Model Driven Fast Prototyping of RIAs: From Conceptual Models to Running Applications

Mario Luca Bernardi\textsuperscript{1}, Giuseppe Antonio Di Luca\textsuperscript{2}, Damiano Distante\textsuperscript{3}  
\textsuperscript{1,2}Department of Engineering - RCOST, University of Sannio, Italy  
\textsuperscript{3}Unitelma Sapienza University, Italy 
\textsuperscript{1}mlbernar@unisannio.it, \textsuperscript{2}dilucca@unisannio.it, \textsuperscript{3}damiano.distante@unitelma.it

Abstract—Fast prototyping is a quick and cost effective development of a (minimum) viable version of a software useful to some purpose (e.g., requirements verification or design validation), which can be discarded or refactored to become the version of the software to be delivered. In this paper we propose a model-driven approach for the fast prototyping of Rich Internet Applications (RIAs). Starting from the conceptual model of a RIA, intermediate models and the source code of a ready-to-deploy application prototype are automatically generated through a model driven development process which exploits well known model-driven engineering frameworks and technologies including Eclipse EMF, GMF, and Xpand. Compared to traditional, non model-driven, prototyping approaches, our proposal allows to drastically reduce the overall prototyping effort to just the effort required to define the conceptual model of the application, as the rest of the process is substantially automatic. The paper describes the overall RIA prototyping approach, the supporting tools and adopted technologies, along with the results from a case study carried out for validation and verification purposes.

Keywords: Rich Internet Applications, Model Driven Web Engineering, Fast Prototyping, MOF, Metamodel, EMF, GMF, Xpand, MVC, JavaServer Faces

I. INTRODUCTION

In the last ten years, we have seen Web Applications (WAs) to move from the traditional (Web 1.0) ones, mainly and mostly static applications where the most of information processing was on the server side while the client had the main responsibility of displaying static contents, to the nowadays so-called Rich Internet Applications (RIAs). RIAs are characterized by a larger and better interaction with end users that have not just a passive role (i.e., just a consumer of contents stored and processed on server side of the application), but have assumed an active role that contributes, by the choice/selection she/he makes, to the definition or processing of the contents to be provided on the client side. RIAs, in addition to show a better usability, allow to have better performances because they reduce significantly the number of server requests by processing on the client many of the operations and the handling of data requested by users, thus reducing page reloading and traffic on the Internet. Indeed, RIAs are based on a (new) client-server architecture style using asynchronous requests composed of small(er) blocks of data. So, RIAs are more similar to desktop applications because they combine “the Webs lightweight distribution architecture with desktop applications interface interactivity and computation power, and the resulting combination improves all the elements of a WA (data, business logic, communication, and presentation).” [7]. Synthetically, a RIA can be seen as “a Web application in which the UI is processed on the client side and the business logic is defined by a services backend.” [19].

The availability of new technologies and platforms, such as AJAX, Microsoft Silverlight, Adobe Flex, has made fast the transition to the development of RIAs. However, being WAs, also RIAs suffers most of the typical problems of WAs development. Indeed, as WAs, RIAs are required to be developed and delivered in a very short time and after that to be updated and evolved even faster. Thus developers usually are focused on the RIA implementation and devote few time and attention to the design of the application, by negatively affecting the quality of the resulting application.

In this scenario, a model driven approach allowing the automatic transformation of the defined design models into a runnable prototype of the application can move the focus of developers from implementation to design issues, improving the final quality of the resulting RIA and facilitating its maintenance and evolution. Such an approach allows developers to validate and incrementally refine the requirements and the design model of the application against a working prototype of it. Moreover, they can regenerate the prototype as many times as needed before starting the actual implementation of the application, or refactor the obtained prototype to develop the final version of the application. Such a model-driven fast prototyping approach adopted in the process of development of a RIA can significantly reduce the risk of rework during or after the implementation phase, and improve the overall quality of the developed WA, as it will rely on a verified and validated design.

