Fraunhofer Institut
Experimentelles
Software Engineering

Experience in model-driven UI-development using a MDA-compliant framework

Authors:
Sebastian Adam
Christian Bunse
Patrick Kolling

Submitted for Publication to
Models/UML 2006

IESE-Report No. 030.06/E
Version 1.0
February 21, 2006

A publication by Fraunhofer IESE

Fraunhofer IESE is an institute of the Fraunhofer Gesellschaft.
The institute transfers innovative software development techniques, methods and tools into industrial practice, assists companies in building software competencies customized to their needs, and helps them to establish a competitive market position.

Fraunhofer IESE is directed by
Prof. Dr. Dieter Rombach (Executive Director)
Prof. Dr. Peter Liggesmeyer (Director)
Fraunhofer-Platz 1
67663 Kaiserslautern

Abstract

The success of a company depends directly on its ability to quickly adapt to changing market requirements. One means for reaching this goal is reuse technology. Reuse, known since the beginning of computer science (e.g., reuse of command sequences), has been raised to the level of components. Component-based development promises fast development of software systems by assembling pre-fabricated building blocks or components. Although, this is a step into the right direction, the biggest remaining problem is that a system should be compatible with as many commonly used implementation and middleware technologies as possible, particular those that are either de facto or de jure standards. This is the goal of the Model Driven Architecture (MDA) approach, which is based on the concept that the essential structure and behavior of a system should be captured in an abstract way so that it can be mapped flexibly to different implementation technologies. When applying the MDA approach to the development of user interfaces, systems can be easily ported or deployed to different platforms. Good examples are web-based applications which can be used on mobile phones, PDAs, portable computers, etc. without the need for developing different interface implementations. This paper presents an experience report concerning the application of an integrated modeling approach for reusable user interfaces, and shows how modeling and MDA can be efficiently used in practice.

Keywords: Model-based software development, model-driven development, user interfaces, human-computer interfaces, Unified Modeling Language (UML), generative programming, empirical evaluation, industrial experience

Table of Contents

1 Introduction

Component-based (CB) and object-oriented (OO) development of high quality systems has become a key issue for many industrial organizations. Typically cited promises include higher reuse opportunities, increased development speed, improved software quality through lower failure rates, and lower costs associated with failure diagnosis and repair [Atk01, Szy02]. Because of these promises, OO and CB approaches have become the approach of choice for many development projects. As a side-effect, the need for having a technology independent model of a system and 'automatic' transformations to different platforms and implementation technologies was born. This is the goal of the OMG's Model Driven Architecture (MDA) approach [OMG03]. The major goal of the MDA approach is to separate the design of a system from architecture and realization technologies. Using the MDA approach, system functionality is defined as a platform-independent model (aka PIM) and then translated to one or more platform-specific models (aka PSMs). To accomplish this goal, the MDA defines an architecture that provides a set of guidelines for structuring specifications expressed as models. The translations between the PIM and PSMs are normally performed using automated tools [Wik06].

The MDA approach is a step into the right direction; however, the user interface (UI), an important, expensive, and often-changed ingredient of modern software systems, seems to be neglected frequently. This is quite strange since the model-driven and generative development of user interfaces (MD-UID) is as old as UIs themselves [Mol04]. In detail, quite some effort has been spent in the development of approaches for UIs that promise to provide the same benefits as MDA but for the development of user interfaces [Lim04]. Unfortunately, none of them is widely accepted in software organizations, or even integrated in model driven software processes [Mol04]. Visual (ad-hoc) programming using WYSIWYG-tools is still the state-of-practice. According to [Mol04] the most prominent reasons are:

- *Scalability:* Most MD-UID approaches origin from research in the area of "Human-Computer Interaction" (HCI) and have not been successfully transferred to industrial software development practice yet. One reason is that these approaches are only loosely coupled to the overall development process.
- *Lack of Standards:* There exists no standard notation for UI modeling. Nearly every approach proposes its own notation, whereby none of these are used in industrial software practice.

