Delft University of Technology
Software Engineering Research Group
Technical Report Series

Modelling and Generating Ajax Applications: A Model-Driven Approach

Vahid Gharavi, Ali Mesbah, and Arie van Deursen

Report TUD-SERG-2008-024

TUD-SERG-2008-024

Published, produced and distributed by:

Software Engineering Research Group
Department of Software Technology
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
Mekelweg 4
2628 CD Delft
The Netherlands

ISSN 1872-5392

Software Engineering Research Group Technical Reports:
http://www.se.ewi.tudelft.nl/techreports/

For more information about the Software Engineering Research Group:
http://www.se.ewi.tudelft.nl/

Note: *This paper is a pre-print of:*

Vahid Gharavi, Ali Mesbah, and Arie van Deursen. Modelling and Generating Ajax Applications: A Model-Driven Approach. In 7th International Workshop on Web-Oriented Software Technologies (IWWOST'08), New York, USA, 2008.

Modelling and Generating AJAX Applications: A Model-Driven Approach

Vahid Gharavi
Delft University of Technology
The Netherlands
S.V.VahidGharavi@ewi.tudelft.nl

Ali Mesbah
Delft University of Technology
The Netherlands
A.Mesbah@tudelft.nl

Arie van Deursen
Delft University of Technology
The Netherlands
Arie.vanDeursen@tudelft.nl

Abstract

AJAX is a promising and rapidly evolving approach for building highly interactive web applications. In AJAX, user interface components and the event-based interaction between them form the founding elements, whereas in classic web applications the notions of web pages and hypertext links are central. Therefore modelling AJAX requires a different approach than what the current web modelling tools are providing. In this paper we propose a UML scheme for modelling AJAX user interfaces based on the MDA approach. We adopt ANDROMDA for creating an AJAX cartridge to generate an entire AJAX-based web application with automatic back-end integration. The implementation of this cartridge is a work in progress.

1 Introduction

In the years following the arrival of the Internet, new web-based software technologies and platforms have been emerging in an overwhelming pace. Recently, a broad collection of new trends have appeared under the *Web 2.0* umbrella, changing the classical web interaction significantly.

A prominent enabling technology in *Web 2.0* is AJAX (Asynchronous JavaScript and XML) [5, 10], in which a clever combination of JavaScript and Document Object Model (DOM) manipulation, along with asynchronous delta-communication is used to achieve a high level of user interactivity on the Web [8]. After its inception in 2005, numerous AJAX frameworks and libraries have appeared and the technology has been evolving fast.

Maintaining web applications and keeping them up to date with new technologies are often complex and expensive tasks. Furthermore, the integration of different technologies, from front-end to back-end, seems challenging, yet a necessity in building enterprise web applications. When adopting a new and evolving technology such as AJAX, it is very important to be able to cope with changing environments from an architectural point of view. A framework which is fit for the project today could easily be out featured by a new one tomorrow.

One way to tackle these challenges is by abstracting from the implementations through a Model-Driven Engineering [13] approach, in which the AJAX application is defined in a modelling language and the corresponding web code is automatically generated. This approach enables us to define the application once and generate the same application to different frameworks.

There are different ways of modelling software systems. The Object Management Group (OMG) has devised a number of standards for software development under its Model-Driven Architecture (MDA) approach [12].

In this paper we explore an MDA approach for AJAX web applications. The first step in an MDA approach to application development is modelling; We therefore look into how an AJAX web application can be modelled, while having the ultimate goal of code generation from the models in mind. We propose a UML scheme for modelling AJAX user interfaces, and report on our intended approach of adopting ANDROMDA for creating an AJAX cartridge to generate the corresponding AJAX application code, in ICEFACES, with back-end integration.

The paper is organized as follows. We start out, in Section 2, by sketching the problem statement. In Section 3, we explore the related work on current model-driven approaches for the web. In Section 4 we discuss ANDROMDA. In Section 5, the proposed AJAX meta-model and the generation process are presented, after which Section 6 discusses the findings and open issues. We conclude the paper with a brief summary of our key contributions and suggestions for future work.

2 Problem Statement

AJAX is a technique based on a collection of web technologies which can be used to create highly interactive web

applications. The interactivity aspect is manifested by the fact that refreshing the whole page is not needed for each client server interaction. The user interface can consist of a single web page composed of UI components which can be acted upon and updated independently of other components. This technique has made it possible to port similar versions of many desktop applications to the web, examples of which include office tools such as word processors (Google Docs and Spreadsheets[1]) and e-mail management software (Yahoo! Mail[2]).

