
```

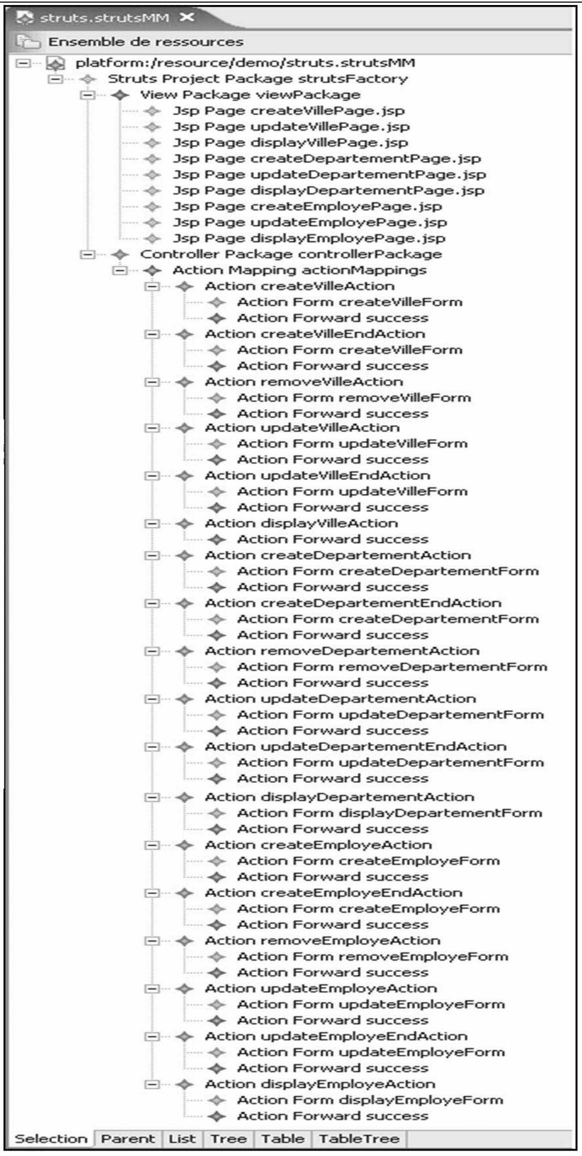
Figure 15: The mapping class2action

The method presented in Figure 15 means that each class corresponds to one or more actions as the name and type of operations which contains it.

The codes and models are publicly available online `http://sites.google.com/site/uml2mvc/`.

7.2 Result

Figure 16 shows the result after applying the transformation rules.



```
struts.strutsMM
Ensemble de ressources
platform:/resource/demo/struts.strutsMM
  Struts Project Package strutsFactory
    View Package viewPackage
      Jsp Page createVillePage.jsp
      Jsp Page updateVillePage.jsp
      Jsp Page displayVillePage.jsp
      Jsp Page createDepartementPage.jsp
      Jsp Page updateDepartementPage.jsp
      Jsp Page displayDepartementPage.jsp
      Jsp Page createEmployePage.jsp
      Jsp Page updateEmployePage.jsp
      Jsp Page displayEmployePage.jsp
    Controller Package controllerPackage
      Action Mapping actionMappings
        Action createVilleAction
          Action Form createVilleForm
          Action Forward success
        Action createVilleEndAction
          Action Form createVilleForm
          Action Forward success
        Action removeVilleAction
          Action Form removeVilleForm
          Action Forward success
        Action updateVilleAction
          Action Form updateVilleForm
          Action Forward success
        Action updateVilleEndAction
          Action Form updateVilleForm
          Action Forward success
        Action displayVilleAction
          Action Forward success
        Action createDepartementAction
          Action Form createDepartementForm
          Action Forward success
        Action createDepartementEndAction
          Action Form createDepartementForm
          Action Forward success
        Action removeDepartementAction
          Action Form removeDepartementForm
          Action Forward success
        Action updateDepartementAction
          Action Form updateDepartementForm
          Action Forward success
        Action updateDepartementEndAction
          Action Form updateDepartementForm
          Action Forward success
        Action displayDepartementAction
          Action Form displayDepartementForm
          Action Forward success
        Action createEmployeAction
          Action Form createEmployeForm
          Action Forward success
        Action createEmployeEndAction
          Action Form createEmployeForm
          Action Forward success
        Action removeEmployeAction
          Action Form removeEmployeForm
          Action Forward success
        Action updateEmployeAction
          Action Form updateEmployeForm
          Action Forward success
        Action updateEmployeEndAction
          Action Form updateEmployeForm
          Action Forward success
        Action displayEmployeAction
          Action Form displayEmployeForm
          Action Forward success
Selection | Parent | List | Tree | Table | TableTree
```

Figure 16: Generated PSM MVC2 Web model

