Modeling and Generating Graphical User Interface for MVC Rich Internet Application using a Model Driven Approach

Sarra Roubi, Mohammed Erramdani
MATSI Laboratory, EST
Mohammed First University
Oujda, Morocco
roubi.sarra@gmail.com, m.erramdani@gmail.com

Samir Mbarki
Department of Computer Science
Ibn Tofail University
Kenitra, Morocco
mbarkisamir@hotmail.com

Abstract—Rich Internet Applications (RIA) have been proposed as a response to growing user needs. Indeed, they have combined the richness and interactivity of desktop interfaces into the web distribution model. However, RIAs are complex applications and their development requires designing and implementation which can be time-consuming and the available tools are specialized in manual design. Besides, the design of a usable Graphical User Interface (GUI) with the layout respecting the user’s wishes remains a challenge. In this paper, we present a model driven development process to generate RIAs, focusing on the graphical interface, based on simplifying the task for the designer and not necessary be aware of the implementation specification.

Keywords—Model Driven Architecture; Graphical User Interface; Model; Meta Model; Transformation; Code generation

I. INTRODUCTION

With the evolution of the Web and its features, HTML-based Web applications have shown their limits, especially when it comes to integrate complex activities to be performed via Graphical User Interfaces (GUI). To remedy this, Rich Internet Application, known as RIA, are given as a solution to overcome these limitations. A RIA is characterized by its combination of the advantages of Web distribution model with the interactivity and richness of the interface of desktop applications.

However, the design and implementation of the GUI for RIA is known for its complexity and difficulty using the existing tools. In addition, designers need to know the computer platforms, the characteristics of the user, the interaction of the environment, etc ... which can become tedious, time consuming and requires additional efforts.

In addition to that, we can say that modeling approach is an efficient way to master complexity and ensure consistency of application. Indeed, The Model Driven Architecture (MDA) [1][7] approach offers significant progress in terms of the development of software applications and helps increase productivity, improve significantly the sustainability and provides better ways to deal with changing requirements.

We can say that such a model-driven approach adopted in the process of development of RIAs [8] can significantly reduce the risk of rework and improve the quality.

In this context, the paper presents a new approach based on the MDA (Model Driven Architecture) paradigm and proposes a complete development process based on a set of models and transformations that allows obtaining the implementation of GUI for Rich Internet Applications with JavaFX platform as a target respecting the MVC pattern.

The paper is organized as follows. In section II we discuss related work dealing with Model-Driven development approaches. Then, in section III we present the Model Driven Architecture approach and its principles. Section IV describes the proposed approach and technologies used to develop it, while in section V we report the result of the case study of designing and generating the RIA to validate the approach. The section VI concludes and presents the future works.

II. RELATED WORK

Several works dealing with GUI’s automatic generation have emerged recently. For Kapitaski the methodology proposes a model based approach and advocates in favor of a complete separation of the web application functionality from the context adaptation at all development phases [2]. Also, a Rich Internet Application for web based product development was presented for the integration of flexible web based GUI in [3].

Besides, the approach called OOH4RIA which proposes a model driven development process that extends OOH methodology is presented in [4]. Also, a combination of the UML based Web Engineering (UWE) method for data and business logic modeling with the RUX-Method for the user interface modeling of RIAs was proposed as model-driven approach to RIA development [5].

Also, Sottet based their research on the plasticity of User Interface and the application of MDA concepts for the purpose of unifying the modeling for GUI is presented [6]. Another related work on applying MDA approach for Rich Internet Applications is found in [10]. The approach is based on XML
User Interface description languages using XSLT as the transformation language between the different levels of abstraction. Also, we find WebML (Web Modeling Language), an already established conceptual model for data-intensive Web applications [11]. Also, WebRatio which is a classic BPM solutions is presented and used in several works [13].

In addition to that, an MDA approach for AJAX web applications [12] was the subject of a study that proposes a UML scheme by using the profiling for modeling AJAX user interfaces, and report on the intended approach of adopting ANDROMDA for creating an AJAX cartridge to generate the corresponding AJAX application code, in ICEFACES, with back-end integration. A meta model of AJAX has been defined using the AndroMDA tool.