- *Lack of integration with business logic:* Since modeling of UIs uses tools and notations different to those applied to the "rest" of the system, it is nearly impossible to generate business logic and UI in an integrated way.
- *Lack of commercial tool support:* Most existing MD-UID tools are unreliable prototypes. However, practical acceptance crucially depends on the existence of reliable tools. Thus, this problem is directly related to the "lack of standards".

In this paper we present experience made in the industrial application of a novel framework for the integrated modeling of reusable user interfaces, also known as IMRU [Ada05]. In the context of IMRU the term "Integrated modeling" specifies that the UI and business logic modeling are integrated with each other, using the same modeling language and tool environment. Regarding HCI and its underlying "usability philosophy", the UI is perceived as an integral part of a software system and not as a less important add-on. The term "reusable" denotes that the high-abstraction models of the UI are reused for generation of completely different characteristics, following the MDA paradigm. In this paper, "characteristic" is defined as a specific UI with a particular look&feel for a particular device, implemented in a particular programming language. In general, the goal of IMRU is to provide a systematic but usable approach for generative UI development, which promises the following benefits:

- Increased quality due to the systematic integration and support of UI development in the overall development process. This is further supported by the use of "formalized" usability knowledge in transformation patterns.
- Increased productivity due to the reuse of models and model-transformations.
- High practical acceptance due to the use of "standard" notations and tools.

The remainder of this paper is structured as follows: Section two describes the foundations of model-driven UI-development and demonstrates how modeling steps can be mapped to the corresponding MDA levels. Section three presents related work, and section four the basic ideas of the IMRU approach. Finally, section five describes the practical application and validation of the IMRU approach, while section six presents a short summary and provides some conclusions.

2 Bridging MDA and MD-UID

In the context of model-driven software development, following the MDA paradigm, a system is specified/characterized independently from its realization in the form of computation independent models (CIM). Based on the systems CIM, its functionality is defined by means of a platform-independent model (PIM), using an appropriate specification language and then translated to one or more platform-specific models (PSMs) for the actual implementation. To accomplish this goal, the MDA defines an architecture that provides a set of guidelines for structuring specifications expressed as models. The translation between a PIM and its PSMs is normally performed using automated tools [Wik06].

Interestingly, recent MD-UID approaches too distinguish four levels of abstraction: tasks & concepts models, abstract user interface models, concrete user interface models and the final user interface model [Van05]. Task & concepts models describe the environment, especially the tasks which should be supported by the software system. Abstract user interface models specify the logical structure of the UI in terms of workspaces which group related data and functions. Concrete user interface models define the layout of an UI for a particular device, but are still independent of the implementation. The final user interface is then the concrete implementation of an UI in a specific programming language.

When comparing both abstraction hierarchies, direct relationships between levels are obvious [Van05] (see Figure 1). Therefore, it can be assumed that MDA and MD-UID can be closely integrated.

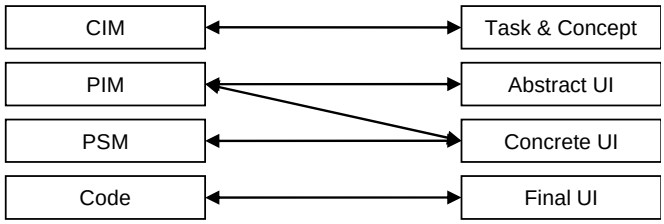

Figure 1. Relationship between MDA- and MD-UID abstraction levels

3 Related Work

User-interface modeling for software systems has been in the focus of research for quite some time (see [Pin00] and [Lim04] for a comprehensive overview). The corresponding approaches can be classified based on their usage of different (input) models, their levels of abstraction, and their platform independence. A good example is the approach presented in [Lim04] that distinguishes different levels of abstraction which can be directly mapped to MDA levels. In addition to the definition of meta-models for each abstraction level, the approach provides a transformation-syntax based on graph grammars and hereby supports a high degree of automation. However, the approach uses a non-standard modeling and transformation language (USIXML). In addition, comprehensive tool support does not exist. Therefore, a broad acceptance in practice can not be assumed and the possibility to integrate this approach in a (typical) UML-based software development method is quite low.

