International Journal of Applied Engineering Research ISSN 0973-4562 Volume 12, Number 14 (2017) pp. 4372-4381

Applying MDA approach for Spring MVC Framework

Aziz Srai[1], Fatima Guerouate[2], Naoual Berbiche[3] and Hilal Drissi Lahsini[4]

System Analysis, Information Treatment and Integrated Management Laboratory
Superior School of Technologies of Sallee, Mohammadia School of engineering, Mohamed V University in Rabat, Morocco
Crown Prince Street –B.P 227 Sallee – Morocco

[1]Orcid: 0000-0002-7942-4206

Abstract

Separation of business logic from any technical platform is an important principle to cope with complexity, and to achieve the required engineering quality factors such as adaptability, maintainability, and reusability. In this context, Model Driven Architecture (MDA) is a framework defined by the OMG for designing high quality software systems. In this paper we are going to present a model-driven approach to the development of the MVC2 web applications especially Spring MVC based on the uml class diagramme. The transformation language is ATL (Atlas Transformation Language). The transformation rules defined in this paper can generate from, the class diagramme, an XML file respecting the architecture MVC2 (Model-View-Controller), this file can be used to generate the end-to-end necessary Spring MVC code of a web application.

Keywords: model transformation; MDA approach; meta-model; ATL.

INTRODUCTION

Nowadays, to deal with the complexity of information systems and the high costs of technological migration, many organizations adopt MDA (Model driven architecture) as an approach to design and implement enterprise applications. Indeed, MDA [1][5] play a major role to increase portability because it focuses on modeling the system using platform independent models which can be easily transformed, as needed, into other chosen platforms.

On the other hand, several authors have developed many algorithms to generate code for web applications, among these works , a recent work has attracted our attention, the work of Mbarki and Erramdani [7, 8], the work was conducted to model Web MVC2 (struts2) generation using the ATL [6] transformation language. In this context, this paper aims to rethink and to complete the work by applying the MDA approach to develop the transformation rules aiming at generating the MVC2 web model (Spring MVC) according to our target model. The result of this transformation is an XMI file. Actually, it is the only work for reaching this goal.

We have developed also two transformations type (Model to Text) to generate Java code for Spring MVC [2][3] applications. To validate our approach a case study is presented.

This paper is organized as follows: we begin in the first section with an introduction. Section 2 is dedicated to the related work. Section 3 introduces MDA architecture, UML and Spring MVC meta-models, the transformation rules using ATL language from UML source model to the Spring MVC target model. The last section concludes this paper and presents some perspectives.

RELATED WORK

Many researches on MDA and generation of code have been conducted in recent years. The most relevant are [1][5][7][8][9][10][12][13][14][15][16][17][18][20] [21].

The authors of the work [7, 8], have developed two meta-models, 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.

Bezivin et al. [9] propose to transform PIMs defined by Enterprise Distributed Object Computing into PSMs for different web services platforms, through the ATL language (Atlas transformation language).

Another work follows the same logic and have been the subject of an article [11]. A meta-model of AJAX has been defined using the AndroMDA tool. The generation of AJAX code was made and illustrated by an application that manages CRUD operations of person.

The work [12] has proposed a model-driven development approach for E-Learning platform.

CONTRIBUTION OF OUR RESEARCH

A. Model Driven Architecture (MDA)

Model Driven Architecture (MDA) In November 2000, OMG, a consortium of over 1000 companies, initiated the MDA approach, the aim of MDA (Model Driven Architecture) is to develop complex applications in simpler ways, the system functionality is separated from the implementation details.

The principal key of MDA is the use of models at different phases of application development, those models are independent from the technical details of platforms implementation.

MDA development process consists of models transformation, starting with a computation independent model (CIM), which is subsequently transformed to a platform independent model (PIM), platform specific model (PSM) and finally to an executable code. It is also possible to transform models at the same abstraction level. Figure 3 shows the following set of transformations: CIM-to-CIM, CIM-to-PIM, PIM-to-PIM, PIM-to-PSM, PSM-to-PSM, and PSM-to-code.

