Facing Architectural and Technological Variability of Rich Internet Applications[1]

Santiago Meliá, Jaime Gómez
IWAD Group, University of Alicante
c/San Vicente S/N
Alicante, Spain
{santi, jgomez}@dlsi.ua.es

Sandy Pérez, Oscar Díaz
ONEKIN Group, University of Basque Country
Plaza Manuel de Lardizabal 1
San Sebastián, Spain
{sandy.perez, oscar.diaz}@ ehu.es

ABSTRACT

The advent of Rich Internet Applications has involved an authentic technological revolution providing Web users with advanced requirements similar to desktop applications. At the same time, RIAs have multiplied the possible architectural and technological alternatives complicating development and increasing risks. The real challenge is to select the right alternatives among the existing RIA variability, thus creating an optimal solution able to satisfy most user requirements. To face this challenge, for the RIA development process, we propose an extended OOH4RIA approach to introduce architectural and technological aspects at the design phase, to propagate these decisions to the rest of concerns and to provide a closer match between the modeled system and the final implementation.

Keywords: Rich Internet Applications, Model-Driven Engineering, Software Architecture, Feature modeling.

1. INTRODUCTION

Rich Internet Applications (RIAs) are breaking through the Internet market, offering better responsiveness and a more extended user experience than traditional Web applications. RIAs are client/server applications that are at the convergence of two competing development cultures: desktop and Web applications. They provide most of the deployment and maintainability benefits of Web applications, while supporting a much richer client User Interface (UI). Moreover, RIAs introduce new architectural characteristics in the field of traditional Web applications, e.g. a stateful UI with connected and disconnected states, an intelligent client/server communication with asynchronous requests. Thus, RIA developers must make many architectural decisions and balance the trade-offs. The real challenge lies in choosing the right alternatives among the RIA architectural and technological variability so as to find an optimal solution that satisfies all the client requirements.

To this end, we propose to introduce architectural and technological concerns at the design phase of RIA development. Thus, we will obtain the advantages of increasing the robustness of a system through the application patterns as well as providing a closer match between the modeled system and the final implementation.

In order to define this extension, we have adopted a Generative Software Development (GSD) [1], which proposes a feature-oriented approach. Specifically, GSD uses the feature model as the starting point of the architecture concern that represents the solution-space viewpoint of the system.

With this aim, we present an extension of an existing RIA model-driven development process called OOH4RIA [2] whose highlights include: (1) a specific RIA feature model to represent architectural and technological variability, (2) a RIA component-based architectural model to define high-level architectures and (3) Model2Text transformations are driven by the architectural model to obtain the final implementation.

The paper is organized along these three artifacts. Next section starts by providing the main rationales for this approach.

2. IMPROVING THE RIA DEVELOPMENT PROCESS

RIA methodologies are relatively new and do not yet cover all design concerns usually encountered in state-of-the-art software engineering (e.g. RUX [3], WebML [4], OOHDM [5], UWE [6] and OOH4RIA [2]). These approaches have proven successful for functional concerns such as domain, navigation and presentation concerns. Broadly, they propose a set of RIA-specific abstractions (i.e. using Domain-Specific Languages (DSLs)) to be used by analysts to specify their needs. Following the Czarnecki [1] classification, these models can be situated on the problem-space of RIAs.

Unfortunately, architectural and technological aspects tend to be overlooked. Consequently, these RIA proposals have a gap between the problem concepts that capture their models and how these concepts end up being implemented through components or RIA frameworks. For this reason, these methods have to realize a set of assumptions selecting predefined architectures and technologies which are often not the most appropriate w.r.t. the solution sought by the customer. Moreover, this characteristic is especially

[1] This work is co-supported by the Spanish Ministry of Education, and the European Social Fund under contracts TIN2008-06507 (MODELINE) and TIN2007-67078 (ESPIA). Perez enjoys a doctoral grant from the Basque Government under the "Researchers Training Program".

important in methodologies that provide a code generation environment (i.e. RUX, WebML and OOH4RIA). The inclusion of the solution-space abstractions would therefore decrease the set of predefined architectural decisions that are usually taken in generating the code in such environments. The first steps that change this situation, is to identify different design alternatives in RIA layers as the ones presented in [7] (Data, Communication, Presentation and Business Logic). They propose guidelines but they did not introduce these architectural or technological aspects to be explicitly defined in their development process. Recently, [8] introduces the specification of a set of architecture aspects in RIAs (such as data distribution, persistence and communication) extending the BPMN model. However, this model blends both problem and solution aspects where no architectural configuration is provided.