AJAX applications can be seen as a hybrid of desktop (e.g., Swing) and web applications, inheriting characteristics from both worlds [10]. Therefore modelling AJAX requires a different approach than what the current web modelling tools are providing. Whereas in classic web applications the notion of web pages and links between them is central, in AJAX, user interface components and the event-based interaction between them form the founding elements.

The advantages of MDA [11] encouraged us to explore the possibilities of creating a tool which can transform a platform independent model (PIM) to a platform specific model (PSM) and subsequently to a single-page AJAX web application. While tools exist for the generation of applications for specific platforms such as J2EE and .NET, we have not encountered any such tool for AJAX. With such a transformation, models used to generate legacy web applications can be used to generate single-page web applications and vice versa. Also the same models can be used to generate code for different AJAX frameworks.

We are interested in an MDA approach for the main structure of the AJAX application, as well as the possibility of integrating with back-end components such as Spring and Hibernate. It should also be possible to model which events affect which components (listeners). For the sake of simplicity, we propose to leave certain layout information such as the precise position of components and styling, out of the abstract models.

3 Related Work

There have already been several developments in the area of MDA for the web. In this section we explore some of the most relevant approaches.

The Web Modeling Language (WebML) [3] 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 [2, 14].

Conallen [4] proposes a UML profile for modelling web applications. This approach is widely referenced as a web modelling scheme. Koch and Kraus [6] propose UWE, a UML profile and notations for modelling the navigation and conceptual aspects of a web application. Both approaches are aimed at classic multi-page web applications.

RUX-model [2, 14] is an MDA approach towards Rich Internet Applications (RIA) based on WebML. In this approach WebML is extended with notations which indicate whether certain data is stored or presented on the client or the server. In the latter stages, the RIA is modelled using the *RUX-model* notation (not UML) and subsequently Flash-based RIA instances (e.g., OpenLaszlo) are generated. According to the authors, also the same models can be used to generate AJAX applications. It appears that only the user interface part is generated by RUX-model since issues such as the back-end or the toolkits employed in the generation of the whole web application are not mentioned. RUX-model is currently unavailable for experimenting.

Another attempt at an MDA approach for RIAs is found in [7]. The approach is based on XML User Interface description languages and XSLT is used as the transformation language between the different levels of abstraction. Again, this approach is oriented towards the User Interface and lacks flexibility in an MDA approach for the whole web application. It also lacks a visual notation and cannot be modelled using existing CASE-tools. This is also true in the case of other XML-based UI languages such as XUL [3], XAML [4] and UIML [1] which do not offer the simplicity of a visual model.

Visser [15] proposes a domain-specific language called WebDSL for developing dynamic web applications with a rich data model. WebDSL applications are translated to classical Java web applications, building on the JSF, Hibernate, and Seam frameworks.

openArchitectureWare[5] (oAW) is currently one of the leading open-source MDA generator frameworks. It is very extensible and supports model-to-model and model-to-text transformations. However, oAW does not come with complete integrable transformations for different web platforms, nor does it define platform-independent elements which can be used across web applications (and possibly other paradigms). The lack of the aforementioned possibilities are certainly not necessities for a good MDA framework yet we consider them very convenient when working with a set of varying technologies intended for web applications.

One approach which has gained much attention in the web-based MDA community is the ANDROMDA MDA generator[6]. We have based our approach on ANDROMDA

[1] http://documents.google.com/
[2] http://mail.yahoo.com
[3] http://www.mozilla.org/projects/xul/
[4] http://www.xaml.net/
[5] http://www.openarchitectureware.org
[6] http://www.andromda.org

which is an open-source extensible MDA framework. This framework provides platform-independent modelling schemes to model and integrate a wide variety of scenarios and comes with a set of plugins, called *cartridges*, which manage transformations to several different web settings that use a combination of frameworks such as JSF, Struts, JSR/EJB, Spring, and Hibernate.

One point in favour of openArchitectureWare is the clean separation between the different models. Where in ANDROMDA it is not possible[7] to tweak the PSM before transforming to code or other PSMs, this remains attainable with openArchitectureWare.

Aside from RUX, none of the approaches mentioned above explore an MDA approach for AJAX web applications.