Other approaches such as [Pat01] or [Bla04] propose to apply UML in MD-UID in order to increase practical acceptance. However, none of these approaches is able to generate UIs out of UML models in a generic manner using the full range of abstraction levels. Typically, windows of the final UI are directly generated from UML classes by mapping each attribute on a form field [Bal95]. Another approach is to use UML state charts to generate controller-components in MVC-compliant UIs. Unfortunately, these approaches either neglect the "look & feel" aspect or provide only fixed transformation patterns. One reason might be that UML does not provide a means for manually designing the presentational aspects of an UI. Some tools (e.g., [Neu04]) try to overcome this weakness by combining UML models for controller components and WYSIWYG-definitions. However, the latter is not abstract and only defines the "look & feel" for a particular device.

In summary, existing UML-based approaches are too restrictive for a generic model driven UI development and do not provide the required methodical maturity. On the other hand, recent MD-UID approaches which don't use UML are not supported by reliable tools or standardized notations and thus are hard to apply in practice.

4 The IMRU-Approach

4.1 Overview

The challenge in MD-UID is to provide a systematic approach based on standard languages and tools which can easily be applied in software industry. The basic idea is to bridge the gap between model-driven application-logic- and user-interface development by adopting the UML for MD-UID, and by integrating MD-UID concepts with corresponding MDA abstraction levels (see also section 2). The IMRU ("integrated modeling of reusable user interfaces") approach [Ada05] provides a framework for MD-UID which in addition addresses the systematical reuse of existing models and transformation patterns.

In detail, the integration of UI modeling in the overall software development process (e.g., into the Unified Process [Kru03]) is primary achieved defining specific activities, artifacts and quality measures that are embedded into the software design phase. Starting from conceptual models and task descriptions, an abstract user interface model is designed. This model is further refined to a more specific user interface model which specifies the widgets needed to visualize it on a given modality. The next development step is the creation of a concrete user interface which describes the layout on a chosen device. The last step is finally the implementation in a given technology. In section 4.2 these development steps and the related models are described in more detail.

The link between UI- and system models is realized by using UML as far as possible as a common language. In cases, where UML is not suitable (e.g. definition of the graphical layout) specific languages for that purpose such as XHTML [W3C02] can be integrated, using UML's in-build extension mechanisms.

The reuse of UI artifacts and models in IMRU is supported in two different ways. On the one hand, models which describe all possible interactions and logical coherences independently of any modality, look & feel or technology can be (re)used without changes to the semi-automatic generation of several UIs within the same system. On the other hand, transformation patterns can be reused across several projects. Good design decisions, usability knowledge and sophisticated layouts can hereby easily and efficiently be applied.

However, in general, IMRU only provides a framework for MB-UID in a MDA-compliant way. An adaptation to particular meta-models and transformation patterns is beyond this general approach.

4.2 IMRU Abstraction & Development Levels

As depicted by Figure 2, IMRU separates five levels of abstraction based on the MDA abstraction hierarchy. These are connected by transformation activities, either with a high degree of automation (solid lines) or manual transformations (dashed lines). In addition there are external influences (dotted lines) such as the impact of the system context on the selection of appropriate transformation rules.

