Focusing Graphical User Interfaces in Model-Driven Software Development

Stefan Link[1], Thomas Schuster[2], Philip Hoyer[1], Sebastian Abeck[1]

[1]*Research Group Cooperation & Management, Universität Karlsruhe (TH), 76128 Karlsruhe, Germany*
[2]*FZI Forschungszentrum Informatik, Haid-und-Neu-Str. 10-14, 76131 Karlsruhe, Germany*
{ link | hoyer | abeck } @ cm-tm.uka.de, schuster@fzi.de

***Abstract* — To meet fast changing demands on modern software architectures the ambition to shorten and improve software development processes has increased. The approach of model-driven software development focuses models as specification of software and on transformations of those models to finally get source code. The advantage of the model-driven approach still has to be proven because a continuous tool-supported transformation process from model to source code with regard to all aspects of a software system is not yet possible. This paper concentrates on the aspect of user interaction by presenting an easy to apply approach allowing for a tool-supported, model-driven software development of graphical user interfaces for any kind of platform. A case study demonstrates the usage and benefit of our model-driven approach applied to a common software development process.**

***Keywords* — Graphical User Interfaces (GUI), Model-Driven Architecture (MDA), Model-Driven Software Development (MDSD), Software Engineering**

I. Introduction

A. Background

Facing fast changing markets, enterprises have to adapt their business processes in decreasing intervals to keep up with their competitors. Supporting such business processes, Information Technology (IT) has to follow the changes, hence the need for a more flexible, interoperable and adjustable IT arises [1]. Due to these requirements, existing process models in software engineering have to be improved to strive for shorter development cycles and to be able to handle more complex software systems.

The approach of model-driven software development (MDSD) aims to achieve these improvements by centering the modeling of a software system in any software development process [2]. Thus MDSD is not a new process model itself but can be applied to the better part of known process models in software engineering [3]. With MDSD a software system is specified through models on a very abstract level. Following the phases of a software development process, models are transformed stepwise to more specialized models with a lower level of abstraction by model-to-model transformations. In a final model-to-text transformation the detailed models are transformed to source code of the desired platform [4].

B. Motivation

The Model Driven Architecture (MDA) published by the Object Management Group (OMG) [5] is one instance of MDSD. One of several current questions of MDA comes with the applicability of the approach itself and the expressiveness of Unified Modeling Language (UML) [6], [7], the modeling language recommended for MDA [8]. With UML it is possible to express different aspects of a software system through different types of diagrams. Having captured the requirements these diagrams are a starting point for a software development process and support a common basis for communication and documentation [9], [10]. These models are then transformed to models with lower abstraction levels since different levels of abstraction ease the collaboration of all roles involved in the software development process [2]. Abstract models specifying, e.g. use cases, are suitable for a business or systems analyst. Later on, more detailed models are used for specification. These contain information not relevant for a business analyst but crucial for a developer or tester [9].

Yet many details cannot be specified through models as the existing model elements of UML are not accurate enough to properly capture all details [8], [11]. Therefore transformations need to be executed by error-prone manual steps which result in a conflict with the MDSD approach. Misunderstandings in the interpretation of models at least protract the underlying software development process or may even lead to unusable software. Especially in the area of user interaction this problem is significant. Early in a software development process many details concerning user interaction are available.

Addressing this problem we present an approach for model-driven development of graphical user interfaces (GUIs), carrying on with our first draft in [12]. This attempt will cover an accurate specification of GUIs through models and further demonstrate how these models can be transformed through an automated multi-stage transformation process down to source code of any platform. We therefore introduce two lightweight extensions of UML terms of two UML profiles. Furthermore we make use of Queries Views Transformations (QVT) as the language to define transformation rules as also recommended

for MDA [13].

In contrast to other related work (cf. section 2) we concentrate on the flexibility, general applicability and portability of our approach also taking different roles involved in a software development process into consideration.

Accordingly, the remainder of this paper is organized as follows: section 2 introduces the state of the art in the context of MDA focusing user interaction and user interfaces. In section 3 our extensions to UML are presented to prepare a case study demonstrating the feasibility of our approach given in section 4. A conclusion and outlook on future work in this area closes the body of this paper.