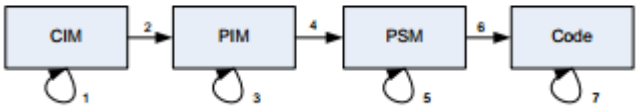


Figure 1. Simplified View of the MDA Process

B. The Spring MVC framework

The Spring MVC framework provides model-view-controller architecture and ready components that can be used to develop web applications. These components are servlets, java server pages, java beans, business logic. For each Spring MVC application we need to configure the two files, spring-beans.xml and web.xml, the components of Spring MVC are :

- **Business model:** it is a model corresponds to the application business logic.
- **Controller:** in Spring MVC the principal controller called DispatcherServlet, which is implemented as a component servlet via the org.springframework.web.servlet.DispatcherServlet, performs the following tasks when receiving a request:

  - Matches URI of the request to the appropriate controller class.

  - The Controller takes the request and calls the appropriate service methods based on used GET or POST method. The service method will set model data based on defined business logic and returns view name to the DispatcherServlet.

  - finally, the DispatcherServlet will take help from ViewResolver to catch the defined view for the request.

- **View:** the view component corresponds to the tier presentation of a Spring MVC application.

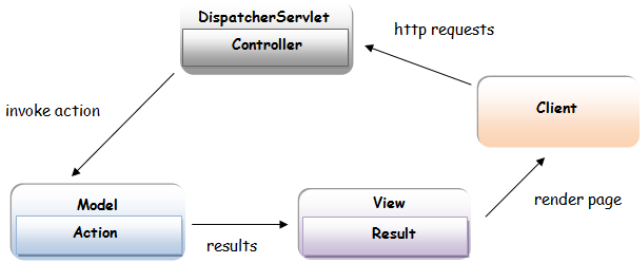


Figure 2. Spring MVC architecture

C. UML and Spring MVC meta-models

To develop the transformation algorithm between source and target model, we present in this section, the various meta-classes forming the meta-model UML source and the target meta-model Spring MVC.

a. Meta-model UML source

The source meta-model structures a simplified UML model based on packages containing classes and associations, classes contain typed properties and they are characterized by multiplicities (upper and lower). The classes are composed of operations with typed parameters. The following figure illustrates the source meta-model.

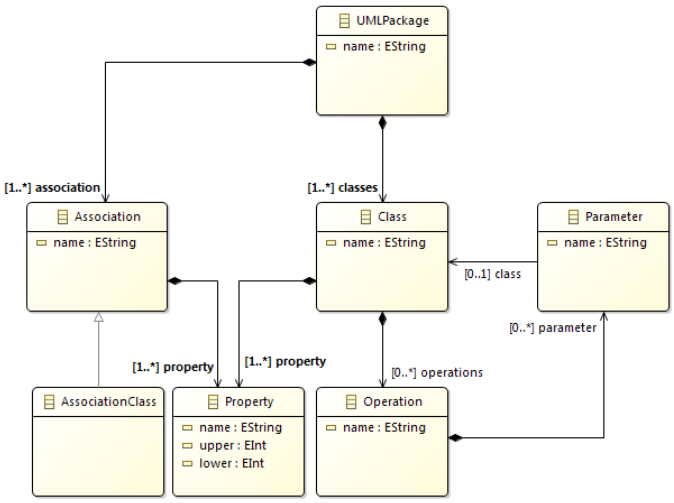


Figure 3. Simplified UML meta-model

- ***Package*:** represents the concept of UML package. This meta-class is connected to the meta-class ***Class***.
- ***Class*:** represents the concept of UML class.
- ***Operation***: is used to express the concept of operations of a UML class.
- **Parameter:** expresses the concept of parameters of an operation.
- ***Property*:** expresses the concept of properties of a UML class The association between ***Class*** meta-class and Property meta-class expresses the fact that a class is composed of properties.

b. Meta-model Spring MVC target

Our target meta-model is composed of two essential part, the first part represents the view package, this package contains many jsp pages. the second part represents the control package, this package contains many controllers and formbeans, each controller contains one or more actions , and each formbean contains one or more actionform. The following figure illustrates the target meta-model.