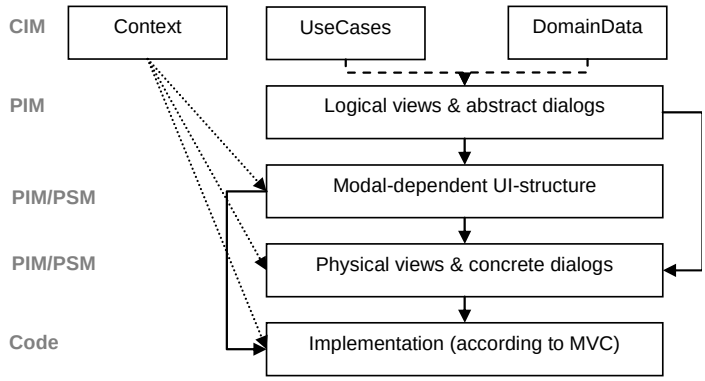

Figure 2. Levels of abstraction in IMRU

IMRU allows the generation of multiple UI characteristics based on three dimensions: modality[1], look & feel (depending on user preferences, application type, and device), and implementation technology, which all have an impact on the transformations between levels. In order to preserve the ease of use, major design decisions are hidden from developers by encapsulating them in reusable, context-specific transformation patterns.

The only models which have to be created manually are the logical views and abstract dialogs, based on the information described in the use case and domain data models. The reason for this manual step is that such transformations are hard to formalize with an acceptable level of quality [Pin00]. Following the general idea of software product lines [Atk01] these models contain the commonalities of all possible UIs of a system.

4.3 Context, Use Cases, and Domain Data

IMRU assumes three artifacts to be existent before UI development can start. In detail, these are use cases for task descriptions, textual descriptions of the application environment, and class diagrams for the domain data specification.

[1] Modality describes the kind of interaction provided by an user interface (e.g., visual vs. speech-based UIs).

These artifacts correspond to the task-, context- and domain- models suggested in other HCI approaches (e.g., see [Lim04]).

4.4 Logical Views & Abstract Dialogs

Logical views group all necessary data and operations a user needs to perform a given activity. For each interaction step in a use case, a (dialog) state is specified in a UML state transition diagram while a related UML class describes all corresponding data and operations for this state. Depending on the underlying meta-model, the access privileges for each user role as well as error handling can be specified.

The specification of logical views and abstract dialogs is completely independent of the modality, look & feel, and the implementation technology. Thus, they represent some form of PIM, and can therefore be generically (re)used for all UIs of a system.

4.5 Modal-dependent UI-structure

The transformation of logical views to a modal-dependent UI-structure is known as "structural design". The UI-structure describes the architecture of the user interface in terms of common widgets, valid for all devices of this modality. The corresponding models are platform-specific regarding modality, but platform-independent regarding technical decisions. Major parts of the UI-structure can be automatically generated, whereby only structural attributes like field names/sizes should be manually adapted.

4.6 Physical views & Concrete Dialogs

To communicate with users (e.g. concerning usability) the real appearance of the UI (called physical view) is needed as soon as possible during development. The IMRU approach allows generating models of the physical view almost automatically from the "modal-dependent UI-structure". The only problem being, that UML does not support the definition of display aspects. Many approaches try to solve this problem by introducing proprietary notations [Lim04]. However, the use of standards is a prerequisite to increase technology acceptance in practice. Therefore, the use of standard notations is required, especially for the definition of physical views. In IMRU declarative language such as XHTML are used to describe the visual aspects.

4.7 Implementation

The "implementation" activity addresses the transformation to the "implementation" abstraction level. The UI-structure and concrete dialog models are automatically transformed to model- and controller-code-components of a MVC-organized [Wik06b] user interface. Also the physical views can be automatically enriched with necessary constructs for the accordant implementation language.

Figure 3 gives examples for all above-mentioned model types.