According to Model-Driven Engineering (MDE) principles [1][2][17], software development is focused on the modeling of the application structure, behavior and requirements using formal modeling languages, and on the usage of transformation engines and generators to produce artifacts at a lower level of abstraction and a higher level of detail, and source code at the final stage. The aim is to increase productivity (by reusing standardized models), simplify the design process (by using recurring design patterns), and promoting communication between individuals and teams working on the system.

A high degree of automation in model driven software development makes more efficient the entire software lifecycle,
Fig. 1. The UWA MDD Generation Process

and makes higher the overall quality of the resulting software products. In order to adopt an MDE approach two main elements are required: (i) the definition of a meta-model of the design model to use in developing the RIA along with a modeling language standard (such as the Object Management Group’s (OMG) Meta Object Facility (MOF) [14]) allowing to represent the design elements in a standardized and formal way, and (ii) a transformation technology (such as Atlas Transformations Language - ATL, or Query View Transformation - QVT [14], and Xpand) to generate output models, including source code.

In this paper we propose a model-driven fast prototyping approach for RIAs which exploits methods and technologies proper of MDE domain implemented upon the Eclipse Modeling Project [21].

The approach is based on the definition of a meta-model that enables designing a RIA at a conceptual (or domain) level using the Ubiquitous Web Application (UWA) design framework [22]. It provides a complete design methodology and a set of models and modeling tools to specify the design of intensive ubiquitous (i.e., multi-channel, multi-user and generally context-aware) WAs. UWA allows to specify the design of a WA by means of three main models: the Information Model (a.k.a., content or domain model), the Navigation Model, and the Presentation Model.

ATL\(^1\) transformation rules are used to automatically transform the UWA conceptual model of the WA into the MVC-RIA design model. Xpand templates are used to generate from this model the source code of a ready-to-deploy prototype of the application. Currently the approach adopts the RichFaces\(^2\) Ajax framework as technology for the generated application prototype and provides tool support both for the design phase and the code generation phase.

The proposed fast prototyping approach can be reimplemented for metamodels and/or target technologies different from those considered in this paper.

The paper is organized as follows. Section II describes the proposed approach, the process and the technologies used to develop it, and the supporting tools. Section III describes the use of the approach to design and generate the prototype of an e-commerce RIA. Finally, Section IV discusses some related work, and Section V provides some conclusive remarks and future work.

II. APPROACH

The proposed model-driven approach allows to develop a RIA prototype from a set of MOF models representing the WA from various perspective (i.e., information, navigation and presentation).

A. User and Design Centered Metamodels

The approach models external requirements using the UWA design framework [22]. It provides a complete design methodology and a set of models and modeling tools to specify the design of intensive ubiquitous (i.e., multi-channel, multi-user and generally context-aware) WAs. UWA allows to specify the design of a WA by means of three main models: the Information Model (a.k.a., content or domain model), the Navigation Model, and the Presentation Model.\(^3\)

The UWA Information model describes the contents of an application in terms of base information classes (Entities), their structure (Components and Slots) and their relationships (Semantic Associations). The Access Structures model defines subsets (Collections) of the application contents, each based on a selection criterion derived from a specific information access user goal. The UWA navigation model assembles elementary information elements (slots from one or more entities, association centers and collection centers) into reusable units of consumption (Navigation Nodes) and defines navigation contexts (Navigation Clusters) by grouping nodes and defining navigation paths through them (using Navigation Links). The UWA presentation model specifies how the application is organized in terms of pages, which are the components of each page (Sections and Publishing Units), and which node is published in each publishing unit. An excerpt of the UWA

\(^1\)http://www.eclipse.org/atl

\(^2\)http://richfaces.jboss.org/

\(^3\)Additional models are included in UWA, such as: the Transaction Model, which models the business processes the application is intended to support; the Operation Model, which is used to specify the elementary operations the application will provide to its users; the Customization Model, which specifies, by means of customization rules, how the application will adapt to different usage contexts.
metamodel, including a portion of the Information, Navigation and Publishing models is reported in Figure 2.