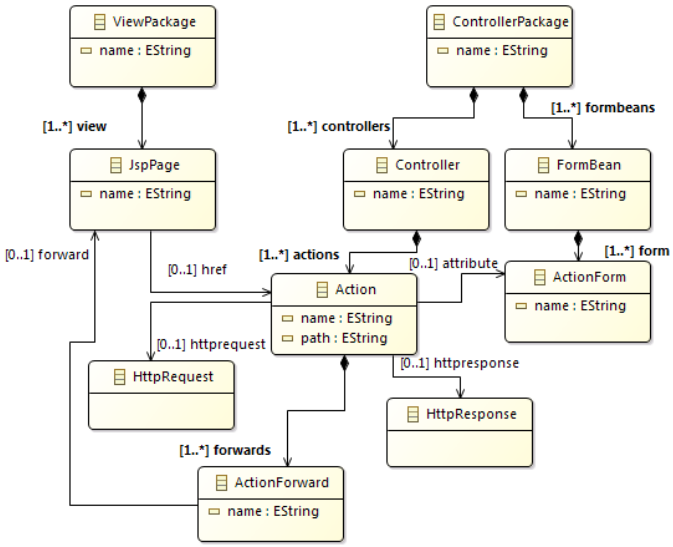


Figure 4. Spring MVC metamodel

Controller: represents the concept of controller classes.

Action: represents the concept of action.

ActionForm: represents the concept of ActionForm classes. An ActionForm represents a form containing the parameters of the request from the view.

JspPage: represents the presentation layer of applicaton.

HttpRequest: is the concept of HttpServletRequest Classes.

HttpResponse : represents the concept of HttpServletResponse classes.

D. Transformation process from UML to Spring MVC framework implementation

Operations like (add, edit, delete, and search) are most commonly implemented in all systems. That is why we have taken into account in our transformation rules these types of transactions.

We first developed ECORE models corresponding to our source and target meta-models, and then we implemented transformation algorithm (see Figure 7) using the transformation language ATL. Atlas Transformation Language, is a model transformation language developed by the team of Jean Bézivin at LINA in Nantes. It is part of the Eclipse M2M (Model-to-Model).