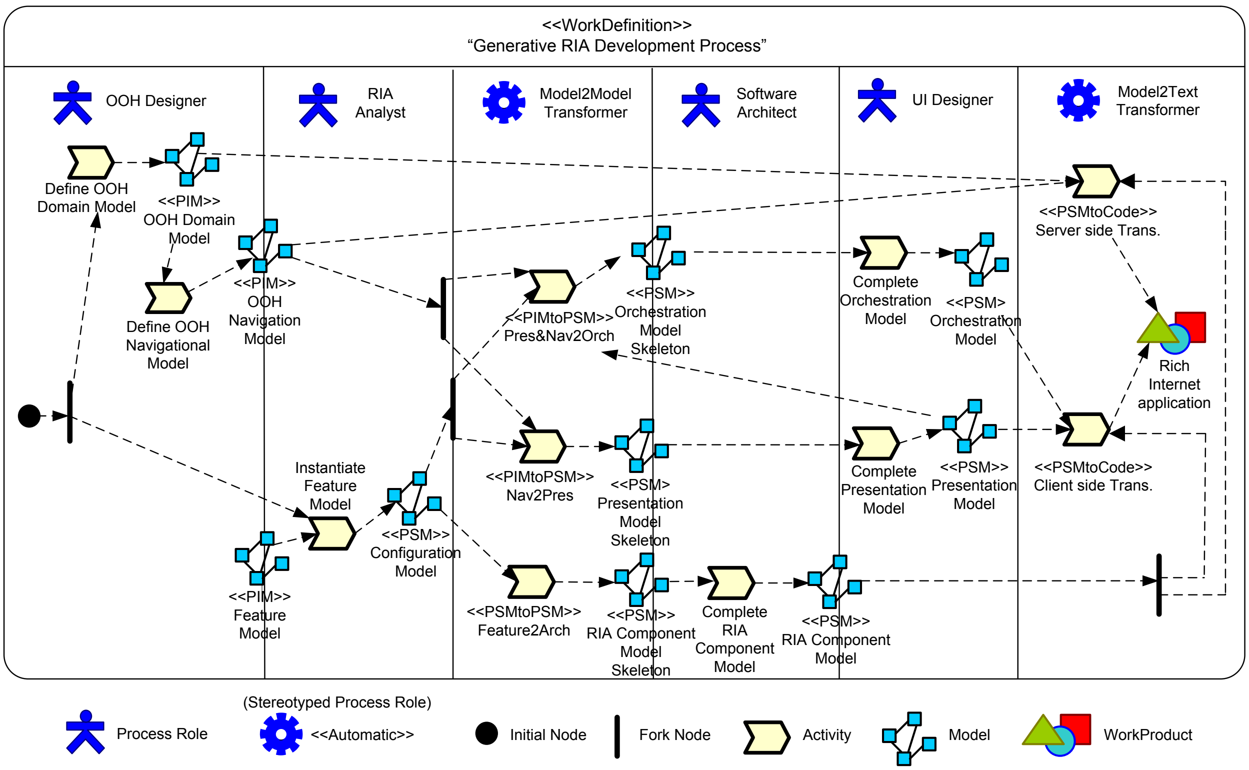

Figure 1. The OOH4RIA Process: A Generative RIA Development Process

Along the lines of the Generative Software Development (GSD) approach defined by Czarnecki, we introduce a feature-oriented proposal to cover the gap between the problem-space and the solution-space. GSD provides the means for introducing the feature model as a mechanism to represent the characteristics more relevant of a system family into a model-driven development process. The feature model captures the general system requirements and permits to form the architecture that represents the solution-space in the RIA process. And a RIA Component Model that establishes an implementation-oriented abstraction that can be instantiated to create RIA implementations. This approach is borne out for the OOH4RIA methodology. Fig. 1 depicts the enhanced OOH4RIA process using a model-driven specific SPEM notation (explained in the fig.1 legend). We have extended the SPEM profile by introducing a ProcessRole stereotype to represent automatic model transformation engines; a set of Model stereotypes to represent different MDA model types such as CIM, PIM and PSM; finally, a set of stereotypes of the metaclass Activity to represent different Model2Model MDA transformations (e.g. PIMtoPSM, PSMtoPSM, etc.) and Model2Text MDA transformations (e.g.. PIMtoCode, PSMtoCode, etc.).