From a design perspective our approach is based upon the definition of a metamodel suitable for representing a generic WA adopting the MVC pattern at an architectural level. The presentation level is built on the View meta-class representing an abstract View made up of VisualComponents modeling a functional-complete RIA widget set. This metamodel can be used to drive the generation of the application source code for some target platforms that adopt MVC and Ajax as architectural patterns (e.g., the RichFaces implementation platform in the case of this paper). This metamodel fulfills two goals: first, it provides developers with precise specifications guiding the modeling of the WA during high-level design tasks by means of the graphical editor; second, it enables to link the model of the WA to the software artifacts implementing it for the target platform.

Figure 3 shows a simplified version of the metamodel. The model is based on three packages:

- the MModel package, representing the domain or business layer of the WA, that is centered on the ModelClass element made up of Methods and Attributes;
- the VModel package, containing meta-classes defined to represent Views and concepts related to them (like VisualComponents and NavigationControlValue objects used to organize navigation among views themselves);
- the CModel package, with meta-classes used to represent the control part of the MVC pattern.

The MModel package contains the structural elements of the domain model of the application, while the other two (VModel and CModel) represent the presentation layer. The latter describes the structure and content of pages in terms of visual/behavioral elements and the navigation between Views through the Controllers objects. Associations between Visual and Behavioral elements of the View with Link and Action elements model the desired navigation of the WA.

In order to support asynchronous Ajax requests the Event meta-class was introduced in the MVC metamodel. The way events are handled in Ajax applications is one of the fundamental differences between Ajax and traditional web applications. In the context of Ajax a whole refresh of the page is not required as a consequence of an event. Hence components which are interested in an event can be registered as listeners and be notified when the event occurs. The Event meta-class is designed to capture these information for VisualComponents contained in a view. Each VisualComponent contains zero or more events which specify: (i) the UI widget that fired the event, (ii) the UI widgets to be displayed as a result, and (iii) the list of elements registered as listeners of the event.

RichFaces views are actually very similar to a component-tree, so the MVC-RIA model has a clear mapping to the generated server-side code components. RichFaces uses file inclusions to form a tree of .xhtml files which represent the skeleton of the UI of the web application. This concept is very similar to XML-based UI descriptors (such as JavaFX or Mozilla XUL) that can be created by the templates while iterating through the model elements.

B. The UWA MDD Generation Process

Figure 1 depicts the overall WA generation process starting from the UWA conceptual model of the application down to source code and other implementation artifacts. The process starts with the Conceptual Modeling step that, using a graphical editor, allows developers to specify and refine a set of model instances that are expressed using the UWA methodology. When developers are confident that their models correctly represent the user-centered requirements of the WA, they apply ATL model-to-model automatic transformations (between the UWA and the MVC-RIA metamodels) in order to generate a first MVC-RIA design model. Figure 4 reports an example ATL transformation rule that maps UWA Entities to MVC Model classes. This design model can be customized and refined using a graphical editor supporting the MVC-RIA metamodel in order to better satisfy requirements and improve the application presentation layer.

When this design refinement step is completed developers can execute a Xpand model-to-code transformation step to automatically generate the final WA artifacts to be deployed on the target environment. In this step, developers can also customize configuration (e.g., data persistence, IP addresses, exposed end-points) and override the generation of specific items by decorating existing transformation templates before the actual generation takes place. Figure 5 reports an excerpt of the JavaCode Xpand template for the generation of a Java class from the ModelClasses that represent a domain model entity.
III. FROM THE CONCEPTUAL DESIGN TO A RUNNING RIA PROTOTYPE - THE E-MARKET CASE STUDY

To validate the proposed approach we used it to design and implement an e-commerce RIA. We will refer to this as the “e-Market” application in the following. The “e-Market” application allows to manage a catalogue of products with their associated vendors, manufacturers, categories, special offers, and navigation links among them.