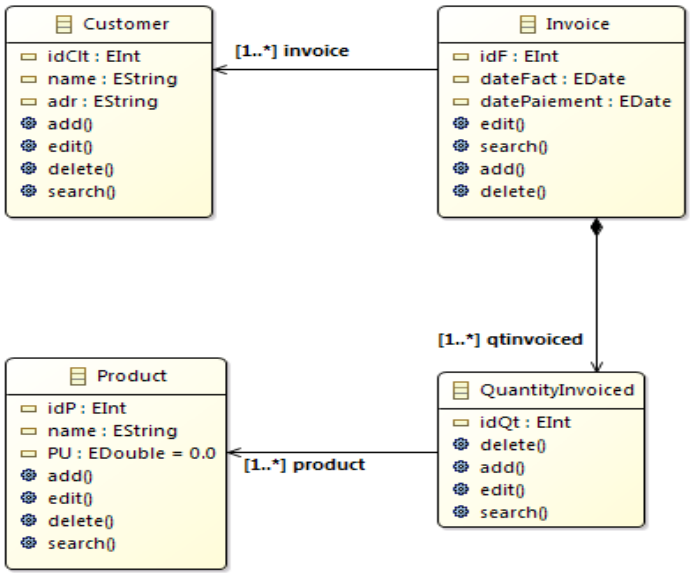


Figure 5. Instance of UML model

To validate our transformation algorithm, we have conducted several tests. For example, we considered the class diagram (see Figure 5). After applying the transformation on the UML model (see Figure 6), composed by the classes Customer, Invoice, Quantity Invoiced and Product, we generate a target model presented by the XML file below:

```
<ViewPackage name="folder">
  <JspPage name="addCustomer.jsp"/>
  <JspPage name="editCustomer.jsp"/>
```

```
  <JspPage name="deleteCustomer.jsp"/>
  <JspPage name="searchCustomer.jsp"/>
  <JspPage name="editInvoice.jsp"/>
  <JspPage name="searchCustomer.jsp"/>
  <JspPage name="addInvoice.jsp"/>
  <JspPage name="deleteInvoice.jsp"/>
  <JspPage name="addProduct.jsp"/>
  <JspPage name="editProduct.jsp"/>
  <JspPage name="deleteProduct.jsp"/>
  <JspPage name="searchProduct.jsp"/>
  <JspPage name="deleteQuantityInvoiced.jsp"/>
  <JspPage name="addQuantityInvoiced.jsp"/>
  <JspPage name="editQuantityInvoiced.jsp"/>
  <JspPage name="searchQuantityInvoiced.jsp"/>
</ViewPackage>

<Controller name="controller">
  <Action name="addCustomerAction" path="/addCustomerAction">
    <forwards name="addCustomer" path="/add"/>
  </Action>
  <Action name="editCustomerAction" path="/editCustomerAction">
    <forwards name="editCustomer" path="/edit"/>
  </Action>
  <Action name="deleteCustomerAction"
path="/deleteCustomerAction">
    <forwards name="deleteCustomer" path="/delete"/>
  </Action>
  <Action name="searchCustomerAction"
path="/searchCustomerAction">
    <forwards name="searchCustomer" path="/search"/>
  </Action>
  <Action name="editInvoiceAction" path="/editInvoiceAction">
    <forwards name="editInvoice" path="/edit"/>
  </Action>
  <Action name="searchCustomerAction"
path="/searchCustomerAction">
    <forwards name="searchCustomer" path="/search"/>
  </Action>
  <Action name="editCustomerAction" path="/editCustomerAction">
    <forwards name="editCustomer" path="/edit"/>
  </Action>
  <Action name="deleteInvoiceAction" path="/deleteInvoiceAction">
    <forwards name="deleteInvoice" path="/delete"/>
  </Action>
  <Action name="addProductAction" path="/addProductAction">
    <forwards name="addProduct" path="/add"/>
  </Action>
  <Action name="editProductAction" path="/editProductAction">
    <forwards name="editProduct" path="/edit"/>
  </Action>
  <Action name="deleteProductAction"
path="/deleteProductAction">
    <forwards name="deleteProduct" path="/delete"/>
  </Action>
  <Action name="searchProductAction"
path="/searchProductAction">
    <forwards name="searchProduct" path="/search"/>
  </Action>
  <Action name="deleteQuantityInvoicedAction"
path="/deleteQuantityInvoicedAction">
    <forwards name="deleteQuantityInvoiced" path="/delete"/>
  </Action>
  <Action name="addQuantityInvoicedAction"
path="/addQuantityInvoicedAction">
    <forwards name="addQuantityInvoiced" path="/add"/>
  </Action>
  <Action name="editQuantityInvoicedAction"
path="/editQuantityInvoicedAction">
    <forwards name="editQuantityInvoiced" path="/edit"/>
  </Action>
  <Action name="searchQuantityInvoicedAction"
path="/searchQuantityInvoicedAction">
    <forwards name="searchQuantityInvoiced" path="/search"/>
  </Action>
<Controller/>

<FormBean name="formbeans">
  <ActionForm name="addCustomerForm"/>
  <ActionForm name="editCustomerForm"/>
  <ActionForm name="deleteCustomerForm"/>
  <ActionForm name="searchCustomerForm"/>
  <ActionForm name="editInvoiceForm"/>
  <ActionForm name="searchCustomerForm"/>
```

```
<ActionForm name="editCustomerForm"/>
<ActionForm name="deleteInvoiceForm"/>
<ActionForm name="addProductForm"/>
<ActionForm name="editProductForm"/>
<ActionForm name="deleteProductForm"/>
<ActionForm name="searchProductForm"/>
<ActionForm name="deleteQuantityInvoicedForm"/>
<ActionForm name="addQuantityInvoicedForm"/>
<ActionForm name="editQuantityInvoicedForm"/>
<ActionForm name="searchQuantityInvoicedForm"/>
</FormBean>
```

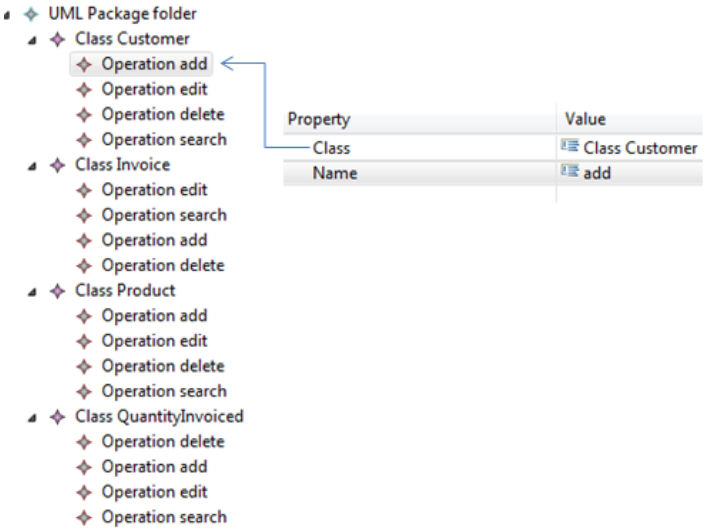


Figure 6. Uml source model

In the following transformation algorithm we have added the method isExist(), an Ocl expression, to verify that the class contains a method, before the execution of the algorithm.