Note that these works focused on using a UML profiling to define new ways to model the RIAs application. That is to say, the user should be aware of the UML modeling language on one hand and of the technical details of the execution platform on the other hand. In our case, we wanted to keep the task for designer as simple as possible and translate the user’s needs in terms of operations and goals to achieve. So, we proposed new meta models as the proposed methodology is described in the following sections.

III. MODEL DRIVEN ARCHITECTURE

A. The Object Management Group Approach and Acronyms

The OMG presents this approach as a way to develop systems that offer greater flexibility in the evolution of the system while remaining true to customer needs and satisfaction.

MDA introduces an approach related to system specification which relies on the separation into three different layers of abstraction; Computing Independent Model (CIM), Platform Independent Model (PIM) and the Platform Specification Model (PSM). We can define the three layers as follow and as shown in Fig.1:

- **CIM**: It represents a high level specification of the system’s functionalities. It is often seen as a business model, as it uses a vocabulary that is familiar to the subject matter experts. It shows exactly what the system is supposed to do, but hides all the technology specifications.

- **PIM**: It allows the extraction of the common concept of the application independently from the platform target. It exhibits a sufficient degree of independence so as to enable its mapping to one or more platforms.

- **PSM**: It combines the specifications in the PIM with the details required of the platform to stipulate how the system uses a particular type of platform which leads to include platform specific details.

The OMG’s Meta Object Facility (MOF) defines a model as an instance of a meta model. The meta model may describe the properties of a particular platform and the models that are instances of such a meta model are said to be platform-specific.

![Fig. 1. The levels separation in the Model Driven Engineering approach](image)

B. Transformation with MOF 2.0 QVT

Thus, the MDA approach allows for the same model to be implemented on multiple platforms through the execution of models’ transformation. A transformation converts models with a particular perspective from one level of abstraction to another, usually from a more abstract to less abstract view, by adding more detail supplied by the transformation rules as illustrated in Fig. 2.

There are two types of transformations in the MDA approach:

- **Model To Model**: it concerns the transition from CIM to PIM or from PIM to PSM
- **Model To Text**: it concerns the generation of the code from the entry model (the PSM) to a specific programming language as a target.

![Fig. 2. Transformation process in the Model Driven Architecture approach](image)

Using the modeling approach in MDA, the Model To Model transformation is designed to have a sustainable and productive models’ transformation, independently of any execution platform. This is why the OMG has developed a
standard for this transformation language which is the MOF 2.0 QVT [9], standing for Query View Transformation. QVT is hybrid character (declarative / imperative) consisting of three languages: QVT-Relation, QVT-Operational and QVT-Core. Fig. 3 shows this distribution.

![QVT 2.0 architecture](image)

**Fig. 3.** QVT 2.0 architecture

For this work, we used the QVT-Operational mappings language implemented by Eclipse modeling [8].

For the Model To Text transformation, we defined a template for the JavaFX platform 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.

IV. THE MODEL-DRIVEN DEVELOPMENT PROPOSED PROCESS

We propose a process based on the MDA principles in order to model and generate the RIA respecting MVC Pattern focusing on the Graphical User Interface:
- We first define the PIM and PSM meta models as presented in the following section,
- Second, we establish the transformation rules and implementing the QVT transformation engine.
- Finally, for generation of the code for the RIA respecting the MVC pattern we elaborate the templates using Acceleo.

A. The PIM and PSM proposed meta models

For our approach, we first develop the PIM meta model. We focused on keeping the terms as simple as possible so the designer does not have to learn a new language or a complex design process, as it was the case for the approach by profiling the UML models in [4][12].

Indeed, the meta model translates the user’s vision of the graphical user interface in terms of actions and interactions, by describing what is expected from the application from a graphical point of view based on the notion of the use case.

The major goal of the view is described through the use case that is divided into several main operations that gathers the atomic actions leading to achieve that goal. We added an enumeration of the basic actions’ types that any user is familiar with (input, selection, clicks...). Finally, we assigned a property in each action (is it a password, a single choice …), to help choose the most appropriate widget while defining the transformation rules.

