Available online at www.elixirpublishers.com (Elixir International Journal)

Computer Engineering

Elixir Comp. Engg. 80 (2015) 31083-31090

ISSN: 2229-712X

Model-driven transformation for GWT with approach by modeling: from UML model to MVP web applications

Redouane Esbai[1,*] Mohammed Erramdani[2] and Samir Mbarki[3]

[1]Department of Commerce ENCGO, Mohammed 1 University, Oujda, Morocco.

[2]Department of Management ESTO, Mohammed 1 University, Oujda, Morocco.

[3]Department of Mathematics and Computer Science, Faculty of Science, Ibn Tofail University, Kenitra, Morocco.

ARTICLE INFO

Article history:

Received: 29 August 2014;

Received in revised form:

28 February 2015;

Accepted: 14 March 2015;

Keywords

GWT,

Model transformation,

Ria, Metamodel,

Model-View-Presenter,

Transformation rules.

ABSTRACT

The continuing evolution of business needs and technology makes Web applications more demanding in terms of development, usability and interactivity of their user interfaces. To cope with this complexity, several frameworks have emerged and a new type of Web applications called RIA (Rich Internet Applications) has recently appeared providing richer and more efficient graphical components similar to desktop applications. 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 MVP pattern (Model-View-Presenter) for a RIA. We adopt GWT (Google web Toolkit) for creating a target meta-model to generate an entire GWT-based web application.

Introduction

In recent years many organizations have begun to consider MDA (Model-View-Presenter) 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.

Rich Internet applications (RIAs) combine the simplicity of the hypertext paradigm with the flexibility of desktop interfaces. Moreover, RIAs provide a new client-server architecture that reduces significantly network traffic using more intelligent asynchronous requests that send only small blocks of data. In fact, the technological advances of RIAs require from the developer to represent a rich user interface based on the composition of Graphical User Interface (GUI) widgets, to define an event-based choreography between these widgets and to establish a fine grained communication between the client and the server layers. Many frameworks that implement the MVP pattern have emerged; for instance: Mvp4g [1], GWT [2], Echo2 [3], JFace [4], Vaadin [5], ZK [6], Nucleo .NET [7].

GWT is an AJAX framework, developed by Google, which permits us to create RIAs by writing the browser-side code in Java, thus gaining all the advantages of Java (e.g. compiling, debugging, etc.) and generating a generic JavaScript and HTML code that can be executed in any browser.

Moreover, GWT makes every attempt to be flexible allowing us to integrate with other client AJAX frameworks (e.g. Script.aculo.us, Dojo, Yahoo! UI) and with server Java frameworks such as Struts [8], EJB, etc.

In [9][10], the authors have developed a source and a target meta-model. The first was a PIM meta-model specific to class diagrams. The second was a PSM meta-model for N-tiers web applications (particularly Struts, Spring, DTO, Hibernate) without UI. The purpose of our contribution is to produce and generate an RIA PSM model (particularly GWT), implementing MVP pattern, from the class diagram. In this case, we elaborate a number of transformation rules using the approach by modeling and MOF 2.0 QVT, as transformation language, to permit the generation of an XML file that can be used to produce the required code of the target application. The advantage of this approach 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 GWT and the MVP model and its implementation as a framework. The transformation language MOF 2.0 QVT is the subject of the fifth section. In the sixth section, we present the UML and MVP meta-models. In the seventh section, we present the transformation rules using MOF 2.0 QVT from UML source model to the MVP 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 [11][12][13][14][15][16][17][18][19][20][21][22] [23][24]. The authors of the work [19] show how to generate JSPs and JavaBeans using the UWE [18], and the ATL transformation language [17]. Among future works cited, the authors considered the integration of AJAX into the engineering process of UWE. Two other works followed the same logic and have been the subject of two works [15][16]. 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. Meliá, Pérez and Díaz propose in [25] a new approach called OOH4RIA which proposes a model driven development process that extends OOH methodology. It introduces new structural and behavioral models in order to represent a complete RIA and to apply

Tele:

E-mail addresses: es.redouane@gmail.com

transformations that reduce the effort and accelerate its development.In another work [26] they present a tool called OIDE (OOH4RIA Integrated Development Environment) aimed at accelerating the RIAs development through the OOH4RIA approach which establishes a RIA-specific model-driven process.