Algorithm :

```
module Uml2SpringMVC;
create OUT : MM1 from IN : MM;

rule P2View {
        from a : MM!UMLPackage
        to  v : MM1!ViewPackage (
        name <-  a.name,
        view <- a.classes ->
                        collect(e |
thisModule.resolveTemp(e,'jsp'))

                        )
}
rule O2Package {
        from o : MM!Operation(o.isExist())
        to   jsp : MM1!JspPage (
        name <- if o.name='search'
        then  o.name+o.class.name+'.jsp'
        else if o.name='delete'
        then  o.name+o.class.name+'.jsp'
        else if o.name='edit'
        then  o.name+o.class.name+'.jsp'
        else if o.name='add'
        then  o.name+o.class.name+'.jsp'
        else OclUndefined
    endif
    endif
    endif
        endif
                        )
}
rule P2Controller {
        from u : MM!UMLPackage
        to  a : MM1!Controller (
        name <-  'controller',
        actions <- u.classes ->
                        collect(e |
thisModule.resolveTemp(e,'act'))

                        )
}
rule O2Action {
        from o : MM!Operation(o.isExist())
        to   act : MM1!Action (
        name <- o.name+o.class.name+'Action',
        path <- '/'+o.name+o.class.name+'Action',
        forwards <- Sequence{forwarding}
        ),
        forwarding : MM1!ActionForward (
```

```
        name <- o.name+o.class.name,
        path <- '/'+o.name

                )

}
rule Pack2FormBean {
        from u : MM!UMLPackage
        to  a : MM1!FormBean(
        name <-  'formbeans',
        form <- u.classes ->
                        collect(e |
thisModule.resolveTemp(e,'f'))

                )
}
rule O2ActionForm {
        from o : MM!Operation(o.isExist())
        to   f : MM1!ActionForm (
        name <- if o.name='search'
        then  o.name+o.class.name+'Form'
        else if o.name='delete'
        then  o.name+o.class.name+'Form'
        else if o.name='edit'
        then  o.name+o.class.name+'Form'
        else if o.name='add'
        then  o.name+o.class.name+'Form'
        else OclUndefined
    endif
    endif
    endif
        endif
                )

}
helper context MM!Operation def: isExist() : Boolean =
        if not self.class.oclIsUndefined() then
                true
        else
                        false
        endif;
```

Figure 7. Algorithm to generate spring MVC model

E. Spring MVC source code

In this section, we have developed a transformation M2T (Model to Text) in order to generate the Spring MVC code.This will avoid the programmer to worry about understanding the application of algorithm. For the Model To Text transformation, we defined template for the Spring MVC code using the OMG standard Acceleo. The execution of these templates gives the source code of the application for the MVC three layers ; the views, the controller and the model.

the template below represents the code to generate forms :

```
[comment encoding = UTF-8 /]
[module generate('http://Spring/1.0')]
[template public Operation2JspPage(c : Class)]
[comment @main/]
[file (c.name.concat('.txt'), false, 'UTF-8')]
  [for (o : Operation | c.operations)]
  <div>
    <f:form modelAttribute="[o.name+'Form'/]" method="post"
action="[o.name+'Action'/]">
      <table>
        <tr>
         <td>...</td>
         <td>...</td>
         ...
        </tr>
      </table>
    </f:form>
  </div>
  [/for]
 [/file]
[/template]
```