First, the OOH Designer defines both the Domain Model and the Navigation Model. At this point, the process requires the RIA Analyst to instantiate the Feature Model selecting different architectural and technological decisions along the Feature Model. For instance, he can choose the architectural layers of the RIA (2 layers, 3 layers, N layers, etc.), the RIA UI framework (e.g. Google Web Toolkit, Flash, Silverlight, etc.) and so on (more in section 3).

Once the selected features establish a valid configuration called Configuration Model, a semiautomatic phase starts where a set of QVT Model2Model transformations obtain the model skeletons. For the Presentation and Orchestration Model details can be found in [2]. Additionally, the Configuration Model also guides the generation of the RIA Component Model Skeleton through the Feature2Arch transformation. This skeleton provides a first blueprint which is then completed by the Software Architect by defining the relationships between components and their specific architectural properties (e.g. a *UIEntityData* component could support *Work-Offline*) (section 4). Finally, a Model2Text transformation will deliver a RIA implementation (section 5)

Next sections delve into the details. The SUMA Content management system (SUMA CMS) is used as the running example, a real project developed by OOH4RIA for the Alicante regional tax agency called SUMA. The SUMA CMS is an intranet RIA which permits employees with no knowledge on Web programming to manage different multimedia content (i.e. documents, files, images, videos, etc.) of SUMA Web applications.

3. THE RIA FEATURE MODEL

Most Model-Driven Engineering (MDE) efforts consider functional variability, but overlook architectural alternatives. However, the ability of representing variability in the solution space i.e. architecture and technology, allows to reduce the risks and improves the mapping of problem domain concepts (classes, attributes, operations, etc.) into solution domain artifacts (components, frameworks, etc.). We use a Feature Model to capture a subset of architectural and technological variations in RIA. At this point, it is worthy highlighting that a Feature Model does not try to represent all the variability of a system family, it only describes a set of valid configurations of the most relevant features (i.e. features that are important to enter a new market, to incur a technological risk and so forth) and each configuration describes a single product from the system family. Thus, a valid configuration of the Feature Model parameterizes the model transformations to output an architecture skeleton model compliant with the selected variants.

Following the GSD approach, this Feature Model is designed during the definition of the OOH4RIA process (i.e. during domain engineering). Next, it is "instantiated" during process execution to cater for the application at hand (i.e. application engineering). To align the feature modeling with the model-driven approach, the Feature Model is formalized by a MOF metamodel and a valid configuration for a specific application is called Configuration Model.

Feature Specification. One of the foremost RIA novelties is that of providing an interactive UI with *Business Logic* and *Work-Offline* properties. In this paper, we focus on the RIA client layer and its architectural artifacts. Table 1 presents the features that we consider in the RIA client design along the work described in [7] [9]. This table also indicates the architectural artifact being affected.

In order to represent the OOH4RIA Feature Model, we have used the Czarnecki [1] extended notation because it allows us to define the feature's attributes and establish a precise cardinality in their relationships as well as a new group relationship type called or-inclusive. Fig. 2 depicts the OOH4RIA Feature Model with a focus on the RIA client layer and the client/server communication. For instance, the *Event-Handling* feature allows to establish an event-based choreography between UI components, so it must be related to the component that deals with UI interactions, i.e. the UIProcessComponent.