The approach was applied by following the three-steps process described in Figure 1. First we used the UWA Graphical Model Editor to define the conceptual model of the application, according to the UWA design methodology. This step produced a UWA model consisting of Entities (product, manufacturer, seller, store, customer, product review) and Entity Slots (e.g., for the entity product, name, image, etc.), Entity Associations (e.g., manufacturer-produces-product), Access Structures (e.g., collection of products by category, collection of product on sale), Navigation Nodes and Clusters, Pages, Page Sections and Page Units. The second step was the automatic generation of the logical model of the application. This was accomplished by executing the defined ATL transformation rules over the UWA conceptual model of the application. The result was a design model complying with the MVC pattern and consisting of model classes, presentation views, relations between views and model classes through controller objects, and navigation paths. Generated views were then further edited using the MVC Graphical Model Editor to choose the best widget for presenting specific contents and enabling user interaction. Finally, we provided the resulting design model as input to the code generation tool wizard which produced the source code and all the other files representing the implementation of the e-Market application prototype.

Figure 6 shows an excerpt of the UWA model that represents the conceptual design of the application. In particular, we can distinguish in it a portion of the Information model of the application, including the entities Store, Seller and Product characterized with the <<Entity>> stereotype, and some of the semantic associations and collections as-
The complete UWA Information model of the application also includes the entities Product, Manufacturer, Customer, ProductReview and collection and associations between them. The overall UWA conceptual model of e-Market also includes the UWA Navigation model and the UWA Publishing model, not reported here due to space reasons.

Figure 7 shows an excerpt of the MVC design model automatically generated for the e-Market application. In the left part of the diagram we can see the classes characterized with the <<ModelClass>> stereotype and representing classes belonging to the Model component of the MVC architecture. In particular the list of the generated model classes are: IME_Product, IME_Manufacturer, IME_Seller, IME_Store, IME_Customer, IME_ProductReview, IMA_Product_Associations, IMA_Manufacturer_Associations, IMA_Seller_Associations, IMA_Store_Associations, IMA_Customer_Associations, IMA_ProductReview_Associations, IMC_Product_Collections, IMC_Store_Collections, IMC_Manufacturer_Collections.

The first 6 classes represent domain classes for the designed application (product, manufacturer, seller, store, customer and product review) with their attributes and methods. The latter 9 classes represent utility classes which implement methods to recover collections of instances of the first 6 classes and associations among them. The diagram in Figure 7, on the right side, also shows some of the classes of the View component of the generated MVC design model characterized by the <<View>> and <<SubView>> stereotypes. In particular the PMP_Customer view and some of its subviews are shown; for one of the subviews, the diagram also shows the included VisualComponents (user interface widgets) and the name of the model class which they are associated to, through the Controller component of the MVC architecture.

A. The Generated e-Market RIA Prototype

The e-Market RIA prototype was generated by the code generator tool providing the MVC design model as described in the previous section. The generated prototype includes all the software artifacts (HTML/JS resources, Java source code, SQL scripts, and project metadata) required to build and deploy the e-Market application on a Tomcat 7.0 application server using a MySQL database for data persistency. The code generator also takes care of automatically copying into the prototype generated Web project the required libraries, such as JDBC drivers, JSF implementation library and tag libraries.
Figure 8 shows the e-Market RIA prototype loaded into the Eclipse IDE as a Web Dynamic Module project. In particular, the Eclipse package explorer view (on the left side in the figure) shows the list of code artifacts produced for the application, while the editor view (main window) shows the source code of the JSP implementing the homepage of the WA (PMP_Home).

Figure 9 and 10 show, respectively, the screenshots of the homepage and the page of a product of the e-Market prototype. The homepage includes a navigation menu and shows a table.
with the list of products for sale which includes, for each product: links to the store selling the product; the manufacturer that produces it; and the page presenting the product details. The table is built using the RichFaces table widget which enables ordering the items by any of the columns data and adapting columns width according to the user preferences. The product page shows all the details of a product and uses an RichFaces Image widget to show a picture of the product which can be zoomed and pane with no need to refresh the Web page. This page also provides links to pages presenting the manufacturer and the store that sells the product. All the pages of the e-Market prototype, their content, structure, and navigation links, derive from the UWA conceptual design defined for the application, while the layout and adopted UI widgets derive from design choices introduced at the MVC design level.

### IV. RELATED WORK

In the Web Engineering context, MDE principles are being used to successfully address the development and evolution of WAs [12].