Also, we added a new specification to help choose the best layout in the final GUI based on the user’s wishes. The user defines positions like in a simple grid by providing the pair (x-axis, y-axis) in the Disposition property. This simple information is then transformed into layout disposition information in the output file that will simplify the task. All these elements are embraced in the UMLPackage. Fig. 4 presents a simplified version of the proposed meta model.

![Proposed PIM meta model for GUI](image)

**Fig. 4.** Proposed PIM meta model for GUI

Second, we defined the PSM meta model for the JavaFX taking into account the relation between the different layers of the MVC architecture. This will avoid the GUI designer to worry about understanding the dynamics of appearance between the different layers of application.

That is to say, as shown in the Fig. 5, we have the three packages: ViewPackage, ModelPackage and ControllerPackage that gather respectively the application’s views, models and controllers.

The model includes the methods that will be triggered when a user event arises. This activation is done through the Handlers that are defined for each component in the controller.

For the View, the scene is composed of graphical components, named controls that could be containers as Roots. Also, we added a hierarchical relationship between graphical components based on their type. Besides, we associated each component with its position in the defined root; that will divide the whole scene so we can put each component in a specific position.
B. The transformation process: Model To Model (M2M)

Once the modeling phase established, comes the most important phase in the MDA approach, which is the transformation process. Indeed, we defined the transformation rules that we developed into a transformation engine using the standard QVTo from the PIM to the defined PSM.

The entry point of the transformation, as shown in Fig. 6, is the main method. This method makes the correspondence between all elements of type `UmlPackage` of the input model and the elements of type `JavaFXPackage` output model.

C. The transformation process: Model To Text (M2T)

When the transformation rules defined and the engine is sufficiently working, the second step consists on generating the code from the obtained PSM model instance. The template of the target JavaFX for MVC is developed using Acceleo, the OMG standard. An excerpt of the template is presented below:

```java
[comment encoding = UTF-8 ]
[module generateJFX('http://javafxmmMVP/1.0')]
[template public generateJFXCode(aJavaFXPackage : JavaFXPackage)]
[for (theScene : Scene | aJavaFXPackage.VPackage.views)]
[file ('/view/'+aJavaFXPackage.VPackage.Name + '/' + theScene.Name.toUpperFirst() + '.fxml', 'UTF-8')]
<xml version="1.0" encoding="UTF-8"/>
<AnchorPane xmlns:fx="http://javafx.com/fxml/1" xmlns-"http://javafx.com/javafx/2.2" fx:controller="controller.[theScene.hasController.Name.escapeSpec ialChar().trim()]">
<children>
  <BorderPane>
    <AnchorPane>
      <GridPane>
        <[if (widget.oclIsTypeOf(label))]Label text="[widget.oclAsTypeOf(label).Text]"/>
        <[for (widget : Control | theRoot.hasWidgets)]>
          <[if (widget.oclIsTypeOf(label))]Label text="[widget.oclAsTypeOf(label).Text]"/>
        </[for]>
      </GridPane>
    </AnchorPane>
  </BorderPane>
</children>
</AnchorPane>
```

For the definition of the right graphical components of the view, we defined the transformation rules based on the properties of each activity. The following table shows an example of these transformations.