Table 1. Simplified Classification of Client-side RIA Features

RIA Client Features	Description	Architectural Artifact
Work-offline	Possibility to work while disconnected by downloading the business logic and the data on the client side. This feature requires Business Logic and a storage mechanism.	UIEntityData
Storage	RIAs provide new client side storage facilities handling the data that come from the server. This storage could be persistent or volatile.	UIEntityData
Caching	RIA client has the ability of keeping the server information during a period of time improving application performance and UI responsiveness. Caching has a *fetching* attribute which supports inmediate or lazy values.	UIEntityData
Business Logic	RIA client has an improved process capability allowing to realize complex processes. This feature has an attribute *location* that determines whether it is a client, mixed, or server business logic.	UIEntityData
Event-Handling	RIAs have an event-based choreography between different UI components that could be synchronized with centralized event bus or with decentralized observer pattern.	UIProcess-Component
Validation	RIA could contain validation rules for user input as well as for business rules on the client.	UIComponent
Templating	Possibility to support the creation of views and the presentation layout at runtime.	UIComponent
Platform	It determines which the platform used for implementing the client layer is.	Client Layer

Feature Selection During the OOH4RIA process, the RIA Analyst must select a subset of features from the OOH4RIA Feature Model (a.k.a the Configuration Model). Fig. 2 presents the Configuration Model for the SUMA CMS. At the top, a set of architectural features establish the different layers of a RIA (client and server). These features are in turn split into alternatives on the components that can make up these layers, (e.g. the client layer could contain three component types like *UIComponent*, *UIProcessComponent* and *UIEntityData*) or to architectural features that can affect the whole layer (e.g. the RIA *Platform*). At this point, component-based features split into distinct design and technological alternatives (i.e. *Templating, Caching, Work-*

Offline, Platform, etc.). Table 1 represents the relationship between these RIA client features and the architecture artifact which is in charge of them. For instance, the *Event-Handling* feature allows to establish an event-based choreography between UI components, so it must related to the component that deals with UI interactions, i.e. the *UIProcessComponent*.

This Feature Model is then "instantiated" for the application at hand, e.g. the SUMA CMS. Fig. 2 describes this application in terms of the selected features, namely: *UIComponent* must have a *Templating* mechanism to create the UI but it does not use a *Validation* for input data; *UIProcessComponent* will follow a decentralized event-based *Observer* pattern; the *UIEntityData* will only require *Storage* of type *Volatile*; the *Platform* will be *RichFaces*; *Communication* will follow a *JSON* format to send messages from a RichFaces client to a server implemented with the *JSFController* provided by the JSF framework; finally, the Communication styles will be *RPC* and asynchronous *Message-based* using AJAX.

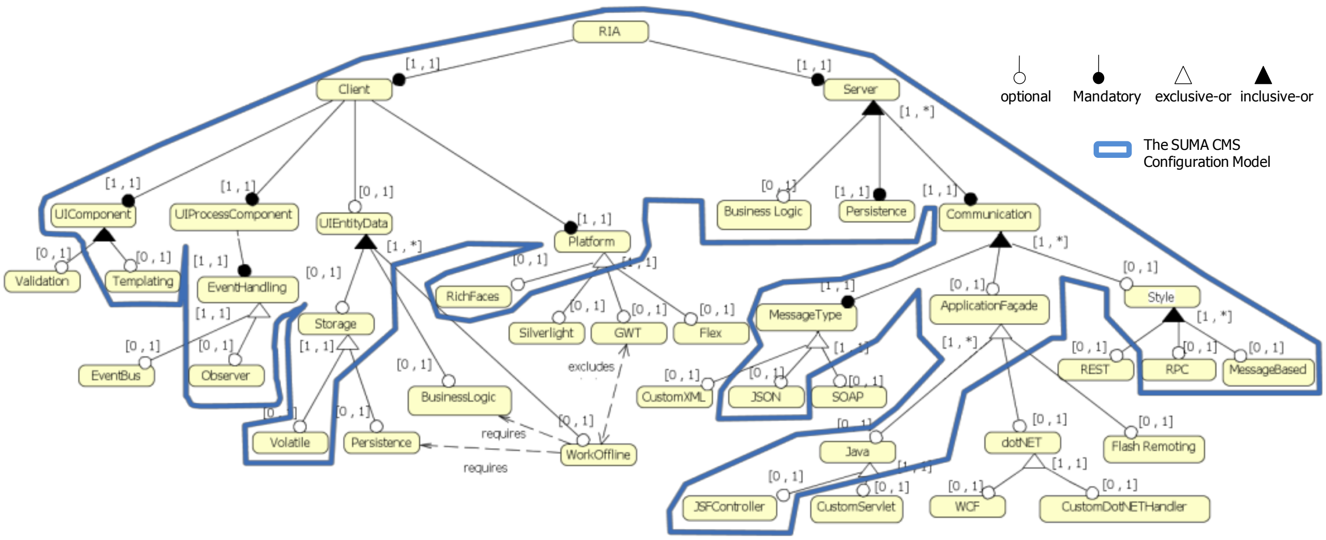

Figure 2. The Simplified OOH4RIA Feature Model and the SUMA CMS Configuration Model