The Web Modeling Language (WebML) [27] is a visual notation for specifying the structure and navigation of legacy web applications. The notation greatly resembles UML class and Entity-Relation diagrams. Presentation in WebML is mainly focused on look and feel and lacks the degree of notation needed for AJAX web user interfaces [28][29].

Nasir, Hamid and Hassan [23] 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.

This paper aims to finalize the work presented in [9][10], by applying the standard MOF 2.0 QVT to develop the transformation rules aiming at generating the MVP target model with UI. It is actually the only work for reaching this goal.

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 [30] 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 [31] is to develop sustainable models; those models are independent from the technical details of platforms implementation (JavaEE, .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 [32], MOF [33] and XMI [34].

The MVP Pattern

The Model View Presenter is a derivative of the Model View Controller Pattern. Its aim is to provide a cleaner implementation of the Observer connection between Application Model and view.

MVP is a user interface architectural pattern engineered to facilitate automated unit testing and improve the separation of concerns in presentation logic.

Figure 1 shows the architecture of the MVP pattern. The main feature of this pattern is to be composed of:

- The model is an interface defining the data to be displayed or otherwise acted upon in the user interface.
- The view is a passive interface that displays data (the model) and routes user commands (events) to the presenter to act upon that data.
- The presenter acts upon the model and the view. It retrieves data from repositories (the model), and formats it for display in the view.

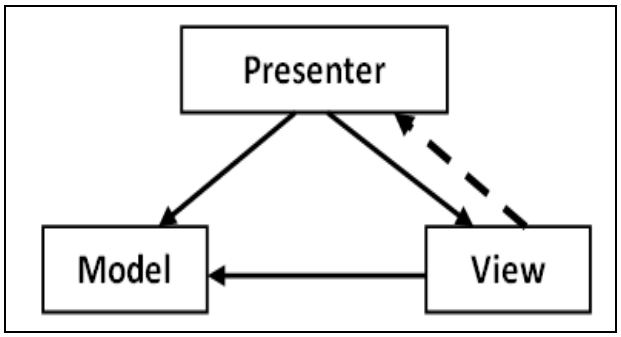

Figure 1. MVP Architecture

Based on this model many frameworks are designed to help developers build the presentation layer of their user interfaces. In the Java community, many frameworks that implements MVP pattern have emerged, among them: Echo2, JFace, Swing, Vaadin, ZK framework, GWT, etc.

The GWT project is one of the best examples. Implementing MVP in Google Web Toolkit requires only that some component implement the view interface.

The GWT framework

Google Web Toolkit (GWT) [35] is an open source web development framework that allows developers to easily create high-performance AJAX applications using Java. With GWT, you are able to write your front end in Java, and it compiles your source code into highly optimized, browser-compliant JavaScript and HTML.

However, GWT is not the only framework for managing the user interfaces. Indeed, other frameworks have been designed for the same goal, but GWT is the most mature. The main advantage of GWT is the reduced complexity compared to other frameworks of the same degree of power, for instance, JFace, Flex and Vaadin.

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.

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 [36]. 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 [37].

Figure 2 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.

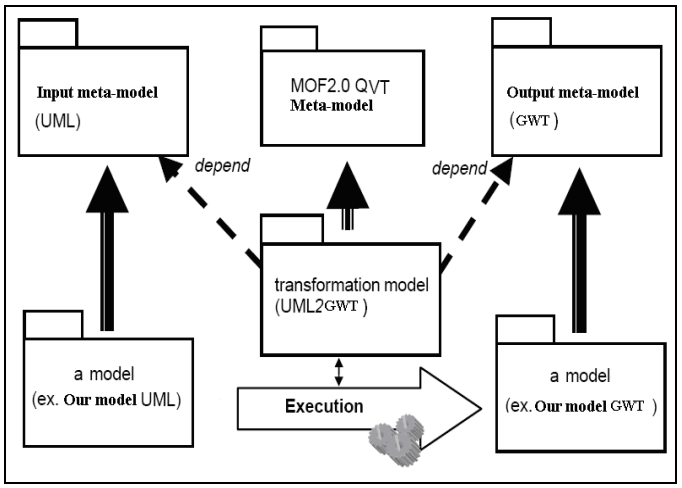

Figure 2. Approach by Modeling

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 3).

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.