<table>
<thead>
<tr>
<th>Activity</th>
<th>PIM GUI elements</th>
<th>PSM GUI Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>isPassword</td>
<td>passwordField</td>
</tr>
<tr>
<td></td>
<td>isTextArea</td>
<td>TextField</td>
</tr>
<tr>
<td>Selection</td>
<td>isSingleChoice</td>
<td>CheckBox</td>
</tr>
<tr>
<td></td>
<td>isDeployable</td>
<td>ComboBox</td>
</tr>
<tr>
<td></td>
<td>N/A</td>
<td>RadioButton</td>
</tr>
<tr>
<td>Click</td>
<td>isLink</td>
<td>Link</td>
</tr>
<tr>
<td></td>
<td>N/A</td>
<td>Button</td>
</tr>
<tr>
<td>Label</td>
<td>N/A</td>
<td>Label</td>
</tr>
</tbody>
</table>
[elseif widget.oclIsTypeOf(TextField)] TextField onAction = "#widget.oclAsType(Button).Text.text onAction = "#widget.oclAsType(Button).Text.text"
[elseif widget.oclIsTypeOf(PasswordField)] PasswordField onAction = "#widget.oclAsType(Button).Text.text onAction = "#widget.oclAsType(Button).Text.text"
[elseif widget.oclIsTypeOf(TextField)] TextField onAction = "#widget.oclAsType(Button).Text.text onAction = "#widget.oclAsType(Button).Text.text"
[elseif widget.oclIsTypeOf(PasswordField)] PasswordField onAction = "#widget.oclAsType(Button).Text.text onAction = "#widget.oclAsType(Button).Text.text"
[elseif widget.oclIsTypeOf(Button)] Button onAction = "#widget.oclAsType(Button).Text.text onAction = "#widget.oclAsType(Button).Text.text"
[elseif widget.oclIsTypeOf(Link)] Hyperlink onAction = "#widget.oclAsType(Button).Text.text onAction = "#widget.oclAsType(Button).Text.text"

In our approach, we defined an automation of the process to develop RIA and focus on the graphical part. Using this method requires no knowledge of the technical aspects and specification of the platform. However, the user should know the structure of the PIM meta model and describe sufficiently the input model, otherwise the generated files will not reflect all the user needs.

V. RUNNING EXAMPLE: PRODUCT CATALOG

In order to validate our approach, we applied it to automatically generate a simple RIA for on-line catalog consultation. The application enables the users to: search for product by characteristics and view a product’s result list. We applied our approach by defining first the input model only using the PIM meta model described above as shown in the Fig. 7.

The model is designed in terms of main goal described in a use case “Search Product”, and the main operations and the activities describing it. Also, we defined how we wanted to arrange these activities in the view in terms of cell positions with the layout disposition property.

![XML file input for the catalog application](image)

Fig. 7. XML file input for the catalog application

Note that the entry model does not require neither a knowledge of the MVC architecture nor the Rich Internet Application structure; but it describes what is expected from the application in terms of interactions and user activity. As a result of the QVTo transformation, we have an output model that describes sufficiently a MVC JavaFX application according to the user’s wishes as defined in the input model above.

![The result model generated respecting JavaFX MVC](image)

Fig. 8. The result model generated respecting JavaFX MVC

This output model, Fig. 8, describes the entry point to the code generation phase executed by the code generator developed using Acceleo in Eclipse. This gives us all the application’s java code sources, FXML files for the views, packages and relationships between all of the layers of the MVC pattern.

![The research product view automatically generated](image)

Fig. 9. The research product view automatically generated
Once the application generated, it can be loaded into the Eclipse IDE as a JavaFX application with the packages generated and the java classes for the different layers, in particular the FXML files for the View part since we focused our approach on the graphical aspect of the generated application, without neglecting the connection with the other layers to respect the MVC pattern. Fig. 9 shows the screenshot of the generated view the application: the search product view with the form for the criteria of the research and a list that will give all the available products.

VI. CONCLUSION AND PERSPECTIVES

In this paper, we applied a Model Driven Architecture approach to generate a Rich Internet Application. The process involves developing meta models, as PIM and PSM, to be able to generate a JavaFX application respecting a MVC pattern. For the transformation phase, we applied the approach by modeling and used the MOF 2.0 QVT standard as a transformation language, to finally define templates with Acceleo for the code generation. A case study conducted on designing a product search catalog RIA. We focus on keeping the notion as simple as possible and abstract all the technical details concerning the RIA and MVC pattern so the generation don’t necessarily require any knowledge of the technical aspects of the platform.

In the future, this work will be extended to allow the generation of other components of RIAs besides the configuration files and integrate other complex layout manager for the graphical components. We also aim at defining a supporting graphical tool as an Eclipse plug-in to simplify more the use of the approach.

References