II. State Of The Art

Model-driven development and especially MDA is the subject of several other research ambitions and current discussions [4], [8], [5]. While strengths and weaknesses of MDA are still being investigated [11], there is a rising demand for the construction of flexible and well-structured user interfaces [14], [15]. One way to create those user interfaces may be achieved by usage of declarative languages. Recent projects like XML User Interface Language (XUL) [16], eXtensible Application Markup Language (XAML) [17] or Views [18] take an XML-based approach to a declarative GUI description. These languages may therefore be used as target platforms for model-driven GUI development.

A similar approach to describe user interfaces is taken by UsiXML [19]. Based on work of the Cameleon reference model [20] this approach additionally focuses on model-driven development of user interfaces. “Concur Task Tree” (CTT) diagrams mark an initial linchpin of this approach with focus on tasks that are the object to user interaction. These tasks are then transferred to an abstract, concrete and final user interface [19] through a series of XSL Transformations. The need for different levels of abstraction in GUI modeling is essential and is therefore also pursued in this paper.

Since UML is sometimes considered as the lingua franca of system modelling [10], notably in object oriented designs, some other approaches like Pinheiro da Silva et al. extend in [21], [22] the UML metamodel with new types of diagrams or new notational symbols. In [22] a language called “UML for interactive applications” (UMLi) is introduced to reflect missing capabilities for designing user interfaces with UML. In contrast to extending UML with new notational symbols, Almendros-Jimenez and Iribarne explore in [23], [24] the possibilities of UML use case and activity diagrams. They use specific action elements to reflect units that are typical for Java applets. Hence models based on their work may be used to generate GUIs within Java applets only.

Unlike focusing on Java applets, Lorenz introduces in his proposal [25] an approach to model GUIs within activity diagrams. He uses semi-formal text building blocks named “scenes” similar to UML annotations that specify details of a GUI. Additionally he introduces, similar to Petrasch [8], two different types of actions, called user respectively system action to model user and system behavior likewise. While this approach covers a clear and desirable classification of user and system actions, the “scenes” contain non-formal information such as attributes or buttons that shall be displayed. Due to its non-formal specification, this information cannot be transformed in a model-driven manner.

While Almendros-Jimenez and Iribarne suggest specializing the model notation to reflect details of their target platform (Java applets) [24], Lorenz [25] and Petrasch [8] propose more abstract notations to cover a more general approach. Yet both use one direct transformation to their target platform J2EE/Struts and do not consider the diversity of today’s platforms.

Subsuming the current state of the art there is a need for a more general and straightforward approach to model-driven development of GUIs which on the one hand allows for a tool-supported development of GUIs for any kind of target platform and on the other hand is easy to apply as based on established modeling languages also used within current software development processes. Taking [25] and [8] into account, a generalization and further adaptation of these approaches promises to be successful in leveraging the current level of software development to higher stages in automation.

In the next section we present an approach of model-driven development of GUIs based on the discussed state of the art. We aim at overcoming the main drawbacks of current approaches stated above by presenting a more general approach which is applicable to common software development processes.

III. Model-Driven Development of Graphical User Interfaces

Common software development processes share the use of graphical modeling languages utilizing models for communication and documentation purposes [10]. With the approach of model-driven software development, a new value is added to these models as they are used in a transformation process that can step down to source code. As mentioned above, due to inappropriate model elements several aspects can still only be captured in a non-formal way. With our approach we aim to specify GUI-relevant aspects through modeling. One central goal is to transform these GUI-relevant aspects to source code. This way we strive for an improved propagation of requirements from models to source code by a higher degree of automation in software development, in this case especially concerning GUI development. Hence we first introduce a mechanism to cover GUI aspects in process models like UML activity diagrams. In a development project these extended UML activity diagrams serve as our first source model. As a next step we present a GUI metamodel as a template that is usable to specify any kind of GUI. On one hand the latter is capable of taking on these GUI related aspects and on the other hand serves as target model of our first model-to-model transformation. Following MDA principles we apply a second model-to-model transformation to achieve a platform-specific GUI model. Finally we end

with a model-to-text transformation resulting in source code. The transformation mechanism used is introduced and described at the end of this section.