Feature-driven transformations. Once the Feature Model has been instantiated for SUMA CMS, the OOH4RIA process executes the Feature2Arch transformation, which queries the Configuration Model in order to generate an architectural model skeleton. Next section presents this architectural model called RIA Component Model.

4. THE RIA COMPONENT MODEL

The presence (or absence) of a meaningful architecture is an essential predictor of the success of an Internet Information System. First, the creation of a stable architecture helps cut the highest risks out of the project. Second, the presence of a stable architecture provides the basis upon which the system may be continuously developed with minimal scrap and rework.

In this way, our work introduces an explicit representation of RIA architecture through the definition of a specific architectural style called RIA Component Model, based on the study of the RIA family and the knowledge acquired from a previous work: WebSA [10] in the Web architectural domain. The RIA Component Model (RIA-CM) is a component-based architectural style that represents a structural view of the RIA application. This model defines a topology of components, each one representing the role or the task performed by one or more common components identified in the RIA family (e.g. *UIComponent*, *UIEntityData*, *UIProcessComponent*, etc.). In this way, RIA-CM allows us to specify an architectural configuration without knowing anything about the problem domain. In order to represent its architectural style, the RIA-CM is defined as a profile of the UML Component Model, and includes components and properties of the RIA domain. Each component has two mandatory attributes: a concern indicating the functional concern which gathers the problem-space information to obtain the final software components, and the layer where they will be located. For the sake of accuracy, these components also provide the specific properties of each component type (e.g. a *UIComponent* specifies if it contains a client validation) thus covering the possible component's variability.

RIA-CM is not defined from scratch, but a first RIA-CM skeleton is obtained from the execution of the Feature2Arch transformation. This transformation maps the checked features of the Configuration model into the RIA-CM components. This skeleton includes isolated components characterized along their technological features.

However, this skeleton is not complete. The Software Architect should provide (1) the relationship between components and (2) architectural attributes (such as design patterns, distribution, etc.).

Specifically, the Feature2Arch transformation looks through the Configuration Model only querying the checked features. Then, it converts the checked Feature elements that represent a specific component (e.g. *UIComponent*) into a component in the RIA-CM. After that, the transformation looks through the checked technological subfeatures (e.g. *Templating* feature) in order to establish the values of the RIA component (e.g. the *Templating* property of the *UIComponent* is set as true).

Once the Feature2Arch transformation is executed, we obtain a RIA-CM skeleton that only contains the components with their technological attributes. However, there is no relationship between components and no architectural attributes are provided. At this point, the Software Architect must complete the RIA-CM by introducing these architectural attributes into the components and interconnecting them. Fig. 3 shows a simplified view of the RIA-CM of the SUMA CMS application.

The front-end part of this model shows a *UIComponent* called Widgets which provides a rich interface. The *UIComponent* properties, obtained from the Configuration Model, indicate that each *UIComponent* is implemented with the RichFaces framework using templating mechanism to create different views. Each *UIComponent* also has a link to a set of *UIEntityData* components called Model in order to access the state of real world entities on the client side.

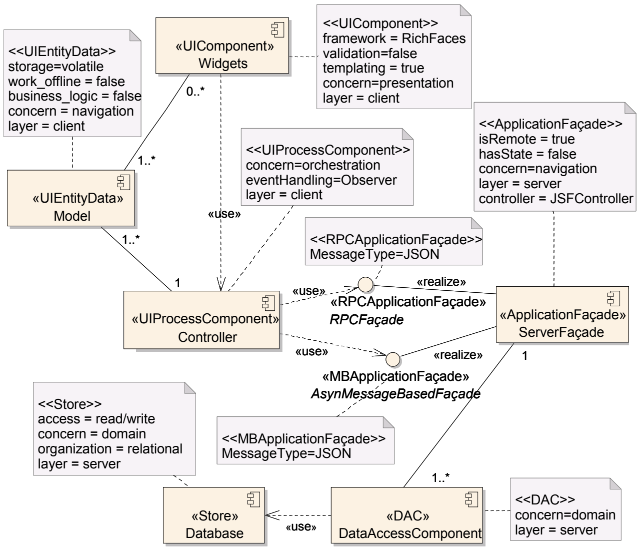

Figure 3. RIA Component Model of the SUMA CMS (simplified view)