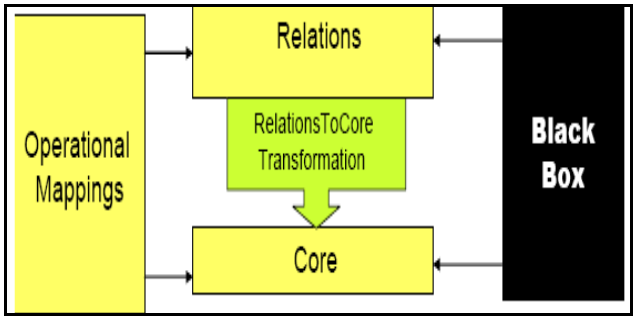

Figure 3. The QVT Structure

This work uses the QVT-Operational mappings language implemented by Eclipse modeling [38].

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.

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 [36] by a model of abstract syntax. The complete abstract syntax ImperativeOCL is shown in Figure 4.

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.

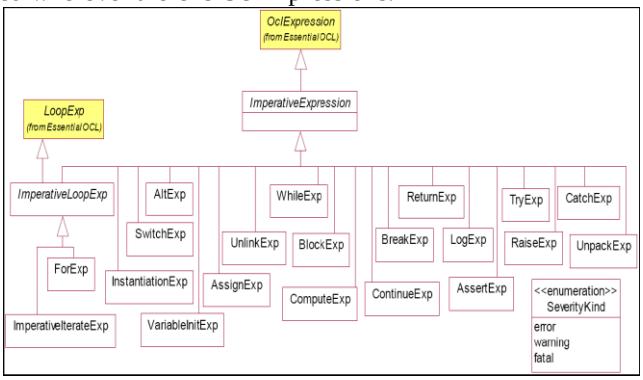

Figure 4. Imperative Expressions of ImperativeOCL

UML and MVP 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 meta-model MVP target.

Meta-model UML source

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 5 illustrates the source meta-model.

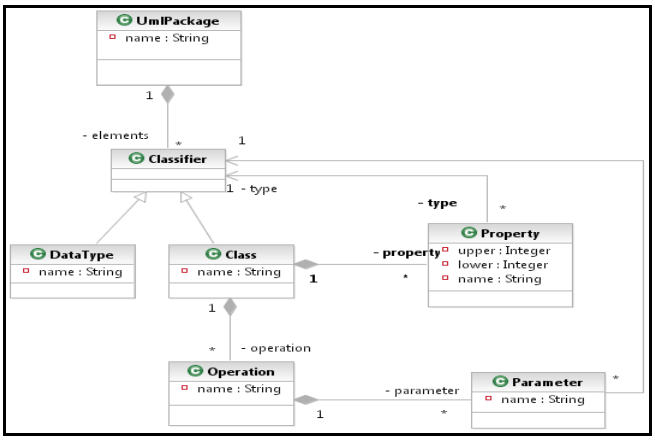

Figure 5. Simplified UML meta-model

- 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.

The works of Mbarki and Erramdani [20] [21] contains more details related to this section topic.

Meta-model GWT MVP target

Our target meta-model is composed of two essential part. Figure 6 illustrates the first part of the target meta-model. This meta-model represents a simplified version of the MVP pattern. It presents the different meta-classes to express the concept of MVP implementation:

- UIPackage: represents the project package. This meta-class is connected to the meta-class MvpPackage
- MvpPackage: represents the different meta-classes to express the concept of MVP. This meta-class is connected to the meta-class ClientPackage and SharedPackage which represents respectively View and Model package.
- GwtXml: expresses the concept of GWT module Encapsulates units of GWT configurations (paths, properties, deferred binding etc); defined in an XML module file and stored in the Java package hierarchy.
- ClientPackage: represents the client package, in this package, we will typically find, and put, all the code required for the client side part of our application (the part in the browser). This meta-class is connected to the meta-class PresenterPackage and ViewPackage
- MainApp: this meta-class implements EntryPoint interface. When a module is loaded, entry point class is instantiated and its onModuleLoad() method gets called.
- EntryPoint: represents the concept of entry point interface containing the method onModuleLoad(). Implement this interface to allow a class to act as a module entry point.
- PresenterPackage: represents the different meta-classes to express the concept of Presenter. This Presenter is Responsible for getting the data, driving the view, listening for GUI events, implements business logic
- IPresenter: represents the concept of basic presenter interface that all of our presenters will implement and containing the methods bind() and go()
- PresenterImpl: expresses the concept of specific Presenter implementation all methods to bind and go are implemented in this meta-class.
- Display: represents the concept of the inner interface type of the view is determined by the getView() method.
- View: expresses the concept of the view contains all of the UI components that make up our application.
- SharedPackage: represents package which contains the different meta-classes to express the concept of model.
- Pojo: represents the concept of pojo. The latter extends the meta-class Class. The pojos represents objects in the area of application.
- Widget: expresses the concept of the GWT Widget.