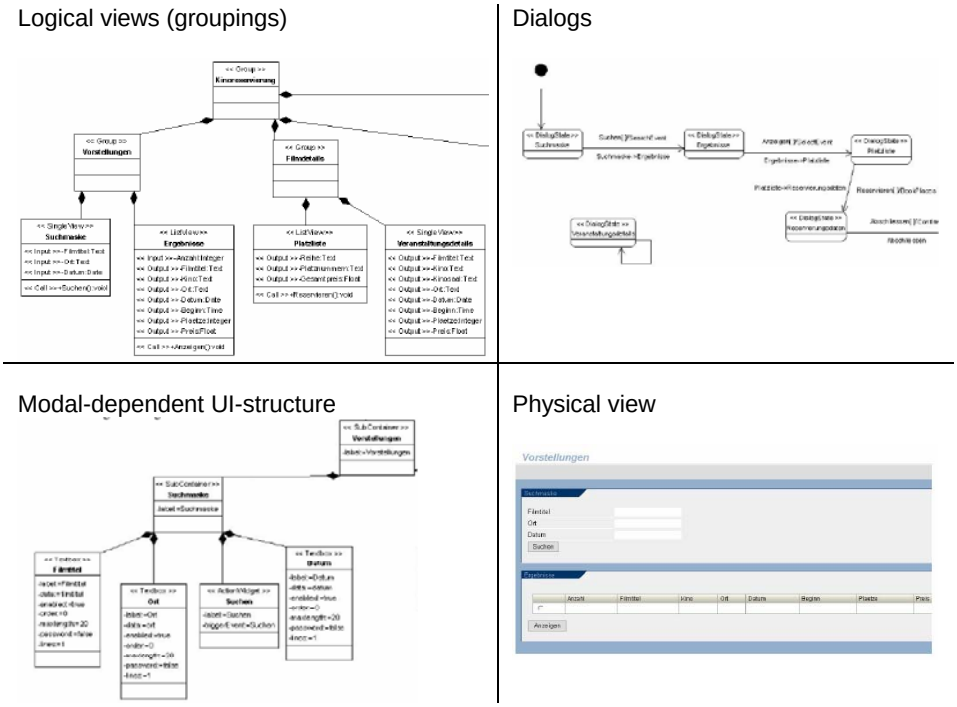

Figure 3: Example IMRU models

5 Practical validation

The validation of the IMRU approach was performed in two steps. First, a case study was performed in order to check if the IMRU concepts can actually be applied as intended. Second, an empirical study was conducted in order to validate if the promised benefits (increased development speed and practical acceptance) are achieved.

5.1 Case Study

The goal of the case study was twofold: First, to perform a kind of a feasibility study; and second, to prepare the empirical study. In the context of this study exemplary UML-profiles (meta-models) for logical views, abstract and concrete dialogs, as well as for the modal-dependent UI-structure (for graphical UIs) were developed. In addition, transformation patterns for all levels of abstraction, taking usability guidelines into account, were defined. At the lowest levels the focus was on web- and WAP-based UIs, using Jakarta Struts as underlying technology.

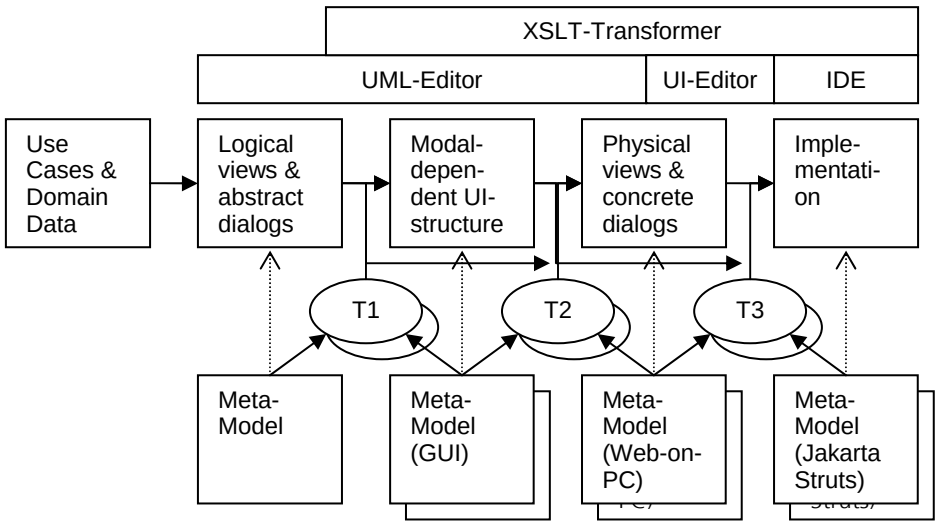

Figure 4. Usage of models, meta-models, transformation patterns and tools

Furthermore, a tool chain was established using standard tools for model-driven software development (e.g. Poseidon UML [Gen06]). The transformation patterns were then realized by using XSLT [W3C99].