The first element in the generated PSM model is: viewPackage that contains the nine JSPs, namely DisplayVillePage.jsp, DisplayDepartementPage.jsp, DisplayEmployePage.jsp, CreateVillePage.jsp, CreateDepartementPage.jsp, CreateEmployePage.jsp, UpdateVillePage.jsp, UpdateDepartementPage.jsp and UpdateEmployePage.jsp. Since the operation of the removal requires any form, we'll go to the controllerPackage element, which contains a single element `ActionMapping`. The latter contains eighteen actions whose names are respectively `DisplayXAction, CreateXAction, UpdateXAction, RemoveXAction, CreateXEndAction, UpdateXEndAction`, where X should be replaced by City(Ville) by Department, and Employee. Operations for creation and update, add forms to enter new values. For this reason, we add `CreateXEndAction` and `UpdateXEndAction`.

For each element, for example, `DisplayDepartementAction` contains two elements: the `attribute` element indicating the form entered in this action is the ActionForm `DisplayDepartementForm`, and `Forwards` element with `forward` attribute DisplayDepartementPage.jsp. The Action element `DisplayVilleAction` contains only one `Forwards` element with `forward` attribute DisplayVillePage.jsp.

The remaining actions follow the same principle.

8 Conclusion and perspectives

In this paper, we applied the MDA to generate the MVC2 code web application based on UML class diagram. The purpose of our contribution is to rethink the works presented by Mbarki and Erramdani [26, 27]. However, the transformation rules were developed applying the approach by modeling and MOF 2.0 QVT, as transformation language, to browse the class diagram and generate, through these rules, an XML file containing all the actions, forms and JSP pages. This file can be used to produce the necessary code to the target application. The transformation algorithm handles all CRUD operations. The advantage of this approach is the bidirectional execution of transformation rules.

Moreover, this work can be complemented by advanced features of Web applications. For example, we can provide some user interface as well as the ability to incorporate other features: the persistence of objects in relational database (Hibernate) and dependency injection (Spring) to produce a complete web application according to the n-tier architecture. This is the subject of a work in finalization phase.

References

[1] Apache software foundation: The apache struts web application software framework. http://struts.apache.org.

[2] Asp.net mvc site http://www.asp.net/mvc/.

[3] Gwittir site http://code.google.com/p/gwittir/.

[4] Puremvc framework http://puremvc.org/.

[5] Spring framework http://www.springsource.org/.

[6] Zend framework http://framework.zend.com/.

[7] *UML Infrastructure Final Adopted Specification, version 2.0*, September 2003. http://www.omg.org/cgi-bin/doc?ptc/03-09-15.pdf.

[8] *Meta Object Facility (MOF), version 2.0*, January 2006. http://www.omg.org/spec/MOF/2.0/PDF/.

[9] *XML Metadata Interchange (XMI), version 2.1.1*, December 2007. http://www.omg.org/spec/XMI/.

[10] *MDT-OCL-Team, MDT OCL*, 2008. http://www.eclipse.org/modeling/mdt/?project=ocl.

[11] *Meta Object Facility 2.0 Query View Transformation (MOF 2.0 QVT), Version 1.1*, December 2009. http://www.omg.org/spec/QVT/1.1/Beta2/PDF/.