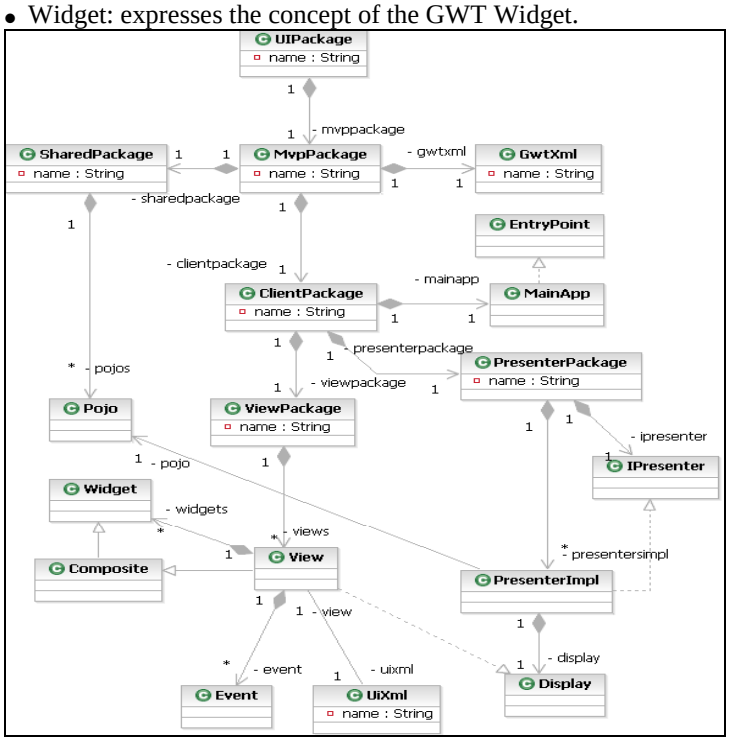

Figure 6. The proposed MVP metamodel

Figure 7 illustrates the second part of target meta-model. Like the Abstract Window Toolkit (AWT) and Swing, GWT is based on widgets. To create a user interface, you instantiate widgets, add them to panels, and then add your panels to the application's root panel, which is a top-level container that contains all of the widgets in a particular view. GWT contains many widgets whose classes are described by an inheritance hierarchy. An illustration of some of those widgets is shown in Figure7.

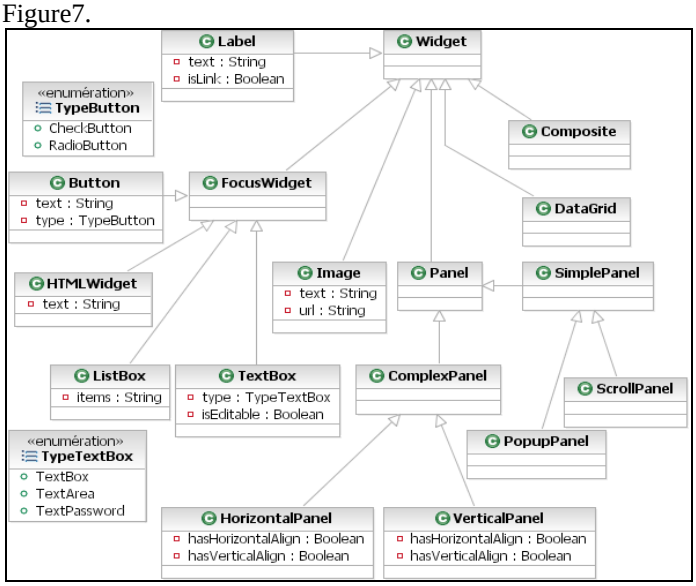

Figure 7. Simplified GWT metamodel

- Panel: A panel that lets you place widgets a pixel locations.
- Button: A button that the user can click.
- Composite: An opaque wrapper for a set of widgets.
- DataGrid: A table that arranges its widgets in a grid.
- HorizontalPanel: A panel that arranges its widgets horizontally.
- VerticalPanel: A panel that arranges its widgets vertically.
- Image: An image that can fire load events when it loads its corresponding image file.
- Label: Text that supports word wrap and horizontal alignment.
- PopupPanel: A panel that pops up when it's shown.
- ScrollPanel: A panel that automatically adds scrollbars to itself on demand.
- TextBox: A single-line text widget.
- ListBox: A list of choices that the user can select.