Figure 4 provides an overview on the case study environment. This framework was used by some students to develop several exemplary UIs of different sizes for web-based systems. In detail, existing use cases were analyzed and then transformed into corresponding logical views and abstract dialogs by means of an UML editor. These models were then automatically transformed to a modal-dependent UI-structure for graphical UIs. In a third step, several transformation patterns for obtaining the physical views and the related concrete dialogs were applied. In detail, one WAP-based and two web-based UIs, each with a different look & feel, were created, followed by some manual refinements using a WYSIWYG-editor. Finally the models were transformed to an executable UI, using Java and JSP (Jakarta Struts). The code was then transferred to an IDE, compiled and executed. Finally, the UI was validated against its use case descriptions. The results of the case-study can be summarized as follows:

- All relevant aspects of UIs can be described in standardized notations according to a given meta-model on different levels of abstraction.
- Relevant concepts of MD-UID approaches can be mapped to common software tool chains.
- Usable and appealing UIs can be automatically generated from UML-models using well known and standardized technologies, such as XSLT.
- Product line concepts are transferable to user interfaces. Commonalities of all system UIs can be defined in abstract models whereby specific characteristics can be generated by applying transformation patterns.
- All transformation steps are largely automatable (up to 100%). This is mainly due to the fact that UIs do not contain complex logic and are typically constructed from a predefined set of possible components.

In summary, it can be concluded that the IMRU approach can be applied in practice. However, the results of the case study were gathered from a subjective point. In order to obtain objective results an empirical study was performed which is described in the next section.

5.2 Empirical Validation

In general, empirical studies in software engineering are used to evaluate whether a "new" technique is superior to other techniques concerning a specific problem or property. This section examines the problem of systematic, user-interface development using UML. The benefits of the IMRU approach are defined by a couple of research hypotheses.

Hypotheses

It is expected that the application of IMRU will reduce development time, improve the efficiency of a project, and increase the MD-UID method's perceived applicability. Thus, the experiments hypotheses can be stated as:

- The creation of abstract UML models for UIs, as well as their refinement and transformation into an executable form needs less effort than straight WYSIWYG-programming.
- MD-UID based on IMRU is regarded as "practically suitable" (subjective view).

Material

The experimental material used in this study to test the hypothesis was a small cinema reservation system with a web- and a WAP based user interface. The system provided functions to search for movies, choose seats, and to order tickets, including payment transactions.

Subjects

The experimental subjects used in this study were six professional developers of T-Systems International, located in Saarbrücken, Germany. All of them already had professional experience in using UML and in developing graphical user interfaces, either by using WYSIWYG-editors, or by manual programming.

Tasks

For the given system document the experimental task was to develop a web- and a WAP-based user interface based on the Jakarta Struts Framework. Therefore subjects had to develop the UIs using the model-driven approach on the one hand and to manually program the UIs on the other hand.

Design

To avoid learning effects, the experiment was performed as a crossed test (see Table 1). In the first round, one half of the participants developed the web-based UI using a WYSIWYG-tool, while the other half used the IMRU approach with its tool chain. In the second round, the same UI was developed using the other technique.

Concerning the development of the WAP-based UI, the task was performed in a less formal way than the development of the web-based UI. Instead of redesigning the UI from scratch, the existing abstract models were used to generate the WAP-based UI. Thus, only the reuse- and transformation effort was measured and compared with the effort for the WYSIWYG design.