The result of this template is the following forms:

```
<div>
    <f:form modelAttribute="addForm" method="post"
action="addAction">
```

```
<table>
  <tr>
   <td>...</td>
   <td>...</td>
   ...
  </tr>
 </table>
</f:form>
</div>
<div>
 <f:form modelAttribute="editForm" method="post" action="editAction">
  <table>
   <tr>
    <td>...</td>
    <td>...</td>
    ...
   </tr>
  </table>
 </f:form>
</div>
<div>
 <f:form modelAttribute="deleteForm" method="post" action="deleteAction">
  <table>
   <tr>
    <td>...</td>
    <td>...</td>
    ...
   </tr>
  </table>
 </f:form>
</div>
<div>
 <f:form modelAttribute="searchForm" method="post" action="searchAction">
  <table>
   <tr>
    <td>...</td>
    <td>...</td>
    ...
   </tr>
  </table>
 </f:form>
</div>
```

To develop the source code of the controller we generate the following template :

```
[comment encoding = UTF-8 /]
[module generate('http://Spring/1.0')]
[template public Operation2Action(c : Class)]
[comment @main/]
[file (c.name.concat('.txt'), false, 'UTF-8')]
@Controller
public class [c.name/] {
        [for (o : Operation | c.operations)]
        @RequestMapping(value = "/[o.name/]", method = RequestMethod.POST)
        public String [o.name/](Model model) {
                ...
                return "[o.name/]";
        }
  [/for]
[/file]
[/template]
```

The result of this template is the following controller:

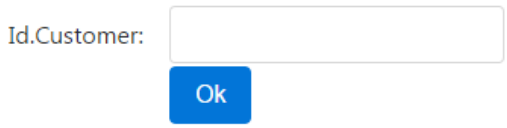
```
@Controller
public class Customer {
        @RequestMapping(value = "/add", method = RequestMethod.POST)
        public String add(Model model) {
                ...
                return "add";
        }
        @RequestMapping(value = "/edit", method = RequestMethod.POST)
        public String edit(Model model) {
                ...
                return "edit";
```

```
	}
	@RequestMapping(value = "/delete", method =
RequestMethod.POST)
	public String delete(Model model) {
		...
		return "delete";
	}

	@RequestMapping(value = "/search", method =
RequestMethod.POST)
	public String search(Model model) {
		...
		return "search";
	}
```

For each class, such as, the class Customer, each action, for exemple the action, searchAction contains two elements : the attribute element indicating the form entred in this action is the actionForm searchForm, and forwards element with forward attribute search.jsp, the remaining actions follow the same principle.

We have demonstrated this process by the the following treatment:



Source code for the searchForm :

```
public class searchForm {

	private int idClt;
	private Customer customer;
	public Customer getCustomer() {
		return customer;
	}
	public void setCustomer(Customer customer) {
		this.customer = customer;
	}
	public int getIdClt() {
		return idClt;
	}
	public void setIdClt(int idClt) {
		this.idClt = idClt;
	}

}
```

Source code for the searchAtion :

```
@Controller
public class Customer {

	@Autowired
	private ICustomer service;

	@RequestMapping(value = "/search")
	public String search(searchForm sf,Model model) {
		Customer
c=service.ConsultCustomer(sf.getIdClt());
		sf.setCustomer(c);
		model.addAttribute("searchForm", sf);
		return "search";
	}

}
```

The method ConsultCustomer(), represents a method from the business layer, can return an object customer from the database.

Source code for the jsp page search.jsp :

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://www.springframework.org/tags/form"
prefix="f" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
	<title>searchCustomer</title>
</head>
<body>
```