The process of transforming UML source model to MVP target model

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.

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 8). After applying the transformation on the UML model, composed by the class Employee, we generated the target model (see Figure 11).

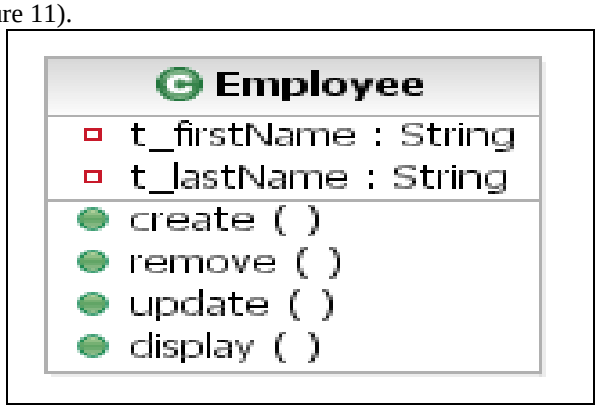

Figure 8. UML instance model

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 gwtModel:UIPackage
begin
create UIPackage crudProjectPackage
create MvpPackage mvpPackage
create ClientPackage clientPackage
create MainApp mainapp
link mainapp to clientPackage
create PresenterPackage presenterPackage
create IPresenter ipresenter
ipresenter.name = 'IPresenter'
ipresenter.methods = declaration of {do,bind}
link ipresenter to presenterPackage
for each e ∈ source model
        x = transformationRuleOne(e)
        link x to presenterPackage
end for
create SharedPackage sharedPackage;
for each e ∈ source model
        x = transformationRuleThree(e)
        link x to sharedPackage
end for
create GwtXml gwtxml;
link presenterPackage to clientPackage
link viewPackage to clientPackage
link clientPackage to mvpPackage
link mvpPackage to crudProjectPackage
link sharedPackage to crudProjectPackage
link gwtxml to crudProjectPackage
return crud
end
function
transformationRuleOne(e:Class):PresenterImpl
begin
    create PresenterImpl presenterImpl
    presenterImpl.name = e.name+ 'PresenterImpl'
for each e1 ∈ PresenterPackage
    if e1.name = 'I'+e.name+ 'Presenter'
      put e1 in interfaces
    end if
end for
link interfaces to presenterImpl
return presenterImpl
end
function
transformationRuleTwo(e:Class):ViewPackage
begin
create ViewPackage vp
for each e ∈ source model
    if e.methods.name ≠ 'remove'
      create View page
      link  page to vp
end if
end for
return vp
end
function
transformationRuleThree(e:Class):Pojo
begin
create Pojo pj
pj.name = e.name
pj.attributes = e.properties
return pj
end
```

Figure 9 illustrates the first part of the transformation code of UML source model to the MVP target model.

```
end for
create ViewPackage viewPackage;
for each e ∈ source model
        x = transformationRuleTwo(e)
        link x to viewPackage
```

```
uml2gwt.qvto
modeltype UMLMM uses "http:///umlMM.ecore";
modeltype gwtMM uses "http:///GwtMM.ecore";

transformation uml2gwt(in umlModel:UMLMM, out gwtModel:gwtMM);
main() {
    umlModel.objects()[UmlPackage]->map UmlPackage2CrudProjectPackage();
}