Subject	1. Task	2. Task
1	WYSIWYG	IMRU
2	IMRU	WYSIWYG
3	IMRU	WYSIWYG
4	IMRU	WYSIWYG
5	WYSIWYG	IMRU
6	WYSIWYG	IMRU

Table 1: Test design

Measured data

Data was collected for all subjects over the two experimental runs. Therefore, six data points were available for each round. Since the experimental design was completely within-subjects, repeated measures analysis can be performed. The first step of the analysis procedure is to check the normality of the data. Because the collected data (see Table 2) were substantially non-normal an appropriate test to use was the non-parametric t-test.

Subject	**1**	**2**	**3**	**4**	**5**	**6**
IMRU [min]	180	245	215	150	115	210
WYSIWYG [min]	325	280	295	290	225	270
Saving [min]	145	35	80	140	110	60
Saving [%]	45	13	27	48	49	22

Table 2: Raw Data (Web-based UI)

Analysis

Based on the raw data shown in Table 2 a statistical analysis has been performed. Results show that the effort for developing a UI with IMRU is, in average, 34% less than the effort for manual programming using a WYSIWYG tool (see Table 3).

Average saved effort [min] / [%]	95 / 34
Standard deviation s [min] / [%]	44,3 / 15,3
t-test variable t_0	$5{,}26 > 4{,}03 = t_{0.005,5}$

Table 3: Descriptive statistics and t-test result

For the creation of both UIs averagely 56% of effort could be saved by using IMRU (the development for the web-based UI took only about 15 minutes), whereby the significance is below the α–level of 0.01. The hypotheses that a

model-driven UI-development using IMRU needs less effort than a manual WYSIWYG-programming can thus be accepted.

Also interestingly, nearly 75% of the whole IMRU effort was used for platform-independent modeling. This supports the assumption that IMRU enables a high degree of reuse. Especially development projects with several UIs can realize significant effort savings.

In addition to the quantitative results, qualitative results were collected by means of asking subjects to fill a debriefing questionnaire. In summary, the results show that the IMRU approach can be applied in industrial software practice. In detail, subjects said that the learning effort for IMRU is "normal" and that the proposed development steps are "intuitive and easy to apply". Furthermore, subjects estimated that IMRU will result in a higher efficiency in system development using different technologies, less faults, and a better quality. Finally, the separation of different model types and abstraction levels was seen as a "helpful support for the division of work between graphical layout, logical design and implementation". In this regard, subjects mentioned that they expect a relief of technology- and layout knowledge in MD-UID. In summary, the analysis of the debriefing questionnaire showed that IMRU is perceived as "a helpful, efficient and practically suitable method". This allows the second hypothesis to be accepted.

In summary, the hypotheses, that model-driven UI-development using IMRU is more efficient than manual programming using WYSIWYG and that IMRU is considered "practically suitable" are confirmed by the performed experiment.

Threats to validity

Threats to construct and external validity are: The low number of participants, the low complexity of the experimental tasks, and the focus on data-oriented software systems using tables and forms as UI elements. Thus, the results can only be considered as an trend but not as a valid general conclusion, which requires additional experiments [Gre90]. Therefore, a replication package is available from the authors.

6 Conclusion & Future Work

The recent advent of the MDA has created a special focus on the model-driven development of software systems. Unfortunately, this focus has not fully been extended to the user interface. The IMRU approach provides a framework for the development of user interfaces in a MDA-compliant and generic way, based on standard notations and tools. Its practical applicability has been validated in form of a case study and an empirical study.

The case study has shown that it is possible to merge/use model-driven development and MDA principles for the development of user interfaces with standard notations and tools. The results affirm the assumption that IMRU is a step into the right direction. In addition, the empirical study has shown that IMRU significantly increases the acceptance of MD-UID in practice, and that IMRU's efficiency is significantly better than the efficiency of "traditional" WYSIWYG-programming. Thus, it can be expected MD-UID using technologies and tools such as suggested by IMRU will become state-of-the-practice.