[12] Akehurst, D. and Patrascoiu, O. The kent modeling framework (kmf). University of Kent, 2005. http://www.cs.kent.ac.uk/projects/ocl.

[13] Ali, A. B. H., Abdellatif, A., and Ahmed, S. B. Transformation des contraintes d'intégrité - des modèles conceptuels vers le relationnel. In *INFORSID*, pages 398–415, 2007.

[14] Allilaire, F., Bézivin, J., Jouault, F., and Kurtev, I. Atl - eclipse support for model transformation. In *In Proceedings of the Eclipse Technology eXchange workshop (eTX) at the ECOOP 2006 Conference*, 2005.

[15] Allilaire, F., Bézivin, J., Jouault, F., and Kurtev., I. Atl: A model transformation tool. *Science of Computer Programming-Elsevier*, 72:31–39, 2008.

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

[17] Chiorean, D. and OCLE-Team. Object constraint language environment 2.0., 2008. http://lci.cs.ubbcluj.ro/ocle/.

[18] Czarnecki, K. and Helsen, S. Classification of model transformation approaches. In *In online proceedings of the 2nd OOPSLA'03 Workshop on Generative Techniques in the Context of MDA*, October 2003.

[19] Davis, M. *Struts, an open-source MVC implementation : Manage complexity in large Web sites with this servlets and JSP framework*. IBM, Feb 2001. http://www.ibm.com/developerworks/library/j-struts/.

[20] Distante, D., Rossi, G., and Canfora, G. Modeling business processes in web applications: An analysis framework. In *In Proceedings of the The 22nd Annual ACM Symposium on Applied Computing*, page 1677.

[21] Dresden-OCL-Team. *Dresden OCL Toolkit*, 2008. http://dresden-ocl.sourceforge.net.

[22] Escalona, M. J. and Koch, N. Metamodeling the requirements of web systems. *Lecture Notes in Business Information Processing*, 1:267–282, August 2007.

[23] France Telecom. *SmartQVT documentation*, 2007. http://smartqvt.elibel.tm.fr/doc/index.html.

[24] Gharavi, V., Mesbah, A., and van Deursen, A. Modelling and generating ajax applications: A model-driven approach. In *Proceeding of the 7th International Workshop on Web-Oriented Software Technologies*, page 38, New York, USA, 2008.

[25] Kraus, A., Knapp, A., and Koch, N. Model-driven generation of web applications in uwe. In *Proceeding of the 3rd International Workshop on Model-Driven Web Engineering*, CEUR-WS, 2007.

[26] Mbarki, S. and Erramdani, M. Toward automatic generation of mvc2 web applications. *InfoComp - Journal of Computer Science*, 7(4):84–91, December 2008.

[27] Mbarki, S. and Erramdani, M. Model-driven transformations: From analysis to mvc 2 web model. *International Review on Computers and Software (I.RE.CO.S.)*, 4(5):612–620, September 2009.

[28] Mbarki, S., Rahmouni, M., and Erramdani, M. Transformation atl pour la génération de modèles web mvc 2. In *10e Colloque Africain sur la Recherche en Informatique et en Mathématiques Appliquées (CARI)*, 2010.

[29] Miller, J. and Mukerji, J. *MDA Guide Version 1.0.1*, 2003. http://www.omg.org/docs/omg/03-06-01.pdf.

[30] Nasir, M., Hamid, S., and Hassan, H. Webml and .net architecture for developing students appointment management system. *Journal of applied science*, 9(8):1432–1440, 2009.

[31] Nora, K. Transformations techniques in the model-driven development process of uwe. In *Proceeding of the 2nd International Workshop Model-Driven Web Engineering*, page 3, Palo Alto, 2006.

[32] Oberortner, E., Vasko, M., and Dustdar, S. Towards modeling role-based pageflow definitions within web applications. In *Proceeding of the 4th Model Driven Web Engineering Workshop*, 2008.