Specifically, the Model properties indicate that only navigation data are temporarily kept in the client layer and no work-offline is allowed. Following a Model-View-Controller pattern on the client side, we define a *UIProcessComponent* called Controller that synchronizes user interactions obtaining information from the Orchestration Model. The *UIProcessComponent* has a relationship with a set of *UIEntityData* components so that they can be updated when a service modifies the model information. Additionally, it also exhibits two `<<use>>` dependencies with the RPC and message-based interfaces of the *ApplicationFaçade* so that services of the RIA server side can be invoked.

5. ARCHITECTURE-DRIVEN MODEL2TEXT TRANSFORMATIONS

The RIA-CM represents a high-level architecture, focused on organizing components that support specific functionalities. The RIA-CM proposes an implementation-based abstraction to locate the content of different functional DSLs into the software components that represent the final implementation. We refer to this organization of functionality as grouping components in "areas of concern".

Both, the separation of functional concerns and the component type classification, permit to establish a transformation policy based on the following best practices. First, Model2Text transformations are based on the concatenation of a set of transformation rules, each one being associated with a specific component type. This improves modularity and traceability. Second, the collection of final software components derived from each RIA-CM component type is inferred from the elements of the related functional concern. That is, each component type has one or more transformation rules that only query the model that corresponds to its concern. For instance, the transformation rules of the *UIComponent* type query the Presentation Model generating for each Widget a *UIComponent* and introducing the static properties (i.e. position, x, y, width, etc.) into each *UIComponent*. Third, the definition of an inheritance hierarchy of transformation rules for each component type which is driven by architectural and technological variability. This hierarchy permits to separate and reuse the platform independent side in charge of querying the functional concerns from the rule side that generates the specific technological code. This separation also improves the maintainability of transformation rules because the platform independent side keeps the root rules more stable while reducing, with the help of polymorphism, the impact of introducing new technologies. For example, we define a rule for the *UIEntityData* component that queries the Navigational Model in order to generate a *UIEntityData* for each NavigationalClass element and to obtain its functional properties. At the same time, without changing previous rules, we can define a set of *UIEntityData* rules specialized in supported frameworks, that is, rules that cater for the specificities of RichFaces, Flex or .NET. Fig. 4 presents an example of three Xpand[2] transformation rules that explain the architecture-driven transformation policy and generate the spatial representation of UIComponent elements. The Xpand has a straightforward notation that defines each rule with the tag `<<DEFINE>>` and indicates the name and the queried metamodel element that establishes its execution. The rule ends with the `<<ENDDEFINE>>` tag.

The top rule called *GeneratingRIACM* is in charge of starting the execution of the Client Side transformation when receiving the ComponentModel metaclass as input metamodel element. This rule looks through all the RIA-CM components and invokes the *GeneratingComponent* rule using the `<<EXPAND>>` tag where `FOREACH this.components` indicates that this rule is invoked once for each component. At this point, we apply polymorphism defining a set of rules with the same name, each of them having a different component type. In our example, *GeneratingComponent* rule is executed when the component has a *UIComponent* type and generates a specific code for this component type. As previously stated, the *UIComponent* type is linked to the presentation concern, for which we have defined a global variable called PM that permits the rules to access the Presentation metamodel. A RIA application, as commented in