4 MDA using AndroMDA

In Section 5 we will investigate how an MDA tool such as ANDROMDA can be used to model AJAX applications. Before doing that, we first discuss ANDROMDA itself.

4.1 Example Application

In order to demonstrate the concepts more clearly, we have created an example CRUD application for managing persons. The example has been built using ANDROMDA and uses the JSF, Spring and Hibernate frameworks. The application is called PERSONMANAGER and can be used to create new persons, remove or update existing ones and to display the list of all persons found in the database.

4.2 AndroMDA

We have decided to use ANDROMDA to create an MDA AJAX cartridge. The main reason behind this decision is the availability of cartridges for different technologies and platforms, as well as the ease of integration which ANDROMDA provides for these web technologies. In ANDROMDA we can use the same models to generate a web application which uses JSF-Spring-Hibernate as well as one which uses Struts-Hibernate. Furthermore, ANDROMDA already defines PIM meta-elements which should hold for a variety of different platforms. This makes the implementation of an AJAX cartridge easier since we can concentrate on the transformation and the creation of an AJAX-specific meta-model.

While the PSM specifications are contained in the cartridges, ANDROMDA contains a list of platform independent elements which can be used to model the PIMs. These elements should generally be enough to model most cases

[7] The latest version of ANDROMDA is 4.0 and our work is based on version 3.3

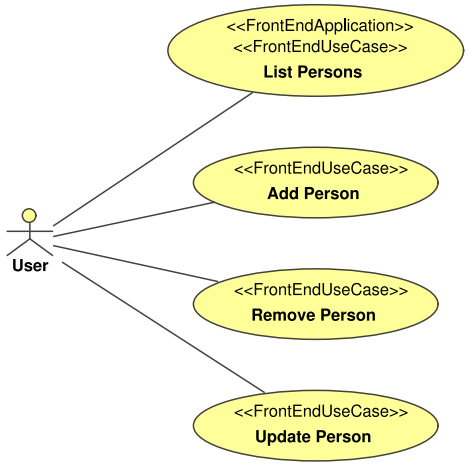

Figure 1. PERSONMANAGER **Use Cases used by** ANDROMDA**.**

but it is possible to add new PIM elements such as UML stereotypes and tagged values. The transformation between the PIM, PSM, and code is carried out by what ANDROMDA calls *metafacades* (metamodel facades). Metafacades are gateways to metamodel elements derived form the models and they provide access to the model from the cartridge templates. The function of metafacades is best described through an example. For instance, in a PIM model, a class exists with a stereotype called *TextInput*. Let's say in our cartridge, we want to transform such a stereotyped class to either a text field or a text area in an HTML page. In the UML metamodel, *TextInput* is an instance of the *Class* class. ANDROMDA allows access to this instance via metafacades, which are classes with methods for accessing various properties of this instance, such as it's fully qualified name. Such a metafacade could be called *TextInputFacade*. A cartridge wanting access to the *TextInput* instance can define its own metafacades, which can access the higher level facades such as *TextInputFacade*. Thus for our cartridge we can define an *HTMLTextInputFacade* which can decide whether to interpret a text input as an HTML text area or a text field. This cartridge metafacade can base this decision on the properties of the *TextInput* instance which it can access through *TextInputFacade*.

For a typical application which contains a user interface, ANDROMDA uses UML use case diagrams to define the possible scenarios and to split up the application into smaller parts. Figure 1 contains an ANDROMDA use case diagram for our PERSONMANAGER example application. It can be applied to different cartridges which are capable of generating a UI. In the context of a web application these

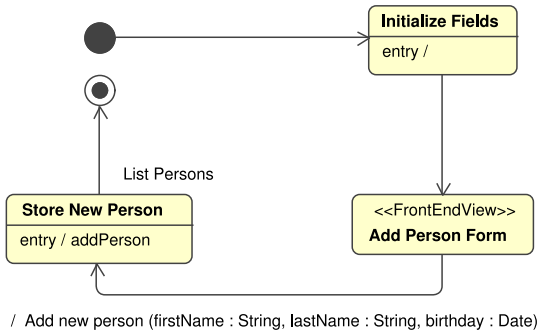

Figure 2. Add Person Activity Diagram.

four use cases should be presentable to a user through the browser. The ANDROMDA *FrontEndUseCase* stereotype is applied to use cases which are used in the front-end of an application. The *FrontEndApplication* stereotype defines which use case is the application's entry-point. Evidently there can be only one use-case which can be an instance of this stereotype.