In particular, Model-Driven Engineering paradigm has been applied successfully by a number of web engineering methods, namely UWE, OO-H, OOHMDA, UWE4JSF, and WebML [9], [16], [4], [3], [18], [11]. These methods use models to separate the platform-independent model (PIM) design of web systems from the platform-dependent (PSM) implementations as much as possible. Usually, they include supporting
development environments for code generation from model specifications, either fully or partially automated.

Similar to our approach, UWE, and OOHDMDA adopt an MDD process that follows MDA principles for the models. WebML differs from our and other considered approaches in that it is MDD but not MDA. Similar to our approach, WebML uses MVC as architectural pattern for its PIMs.

In [5] the authors present a conceptual framework to model context-aware, multichannel Web applications and show how high-level modeling constructs can drive the application development process through automatic code generation. WebML is used to model the concepts of the Web application to be developed.

In [10] a pattern approach for the model-driven development of RIAs to reduce design efforts is proposed. State machines are used for the representation of the patterns. The models of the RIA patterns they specify can be embedded in almost all existing methodologies, and the UML-based Web Engineering (UWE) methodology is used in the paper to demonstrate the validity of the proposed approach.

The paper by Melia et al. [8] presents the OOH4RIA model-driven development process, extending the OOH methodology, to reduce the effort and accelerate its development. The OOH methodology is used to define the domain and navigation models to generate a CRUD server side of the RIA. The same OOH models are also the starting point of model-to-model transformations defining the skeleton of the presentation and orchestration models of the client side of RIA. In [19] the authors propose a model-driven approach based on a RIA Metamodel to support the definition of the User Interface as a combination of widgets from a selected RIA technology and the specification of the UI interaction as a consequence of the events produced by the user. The OOWS method is exploited to apply the proposed approach.

Finally, in [13] the HyperDe environment is presented: it supports the rapid prototyping of Web applications by combining the Model Driven Development approach with the use of Domain Specific Languages. This combination allows the designer/developer to write code by directly manipulating the models that specify the application. HyperDe shares with our approach the use of a meta-model and of the MVC pattern to design a Web application, while it uses different technologies to implement the application to deploy.

The main difference between the proposed approach and the considered related work is that they enable different technologies to be used for the implementation of the PSMs. Our choice of adopting MVC as architecture for the PIM logical model and JSF for the PSM guarantees the availability of a wide range of open-source and commercial technology frameworks to choose from for the different platforms, such as J2EE, .Net and PHP.

V. CONCLUSIONS

In this paper we have presented an approach for the model-driven fast prototyping of Web applications developed using Eclipse technologies and frameworks such as EMF, GMF and Xpand2. The approach consists of a two-steps process, modeling-generate, and is accompanied by two supporting tools: a modeling tool for defining the design of the application by adopting the Model-View-Controller architectural design pattern, and a generator tool that transforms the defined design model into a “ready to run” prototype of the application. The case study has shown that the approach is feasible, valid and the supporting tools to work properly. The code generation phase is completely automatic and produces as output a “ready to run” application prototype, thus a designer using our approach can go through different “modeling-generation-validation” cycles easily and effortlessly. The fast and automatic generation of a fully functioning prototype of the designed application makes it possible to verify and validate the design itself and to undertake a design refinement process effortlessly if required. The generated prototype can also be reused to build the final version of the application which takes advantage of the MVC design pattern and the AJAX technologies. However, the process and the technologies adopted to implement the proposed approach can be (re)used to develop the fast prototyping approach for a design model different from MVC and/or a different target technology platform. The approach can be combined with any conceptual design method for Web applications to realize a fast prototyping approach for this method, not only with the UWA design methodology we used in this paper.

More ongoing work is intended to improve the approach to support other interesting target platforms (e.g., for mobile
devices), as well as to improve persistence management by supporting JPA and incremental updates for SQL schema scripts. Future work will be devoted also to provide a better support for designing the layout and look & feel of the user interface of the application, and to support round trip engineering of generated artifacts, so that manual customizations are taken into account and merged by the generator. This is needed to support manual customization of the generated source code.

REFERENCES