[2] Xpand. Model-to-Text project http://wiki.eclipse.org/Xpand

[2] can be very complex as the *UIComponent* elements are not shown at the same time and with the same appearance. For this reason, the PresentationModel is organized into a set of elements called Screenshot which represents a container that allows the UI Designer to realize a spatial distribution of different Widget elements rendered at a given moment. Thus, the *GeneratingComponent* rule queries all the Screenshots elements invoking the *DefiningScreenshot* for each one of them.

```
<<DEFINE GeneratingRIACM FOR ComponentModel>>
     <<EXPAND GeneratingComponent
               FOREACH this.components>>
<<ENDDEFINE>>

<<DEFINE GeneratingComponent FOR UIComponent>>
     <<EXPAND DefiningScreenshot(this)
               FOREACH ((PresentationModel)GLOBALVAR PM).sshot>>
<<ENDDEFINE>>

<<DEFINE DefiningScreenshot (UIComponent ui) FOR
    RichFacesScreenshot>>
         <<FILE "+name.toFirstUpper()+".xhtml">>
     <html xmlns="http://www.w3.org/1999/xhtml"
        xmlns:f="http://java.sun.com/jsf/core"
        xmlns:h="http://java.sun.com/jsf/html"
        xmlns:a4j="http://richfaces.org/a4j"
        xmlns:rich="http://richfaces.org/rich">
    <head><title><<name.toFirstUpper()>></title></head>
    <body>
    <<FOREACH  this.referredWidgets() AS wgt>>
           <<<wgt.metaType.toString().richFacesType()>>
              style="position: <<wgt.position>>;
               top:<<wgt.posX>>px;
               left:<<wgt.posY>>px;"
              <<EXPAND DefiningSpecificPropertiesWidget FOR wgt>>
           >
          <<EXPAND DefiningContainedWidgets FOREACH wgt.widgets()>>

     <<ENDFOREACH>>
     <<EXPAND DefiningJSFBackingbean FOR this>>
     <body>
<<ENDDEFINE>>
```

Figure 4. Example of Xpand Transformation rules for generating UIComponents

The last rule presents the use of the third best practice where we apply polymorphism to separate platform independent rules from technological specific ones. The example presents a *DefiningScreenshot* rule that receives a RichFacesScreenshot element from the Presentation metamodel. The Presentation Model, as explained in the process section, is defined by the introduction of the technological variability of the Feature Model. According to the Configuration Model (see fig. 2), we have selected the RichFaces framework feature to generate the Screenshots and the UIComponents with the static and behavioral properties. The *DefiningScreenshot* rule creates a XHTML file for each Screenshot element which contains the code for defining the spatial representation of the UIComponents of the Screenshot. To do that, it defines a `<<FOREACH>>` loop, which queries the position and the x and y dimensions of each widget and invokes the *DefiningSpecificPropertiesWidgets* to generate their specific properties. Moreover, each widget can in turn contain other widgets which are generated by *DefiningContainedWidgets*. Finally, this rule invokes the *DefiningJSFBackingbean* rule to obtain the Screenshot dynamic behavior implemented by RichFaces.

6. CONCLUSIONS

Current RIA development process is challenged by bridging the gap between the current functional models situated on the problem space and the final implementation. This work introduces architectural and technological concerns to an existing RIA methodology named OOH4RIA. To this end, OOH4RIA blends model-driven and feature-oriented development so that a Feature Model permits an early capture of the RIA variability and the RIA-CM architecture style that establishes a component configuration in the solution space. Moreover, OOH4RIA proposes an architectural-driven Model2Text transformation that promotes the design of best practices solutions, establishes a separation of concern where each component has a specific role. As a result, system cohesion is increased and traceability is ensured between user requirements and the final implementation.

In order to give support to this approach, we are working on the introduction of the feature and architecture models into the OOH4RIA tool. Currently, several RIA projects[3] are being developed (e.g. SUMA CMS, an online MediaPlayer, a RIA Mail, etc.) where this approach is being applied successfully.

7. REFERENCES

[1] Czarnecki K., Helsen S., Eisenecker U.W. Staged configuration through specialization and multilevel configuration of feature models. Software Process: Improvement and Practice, vol.10 (2), pp.143-169, 2005.

[2] Meliá S., Gómez J., Pérez S., Diaz O. A Model-Driven Development for GWT-Based Rich Internet Applications with OOH4RIA. IEEE Conference Proceedings, ICWE'08, pp.13-23, USA, 2008.

[3] Linaje M., Preciado J.C., Sánchez-Figueroa F. Engineering Rich Internet Application User Interfaces over Legacy Web Models. IEEE Internet Computing, vol.11, no.6, pp.53-59, 2007.