The inner workings of each use case is subsequently specified using UML Activity Diagrams. Figure 2 shows the activity diagram corresponding with the *add person* use case of Figure 1. The activities are either server side or client side. The *FrontEndView* stereotype denotes an activity which is client side, meaning it should be rendered as a user interface. The server side activities can initiate events which initialise form fields or store data retrieved from forms. Outgoing *associations* from *FrontEndView* can contain triggers with a list of parameters. These parameters will be transformed to input fields in the form.

The diagrams mentioned above are concerned with the front-end and should be connected to diagrams which are used to model the back-end.

In ANDROMDA, the back-end of the web application is modelled using stereotyped class diagrams, similar to the conceptual models of UWE and WebML. The classes in these diagrams represent controllers and persistence entities of frameworks such as Hibernate and Spring. The methods described in the activities of 2 refer to the methods of the back-end controller classes.

5 Modelling Ajax User Interfaces

In this section we discuss the metamodel we propose for modelling AJAX UIs and the code generation aspects for the AJAX cartridge. Finally, we apply the modelling concepts to the PERSONMANAGER example.

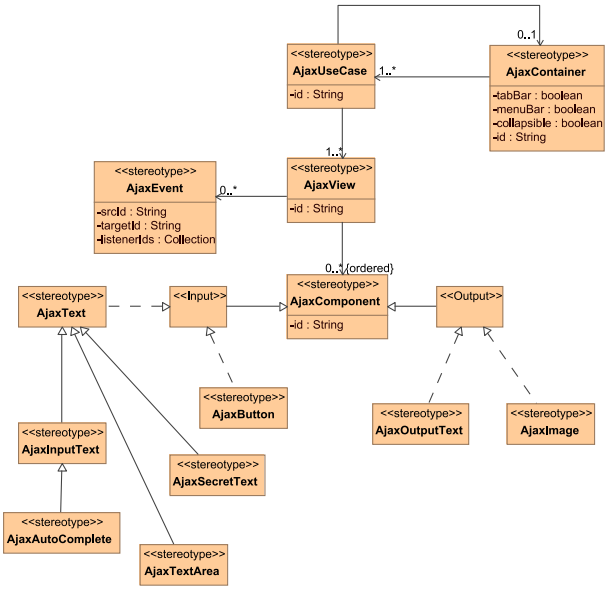

Figure 3. The proposed AJAX metamodel.

5.1 A Metamodel for AJAX

We have devised an AJAX metamodel to be used in the PSM layer of ANDROMDA. This metamodel should be able to capture the main structure of most AJAX user interfaces.

The semantics of an AJAX user interface closely resembles that of a native desktop application. In both cases the user interface consists of a tree of application artifacts such as windows, labels and buttons. Therefore examining how the UI of desktop applications is modelled seems to be a reasonable starting point to creating a modelling scheme for AJAX. Desktop and AJAX applications generally offer the following UI components [9]: Input components (e.g., text field with *auto-completion* feature, text area, button, file, anchor, radio button and checkbox); Output components (e.g., label and media); Layout components (e.g., panel layout, tab bar and menu bar).

Based on the structure of the Swing API, the tree like structure of markup languages (e.g., XUL), and general components of AJAX frameworks (ICEFACES) we have created a modelling scheme for AJAX. Figure 3 shows a simplified version of the AJAX metamodel which can be expanded by adding more UI components. In principle, this metamodel can also be used to model the UI of desktop applications employing event-based libraries such as Swing.

Containers and Navigation As can be seen in Figure 3, an instance of *AjaxContainer* is the owner of one or more *AjaxUseCase* objects. It should provide access to the ap-

plication use cases by using one of the several navigation mechanisms available to the object. An *AjaxUseCase* instance is the result of the transformation of the PIM *FrontEndUseCase* element. In the context of the PSM, a *use case* defines a section of an AJAX UI which contains the elements responsible for performing a use case and should be navigable using a single-page mechanism. Ideally, information such as which navigation mechanism is to be used should be specified in the PSM, once it has been generated from the PIM. As we mentioned earlier, this is not possible in the version of ANDROMDA we have been working on and therefore *marks* should be defined in the PIM to support a correct and complete generation of a PSM. Marks are pointers in the PIM which give hints as to how certain PIM elements should be transformed to PSM elements [11].