However, to reach industrial acceptance future work is needed for completing the IMRU approach. This includes the completion/definition of meta-models for each abstraction levels, transformation patterns for all well-established platforms and UI elements, sophisticated tool support (e.g., a plugin for case tools such as Rational Rose or IDEs such as Eclipse), and further evaluations by means of industrial case-studies.

References

[Ada05] Adam, S.: Derivation of an approach for the integrated modeling of reusable user interfaces (in german: "Entwicklung eines Ansatzes zur integrierten Modellierung wiederverwendbarer Benutzerschnittstellen"). TU Kaiserslautern, 2005

[Atk01] Atkinson, C., Bayer, J., Bunse, C., et al. Component-based Product Line Engineering with UML. Addison Wesley, 2001.

[Bal95] Balzer, H.: From OOA to GUI – The JANUS-System. In: Proceedings of the Fifth Conference in Human-Computer Interaction INTERACT'95. 1995

[Bla04] Blankenhorn, K.: A UML Profile for GUI Layout. In: Lecture Notes in Computer Science, LNCS 3263, Springer-Verlag, Berlin Heidelberg, 2004

[Gen06] Gentleware: Poseidon UML. http://gentleware.com/downloadcenter.0.html, 2006

[Gre90] Green, S., Kouchakdjian, A., Basili, V., Weidow, D.: The Cleanroom Case Study in the SEL. TR SEL-90-002, NASA-SEL, 1990

[Kru03] Kruchten, P., Rational Unified Process: An Introduction. Addison-Wesley, 2003

[Lim04] Limbourg, Q.: Multi-Path Development of User Interfaces. Dissertation. Université catholique de Louvain, 2004

[Mol04] Molina, P.: A Review to Model-Based User Interface Development Technolgy. In: Proceedings of the first Workshop on Making Model-Based User Interfaces Practical, Island of Madeira, 2004

[Neu04] Neuhaus, W.: Modellgetriebene Softwareentwicklung – Alles UML, oder? In: ObjektSpektrum 1/2004. Sigs-Datacom, Troisdorf, 2004

[OMG03] Object Management Group: MDA Guide Version 1.0.1, http://www.omg.org/mda/, 2003

[OMG05] Object Management Group: Unified Modeling Language, Version 2.0, http://www.uml.org, 2005

[Pat01] Paterno, F.: Towards a UML for Interactive Systems. In: Engineering for Human-Computer Interaction, LNCS 2254, pp. 7-18. Springer, 2001

[Pin00] Pinheiro, P.: User Interface Declarative Models and Development Environments; A Survey. In: DSV-IS 2000, LNCS 1946, pp. 207-226. Springer, 2000

[Szy02] Szyperski, C., Gruntz, D., Murer, S., Component Software. Beyond Object-Oriented Programming, Addison-Wesley, 2002

[Van05] Vanderdonckt, J.: A MDA-Compliant Environment for Developing User Interfaces of Information Systems. In: CAiSE 2005, LNCS 3520, Springer, 2005

[W3C99] W3C: XSL Transformations (XSLT) http://www.w3.org/TR/xslt, 1999

[W3C99b] W3C: Cascadian Style Sheets. http://www.w3.org/Style/CSS/, 1999

[W3C02] W3C: XHTML 1.0 The Extensible HyperText Markup Language, http://www.w3.org/TR/xhtml1/, 2002

[Wik06] Wikipedia: Model Driven Architecture. http://de.wikipedia.org/wiki/Model_Driven_Architecture#Werkzeuge, 2006

[Wik06b] Wikipedia: Model View Controller. http://de.wikipedia.org/wiki/MVC, 2006

Document Information

Title:	Experience in model-driven UI-development using a MDA-compliant framework
Date:	February 21, 2006
Report:	IESE-030.06/E
Status:	Final
Distribution:	Public