[4] Bozzon A., Comai S., Fraternali P., Toffetti G. Conceptual Modeling and Code Generation for Rich Internet Applications. ACM Proceedings, ICWE'06, pp.353-360, USA, 2006.

[5] Urbieta M., Rossi G., Ginzburg J., Schwabe D. Designing the Interface of Rich Internet Applications. LA-Web Conference Proceedings, pp.144-153, 2007.

[6] Koch N., Pigerl M., Zhang G., and Morozova T. Patterns for the Model-based Development of RIAs. LNCS 5648, ICWE'09, pp.283-291, Spain, 2009.

[7] Preciado J.C., Linaje M., Comai S., Sanchez-Figueroa F. Designing Rich Internet Applications with Web Engineering Methodologies. WSE'07 Proceedings, pp.23-30, France, 2007.

[8] Bambrilla M., Preciado J.C., Linaje M., Sánchez-Figueroa F. Business Process-Based Conceptual Design of Rich Internet Applications. IEEE Conference Proceedings, ICWE'08, pp.155-161, USA, 2008.

[9] Microsoft Patterns & Practices. Rich Internet Application Guide, 2009.

[10] Meliá S., Gomez J. The WebSA Approach: Applying Model Driven Engineering to Web Applications. Journal of Web Engineering, vol.5, no.2, pp.121-149, 2006.

[3] OOH4RIA Projects. http://suma2.dlsi.ua.es/ooh4ria/Projects

ABOUT THE AUTHORS

Santiago Meliá is Assistant Professor at the Universidad de Alicante, Spain. His research topics of interest include Rich Internet Applications, Model-Driven Engineering, Generative Software Development and Semantic Web. Most of these topics are part of his Ph.D in Computer Science received at the University of Alicante in 2007. He has published in prestigious journals and conferences (e.g. EJIS, JWE, OOPSLA, ER, EC-MDA, ER, etc.) and regularly serves in the PC of several international conferences and workshops (e.g. SAC, MDWE, WMR, etc.). Currently, he is working in the development of tools and methods in order to be applied in the software industry. A more detailed curriculum vitae and the list of publications can be found at http://www.dlsi.ua.es/~santi.

Mailing address: Universidad de Alicante, Departamento de Lenguajes y Sistemas Informáticos, c/ Carretera San Vicente del Raspeig s/n – 03690, San Vicente del Raspeig, Alicante, Spain; **phone**: 965903400 +2383; **email**: santi@ua.es.

Jaime Gómez is a Professor of Software Engineering at the Computer Science School at the University of Alicante. He received a B.S. in Computer Science from Technical University of Valencia in 1992 and Ph.D. in Computer Science from the University of Alicante in 1999. His current research interests include Web Engineering, Model-Driven Development, Rich Internet Applications Design and Semantic Web. He is the head of Web Engineering, Applications and Developments Research Group at University of Alicante. For more than ten years, his research group has developed methods and tools to facilitate software development in Industry. He is author of more than 50 papers in Journals and International Conferences.

Mailing address: Universidad de Alicante, Departamento de Lenguajes y Sistemas Informáticos, c/ Carretera San Vicente del Raspeig s/n – 03690, San Vicente del Raspeig, Alicante, Spain; **phone**: 965903400 +3309; email: jgomez@dlsi.ua.es.

Sandy Pérez is currently a research assistant at the University of the Basque Country, Spain. His current research interests are in the areas of portlet-based portals, Rich Internet Applications and social networking. He holds a B.S. in Computer Science from the University of the Basque Country.

Mailing address: Facultad de Informática, paseo Manuel Lardizabal 1, 20018 Donostia-San Sebastián, Spain; **phone**: 943018500; **email**: sandy.perez@ehu.es.

Oscar Díaz is full professor at the University of the Basque Country. He obtained the BSc in Computing at the University of the Basque Country, and a PhD by the University of Aberdeen in 1992. His current interests include portlet-based portals, Model-Driven Engineering and Software Product Lines.

Mailing Address: Facultad de Informática, paseo Manuel Lardizabal 1, 20018 Donostia-San Sebastián, Spain; **phone**: 943018500; **email**: oscar.diaz@ehu.es.