Nested containers are possible by allowing an instance of *AjaxUseCase* to own a container which in turn consists of one or more use cases. Examples of this behaviour can be seen in nested menus or navigation trees.

Views and Components The core of the user interface which is presented to the user is contained in instances of *AjaxView*. Each use case can hold one or more views. Subsequently, each view can contain a number of ordered user interface components. Multiple views are convenient in cases where all the components should be replaced by new components, e.g., a large form spanning several pages.

The views are holders of the categorised UI components mentioned above. While the layout components are reflected in the *AjaxContainer* stereotype, the core UI components fit into the view and are either components handling input from the user, or displaying information (output) on the browser.

Events and Listeners The way events are initiated and handled is one of the fundamental differences between AJAX and classical web applications. With AJAX a whole refresh of the page is not required as a consequence of an event. Just as in desktop applications, components which are interested in an event can be registered as listeners and be notified when the event has occurred. The *AjaxEvent* stereotype is designed to capture this information for components contained in a view. Each view contains zero or more events. For each event it should be specified which element has initiated it (*srcId*), which element should be displayed as a result, and a list of elements which have been registered as listeners of this event.

5.2 Code Generation

Code generation for the metamodel described above is carried out in the same manner as in the other cartridges of ANDROMDA. Apache Velocity [8] template files gain access

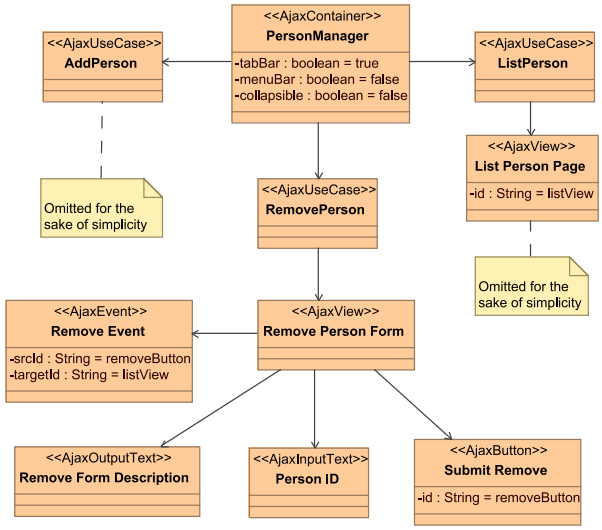

Figure 4. An instance of the AJAX metamodel for PERSONMANAGER.

to model elements through metafacades. They can subsequently iterate through the model abstract syntax tree and write web application files. These files will be based on the ICEFACES AJAX framework, which is itself based on the JSF framework. The completeness[9] of ICEFACES in terms of the AJAX components it offers persuaded us to make this choice. Furthermore, we have chosen to implement a *facelets*[10] version of ICEFACES, which constitutes that the implemented files concerned with the view are actually very similar to a component-tree, so the model has a somewhat intuitive mapping with the generated code. *Facelets* uses file inclusions to form a tree of *.xhtml* files which represent the UI of the web application. This is very similar to XML-based UI descriptors such as XUL and can be created by the templates while iterating through the model elements. The files generated by ANDROMDA form the structure of the web application and are already integrated with the other employed frameworks through the back-end classes. The core business logic which typically resides in the *controller* part of the web application should be coded manually by the developer.

5.3 Case Study Using the AJAX Cartridge

We apply the above concepts for modelling AJAX UIs to the PERSONMANAGER example mentioned earlier. PERSONMANAGER has already been built using the AN-

[8] http://velocity.apache.org/

[9] http://component-showcase.icefaces.org/

[10] Facelets is a view technology for JSF, and a powerful templating system based on XML-style templates, https://facelets.dev.java.net

DROMDA JSF cartridge, which uses a legacy UI. A shift to AJAX would mean each of the use cases should be accessed through a single-page mechanism. An example model for PERSONMANAGER based on the AJAX metamodel is shown in Figure 4. In this model, the navigation to the use cases is managed through a tab bar and the only event described for the *remove person* use case is captured using the *AjaxEvent* stereotype. This event only causes the view to be changed but it could also have a list of listeners (other components) be updated.