mapping UmlPackage::UmlPackage2CrudProjectPackage () : UIPackage {
    name:= 'crud'+self.name;
    mvppackage:= object MvpPackage {
        name:= 'mvpPackage';
        clientpackage:= object ClientPackage {
            name := 'clientPackage';
            mainapp:= object MainApp {
                name:= 'MainApp';
            };
            presenterpackage := object PresenterPackage {
                name:= 'presenter';
                ipresenter:= map class2IPresenter();
                presentersimpl:= umlModel.objects()[Class]->map class2PresenterImpl();
                };
            viewpackage := object ViewPackage {
                name:= 'view';
                views:= umlModel.objects()[Class].map class2view();
                };
        };
        sharedpackage := object SharedPackage {
            name:= 'shared';
            pojos:= umlModel.objects()[Class]->map class2Pojo();
        };
        gwtxml:= object GwtXml {
            name:= result.name;
        }
    };
}
```

Figure 9. The transformation code UML2Gwt

The transformation uses as input a UML type model, named umlModel, and as output a GWT type model named gwtModel. 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 UIPackage output model.

The objective of the second part of this code is to transform a UML package to GWT package, by creating the elements of type package 'Presenter', 'View' and 'Shared'. It is a question of transforming each class of package UML, to IPresenter and PresenterImpl in the Presenter package, and to Pojo, in the Shared package, to Dispaly contains widgets in the View Package without forgetting to give names to the different packages.