```
<div>
    <f:form modelAttribute="searchForm" method="post"
action="search">
      <table>
        <tr>
         <td>Code:</td>
         <td><f:input path="idClt"/></td>
         <td><f:errors path="idClt"></f:errors</td>
        </tr>
        <tr>
         <td><input type="submit" value="ok"></td>
        </tr>
      </table>
    </f:form>
  </div>
<div>
<table>
        <tr>
         <td>name</td>
         <td>${searchForm.customer.name}</td>
        </tr>
      </table>
</div>
</body>
</html>
```

CONCLUSION AND FUTURE WORK

In this paper we applied the MDA to generate the Spring MVC code web application based on uml class diagramme. The purpose of our contribution is to rethink the works presented by Mbarki and Erramdani [7, 8] .

The process involves first defining two meta-models as PIM and PSM respecting the MVC pattern for Spring MVC web applications. Second, we defined the transformation rules for both Model To Model and Model To Text transformation.

through these rules, we can generate an XML file containing all the actions, forms and JSP pages, that can be used to generate the necessary code of the target application.

Furthermore, a case study conducted on generating a Spring MVC model for a customer invoices application, has shown that the approach is valid .

In the future, we aim at extending this work to allow the generation of other models for other frameworks like php, .NET MVC frameworks.

REFERENCES

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

[2] Spring Source Web Site (http://www.springsource.org/).

[3] SpringNet Web Site (http://www.springframework.net/).

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

[5] Object Management Group (OMG). Model Driven Architecture. Retrieved December 12, 2013, from http://www.omg.org/mda/.

[6] Eclipse.org. ATLAS Transformation Language (ATL). http://www.eclipse.org/m2m/atl/.

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

[8] 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.

[9] Bezivin, J., Hammoudi, S., Lopes, D., Jouault, F., "Applying MDA approach for web service platform," In EDOC'04 preceedings of the 8th IEEE International Entreprise Distributed Object Computing Conference, pages 58-70, 2004.

[10] Czarnecki, K., Helsen, S., "Classification of Model Transformation Approaches," in online proceedings of the 2nd OOPSLA'03 Workshop on Generative Techniques in the Context of MDA. Anaheim, October, 2003.

[11] Gharavi, V., Mesbah, A., Deursen, A. V., "Modelling and Generating AJAX Applications: A Model-Driven Approach," Proceeding of the7th International Workshop on Web-Oriented Software Technologies, New York, USA (Page: 38, Year of publication: 2008, ISBN: 978-80-227-2899-7).

[12] Cong, X., Zhang, H., Zhou, D., Lu, P., Qin, L., "A Model-Driven Architecture Approach for Developing E-Learning Platform," Entertainment for Education. Digital Techniques and Systems Lecture Notes in Computer Science, Volume 6249/2010, 111-122, DOI: 10.1007/978-3-642-14533-9_12, 2010.

[13] Mbarki, Rahmouni, "MDA – Based Modeling and Transformation to Generate N – Tiers Web Models,"

Journal of Software, March 2015. DOI:10.17706/jsw.10.3.222-238.

[14] Frédéric J., & Ivan, K. (2006), "Transforming models with ATL," Proceedings of MoDELS 2005 Workshops, LNCS 3844, (pp. 128 – 138), Springer-Verlag Berlin Heidelberg.

[15] Mbarki, S. and Rahmouni, "M. Combining UML class and activity diagrams for MDA generation of MVC 2 web applications," International Review on Computers and Software (I.RE.CO.S), 8(4):949-957 · April 2013.

[16] Roubi, S. , Erramdani, M. and Mbarki, S., "Model Driven Architecture as an Approach for Modeling and Generating Graphical User Interface," Proceedings of the Mediterranean Conference on Information & Communication Technologies 2015, pp.651-656. DOI: 10.1007/978-3-319-30298-0_72.

[17] Rhazali, Y. , Hadi, Y. and Mouloudi A. , "Model Transformation with ATL into MDA from CIM to PIM Structured through MVC," Procedia Computer Science 83:1096-1101 December 2016. DOI: 10.1016/j.procs.2016.04.229.

[18] Essebaa, I. , Chantit, S., "QVT Transformation Rules to Get PIM Model from CIM Model," Europe and MENA Cooperation Advances in Information and Communication Technologies, pp.195-207, January 2017. DOI: 10.1007/978-3-319-46568-5_20.

[19] J. A. Monte-Mor, E. O. Ferreira, H. F. Campos, A. M. da Cunha, and L. A. V. Dias, "Applying MDA Approach to Create Graphical User Interfaces," Eighth International Conference on Information Technology: New Generations, Las Vegas, NV, IEEE, (2011) April 11-13 , p.p. 766-771.

[20] Acceleo, viewed September 2014. https://eclipse.org/acceleo/.

[21] EMF, Eclipse Modeling Framework, viewed September 2014.http://eclipse.org/modeling/emf/.

[22] Liliana Favre, Liliana Martinez, Claudia Pereira, "Modernizing software in science and engineering: From C/C++ applications to mobile platforms," Conference: ECCOMAS Congress 2016 European Congress on Computational Methods in Applied Sciences and Engineering, DOI: 10.7712/100016.2402.4906.