The development of the AJAX cartridge for ANDROMDA is a work in progress and a working cartridge will be available for download as soon as enough code can be generated for a simple AJAX application.

6 Discussion

Issues regarding which data resides on the server or the client are currently not captured by our model, since we believe these are framework-specific (for example GWT and Echo2 designs differ in this issue). The choice of AJAX framework may somewhat affect the AJAX metamodel proposed in this paper. This will be mainly in the area of the offered components and not the main structure of the web application.

Many web applications seen today are a blend of AJAX and classical multi-page applications where small components in the page use AJAX to avoid refreshing the whole page. Our aim has been to devise a metamodel for single-page user interfaces and the aforementioned applications do not fit in this category. Nevertheless we believe that with minor adjustments to the metamodel these hybrid applications can also be modelled, e.g., events which cause a redirect to a new instance of *AjaxContainer*.

7 Concluding Remarks

The main contributions of this paper can be summarised as follows: an overview of the current model-driven web approaches; a meta-model for modelling AJAX user interfaces in UML and the proposed approach for creating an AJAX-based ANDROMDA cartridge with ability to integrate with back-end components such as Spring and Hibernate.

Future work consists of completing and testing the implementation of the AJAX cartridge and conducting case studies to evaluate our proposed meta-model and the overall MDA approach for modelling and generating AJAX web applications. The cartridge will be made available for download soon.

References

[1] M. Abrams, C. Phanouriou, A. L. Batongbacal, S. M. Williams, and J. E. Shuster. UIML: An appliance-independent XML user interface language. In *WWW '99: 8th International Conference on World Wide Web*, pages 1695–1708, 1999.

[2] J. Carlos Preciado, M. Linaje, S. Comai, and F. Sanchez-Figueroa. Designing Rich Internet Applications with Web engineering methodologies. In *Proceedings of the 9th IEEE International Symposium on Web Site Evolution (WSE'07)*, pages 23–30. IEEE Computer Society, 2007.

[3] S. Ceri, P. Fraternali, and A. Bongio. Web modeling language (WebML): a modeling language for designing web sites. *Computer Networks*, 33(1-6):137–157, 2000.

[4] J. Conallen. *Building Web Applications with UML (2nd Edition)*. Addison-Wesley, 2003.

[5] J. Garrett. Ajax: A new approach to web applications. Adaptive path, 2005. `http://www.adaptivepath.com/publications/essays/archives/000385.php`.

[6] N. Koch and A. Kraus. The expressive power of UML-based web engineering. In *IWWOST '02: 2nd International Workshop on Web-oriented Software Technology*, pages 105–119. CYTED, 2002.

[7] F. J. Martìnez-Ruiz, J. Munoz Arteaga, J. Vanderdonckt, and J. M. Gonzàlez-Calleros. A first draft of a model-driven method for designing graphical user interfaces of Rich Internet Applications. In *LA-Web '06: Proceedings of the 4th Latin American Web Congress*, pages 32–38. IEEE Computer Society, 2006.

[8] A. Mesbah, E. Bozdag, and A. van Deursen. Crawling Ajax by inferring user interface state changes. In *Proceedings of the 8th International Conference on Web Engineering (ICWE'08)*. IEEE Computer Society, 2008.

[9] A. Mesbah and A. van Deursen. Migrating multi-page web applications to single-page Ajax interfaces. In *Proceedings of the 11th European Conference on Software Maintenance and Reengineering (CSMR'07)*, pages 181–190. IEEE Computer Society, 2007.

[10] A. Mesbah and A. van Deursen. A component- and push-based architectural style for Ajax applications. *Journal of Systems and Software (JSS)*, 2008. To appear.

[11] J. Miller, J. Mukerji, et al. MDA Guide Version 1.0.1, 2003. `http://www.omg.org/docs/omg/03-06-01.pdf`.

[12] OMG. MDA, 2008. `http://www.omg.org/mda`.

[13] D. C. Schmidt. Model-driven engineering. *Computer*, 39(2):25–31, 2006.

[14] M. L. Trigueros, J. C. Preciado, and F. Sánchez-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*, pages 226–241. Springer, 2007.

[15] E. Visser. WebDSL: A case study in domain-specific language engineering. In *Generative and Transformational Techniques in Software Engineering (GTTSE 2007)*, Lecture Notes in Computer Science. Springer, 2008.

TUD-SERG-2008-024
ISSN 1872-5392