```
mapping Operation::Op2View (cl:Class) : View{
    name:=self.name+cl.name+'View';
    widgets := Sequence {
        object Panel {
            name:=self.name+'Panel';
            }
    };
    if(self.name<>'display')then
        widgets+= cl._property.map property2widget(self.name)
    endif;
    if(self.name='display')then
        widgets+= object DataGrid {
            name:=self.name+cl.name+'s';
        }
    endif;
    if(self.name='create')then
        widgets+=object Button {
            name:=self.name+'Button';
        }
    endif;
    if(self.name='update')then
        widgets+=object Button {
            name:=self.name+'Button';
        }
    endif;
    widgets+=object Button {
        name:='cancelButton';
    }
}
```

Figure 10. The mapping Operation2View

The methods presented in Figure 10 means that each operation in a class corresponds to View. The codes and models are publicly available online http://sites.google.com/site/uml2mvp/.

Result

Figure 11 shows the result after applying the transformation rules.

The first element in the generated PSM model is UIPackage which includes MvpPackage that contains gwt.xml file, Client Package and Shared Package. The Client Package contains the main application, the Presenter Package and the View Package that contains the Three Views, namely CreateEmployeeView, DisplayEmployeeView and UpdateEmployeeView. Since the operation of the removal requires any view, we'll go to every view element, which contains a multiple element widget like Panel, firstNameTextBox, lastNameTextBox, actionButton and cancelButton. Since the view Display contains the DataGrid widget that contains removal button.

The Presenter Package includes one presenter' interface, one presenter' implementation that contains methods with their parameters and their implementations and the last package element in the generated PSM model is Shared Package which contains one Pojo' object that contains their attributes correspond to the object 'Employee'.

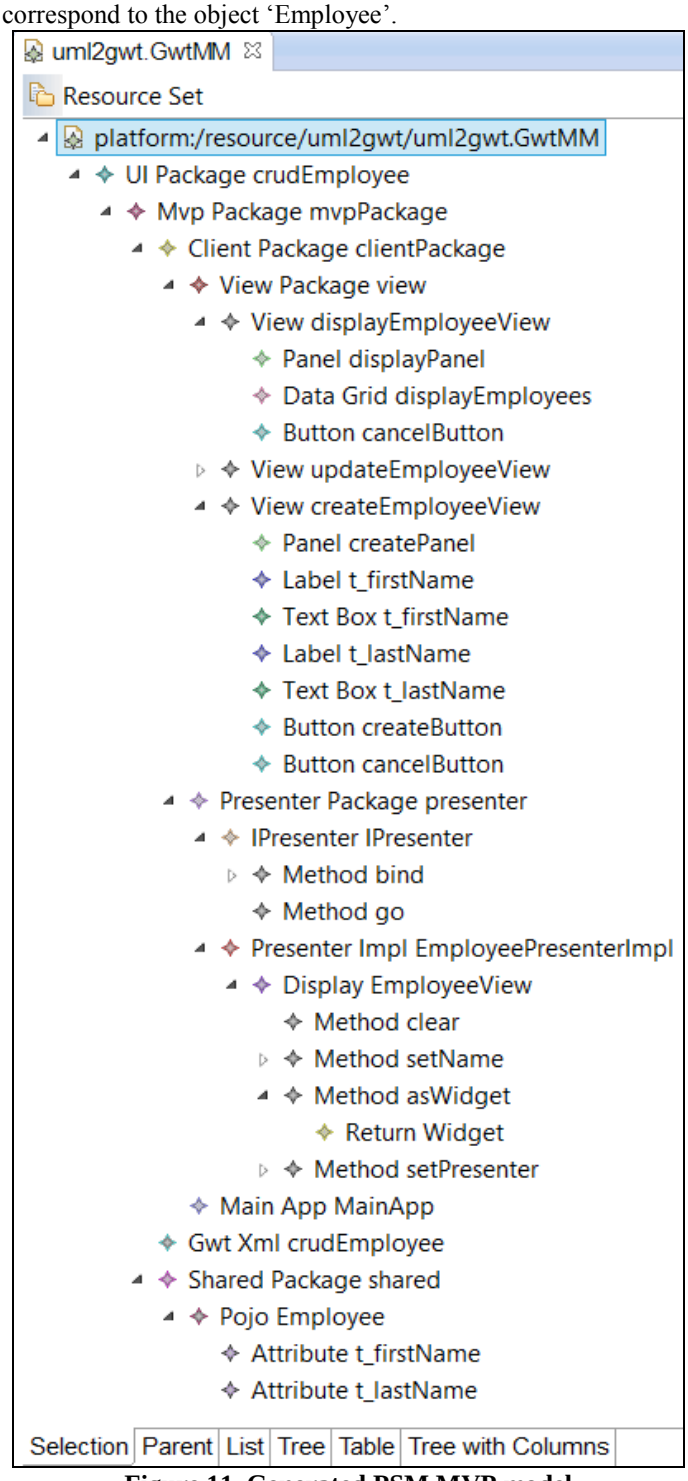

Figure 11. Generated PSM MVP model

Conclusion and perspectives

In this paper, we applied the MDA approach to generate the MVP web application based on UML class diagram.

The purpose of our contribution is to finalize the works presented in [9] [10]. This involves developing all meta-classes needed to be able to generate a GWT application respecting a MVP pattern and then we applied the approach by modeling and used the MOF 2.0 QVT standard as a transformation language. The transformation rules defined allow browsing the source model instance class diagram, and generating, through these rules, an XML file containing layers of MVP architecture according to our target model. This file can be used to produce the necessary code of the target application. The algorithm of transformation manages all CRUD operations. Moreover, it can be re-used with any kind of methods represented in the UML class diagram.

In the future, this work should be extended to allow the generation of other components of Web application besides the configuration files. Afterward we can consider integrating other frameworks like Flex and JFace.

References

[1] Mvp4g A framework to build a GWT application the right way (https://code.google.com/p/mvp4g/)

[2] GWT source web site (https://code.google.com/p/google-web-toolkit/)

[3] Echo2 source web site (http://echopoint.sourceforge.net/)

[4] Harris, Robert; Warner, Rob, The Definitive Guide to SWT and JFACE (1st ed.), (Apress, 2004).

[5] Vaadin Framework web site (https://vaadin.com/home)

[6] ZK framework web site (http://www.zkoss.org)

[7] Nucleo .NET framework web site (http://nucleo.codeplex.com/)

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

[9] Esbai. R, Erramdani, M., Mbarki, S., Arrassen. I, Meziane. A. and Moussaoui. M., Model-Driven transformation with approach by modeling: From UML to N-tiers Web Model, International Journal of Computer Science Issues (IJCSI) , Vol. 8, Issue 3, May 2011, ISSN (Online): 1694-0814

[10] Esbai. R, Erramdani, M., Mbarki, S., Arrassen. I, Meziane. A. and Moussaoui. M., Transformation by Modeling MOF 2.0 QVT: From UML to MVC2 Web model, InfoComp - Journal of Computer Science, vol. 10, no. 3, p. 01-11, September of 2011, ISSN 1807-4545.

[11] AndroMDA web site (http://www.andromda.org/).

[12] Bezivin, J., Busse, S., Leicher, A., Suss, J.G, Platform Independent Model Transformation Based on TRIPLE. Proceedings of the 5th ACM/IFIP/USENIX International Conference on Middleware, (Page: 493, Year of publication: 2004).

[13] Bezivin, J., Hammoudi, S., Lopes, D., Jouault, F., Applying MDA approach for web service platform. Proceedings of the 8th IEEE International Enterprise Distributed Object Computing Conference, (Page: 58, Year of publication: 2004).

[14] 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, 2010.

[15] Distante, D., Rossi, G., Canfora, G., Modeling Business Processes in Web Applications: An Analysis Framework. In Proceedings of the The 22nd Annual ACM Symposium on Applied Computing (Page: 1677, Year of publication: 2007, ISBN: 1-59593-480-4).

[16] 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)

[17] Jouault, F., Allilaire, F., Bézivin, J., Kurtev, I., ATL: A model transformation tool. Science of Computer Programming-Elsevier Vol. 72, n. 1-2: pp. 31-39, 2008.

[18] Koch, N., Transformations Techniques in the Model-Driven Development Process of UWE, Proceeding of the 2nd International Workshop Model-Driven Web Engineering, Palo Alto (Page: 3 Year of publication: 2006 ISBN: 1-59593-435-9).

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

[20] Mbarki, S., Erramdani, M., Toward automatic generation of mvc2 web applications, InfoComp - Journal of Computer Science, Vol.7 n.4, pp. 84-91, December 2008, ISSN: 1807-4545.

[21] Mbarki, S., Erramdani, M., Model-Driven Transformations: From Analysis to MVC 2 Web Model, International Review on Computers and Software (I.RE.CO.S.), Vol. 4. n. 5, pp. 612-620, September 2009.

[22] Mbarki, S., Rahmouni, M., Erramdani, M., Transformation ATL pour la génération de modèles Web MVC 2, Proceeding of the 10e Colloque Africain sur la Recherche en Informatique et en Mathématiques Appliquées, Theme5:Information Systems, CARI (Year of publication: 2006).

[23] Nasir, M.H.N.M., Hamid, S.H., Hassan, H., WebML and .NET Architecture for Developing Students Appointment Management System, Journal of applied science, Vol. 9, n. 8, pp. 1432-1440, 2009

[24] Ndie, T. D., Tangha1, C., Ekwoge, F. E., MDA (Model-Driven Architecture) as a Software Industrialization Pattern: An Approach for a Pragmatic Software Factories. J. Software Engineering & Applications, pages 561-571, 2010

[25] Meliá S., Gómez J., Pérez P., Díaz O., A Model-Driven Development for GWT-Based Rich Internet Applications with OOH4RIA, Proceedings of ICWE '08. Eighth International Conference on, Yorktown Heights, NJ, (Page: 13, Year of publication: 2008, ISBN: 978-0-7695-3261-5).

[26] Meliá S., Gómez J., Pérez S., Diaz O. Facing Architectural and Technological Variability of Rich Internet Applications. IEEE Internet Computing, vol. 99, pp.30-38, 2010.

[27] S. Ceri, P. Fraternali, and A. Bongio. Web modeling language (WebML): a modeling language for designing web sites. Computer Networks, vol. 33(1-6) pp137–157, 2000.

[28] Preciado J. Carlos, M. Linaje, S. Comai, and F. Sanchez-Figueroa. Designing Rich Internet Applications with Web engineering methodologies. Proceedings of the 9th IEEE International Symposium on Web Site Evolution (WSE'07)(Page: 23 Year of publication: 2007).

[29] Trigueros M. L., J. C. Preciado, and F. S´anchez-Figueroa. A method for model based design of Rich Internet Application interactive user interfaces. In ICWE'07: Proceedings of the 7th International Conference Web Engineering (page: 226 Year of publication: 2007).

[30] Miller, J., Mukerji, J., al. MDA Guide Version 1.0.1 (OMG, 2003).

[31] Blanc, X., MDA en action : Ingénierie logicielle guidée par les modèles (Eyrolles, 2005).

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

[33] Meta Object Facility (MOF), version 2.0 (OMG, 2006)
[34] XML Metadata Interchange (XMI), version 2.1.1 (OMG, 2007),
[35] GWT project web site http://www.gwtproject.org/
[36] Meta Object Facility (MOF) 2.0 Query/View/Transformation (QVT), Version 1.1 (OMG, 2009).
[37] Czarnecki, K., Helsen, S., Classification of Model Transformation Approaches, Proceedings of the 2nd OOPSLA'03 Workshop on Generative Techniques in the Context of MDA. Anaheim (Year of publication: 2003).
[38] Eclipse modeling, http://www.eclipse.org/modeling/.