A. *Process Model Extension for Graphical User Interfaces*

While necessary requirements are assumed to be already captured, use cases refined by UML activity diagrams are the starting point of our approach. Activity diagrams may be used to serve as platform independent model (PIM) in terms of MDA. According to MDA this means to move on from a computation independent model (CIM) to a PIM [26]. In order to enrich activity diagrams with GUI-related aspects, it is necessary to enhance standard activity diagrams by specialized model elements. Following [25] (cf. section 2), our first extension introduces two new types of actions, which allow us to distinguish between *System-* and *UserActions*. This differentiation determines in a formal manner if an action has to be performed by the user, e.g. entering some data, or if the system itself is required to be active. The extension is attained by a UML profile, a lightweight extension mechanism of UML [6]. As UML profiles do not change the metamodel of UML itself but extend existing meta-classes the benefit of using an UML profile comes with its reusability and the availability of more precise stereotypes which may be used by any tool supporting UML profiles. Yet associations specified in a UML profile are not handed down to its instances, so one has to make use of another approach to apply constraints. This paper uses the Object Constraint Language (OCL) as an integral part of UML [6]. Since OCL expressions usually become quite lengthy, only one OCL expression is exemplified in figure 1 and explained below.

To provide further details needed to handle user actions, we specify another stereotype named *GUIInputPin*. As a specialization of a regular UML input pin, a *GUIInputPin* allocates an additional tagged value *guiType*. A tagged value is a key-value pair which attaches supplementary information to a model element [5]. In this case every instance of the *GUIInputPin* has to have a *GUIType*. All available *GUITypes* themselves are given by an enumeration also included in the UML profile. Through this enumeration modelling is restricted to known elements reasonable for usage in a GUI, secondly the application of transformation rules is guaranteed. With *GUIInputPins* all attributes that need to be provided by the user and the type of these attributes can be modelled by choosing the corresponding *GUIType*. To ensure that a *GUIInputPin* may not be applied to a *SystemAction*, there is an OCL expression in the UML profile prohibiting that. Figure 1 depicts a partial view on our UML profile named *GUIActivityProfile*. With the *GUIActivityProfile* in place it is possible to add GUI-relevant information to an activity diagram in a formal manner allowing for a tool-supported transformation.

While *GUITypes* describe what is needed, they do not comprise an assertion about concrete GUI elements. If e.g. a *GUIInputPin* is of *GUIType Text*, the corresponding GUI element could be an input field or a text area. So *GUITypes* are given as a declarative description for displayable elements. The advantage of this declarative description is twofold. First the *GUIType* does not make any restrictions on special GUI elements allowing for a variety of different GUI libraries. The precise mapping of *GUIType* to a GUI element is specified through transformation rules which we will address later. A second advantage is found in the modelling itself. If a *GUIType* is not a simple type as *Text* or *Boolean* but complex, the refinement of this data type can be moved to the next phase in the software development process using e.g. *XML Schema* [27] as *GUIType*. This avoids mixing up different architect roles which would lead to dismantle the separation of concerns principle, which is especially not desirable for larger projects [9].

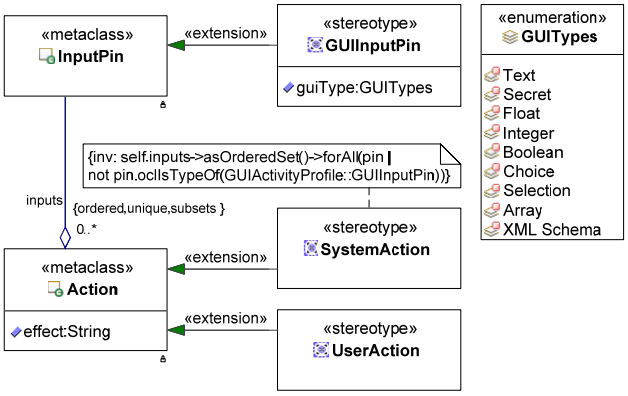


Fig. 1. Partial view on *GUIActivityProfile*

Having provided the extended activity diagrams, the next phase in the software development process can be addressed by a transformation to another PIM, the GUI model as presented in subsection B.

B. *A Metamodel for GUIs*

Basically GUIs are built up from a number of dialogue modules that are linked to each other in a specific way [15]. As a result, a GUI contains static and dynamic aspects. All displayable elements including their container element (e.g. a Web browser window) are part of these static aspects. The link structure and the way this structure is built can be regarded as the dynamic part of a GUI. Dynamic aspects can be described in additional navigational models [28] which we do not investigate further.

Since many different displayable elements build up a dialogue module [29], [15] it is necessary to provide the means to model the build-up of a single dialogue module and the complete GUI itself. We therefore introduce a second UML profile named *GUIProfile* based on a UML class diagram which supports the modeling of GUI-related development decisions. It can be used to model GUIs in a more detailed and formal way compared to standard UML. To assure independence of any GUI toolkits or libraries, the *GUIProfile* can be considered as a crosscut of many of the

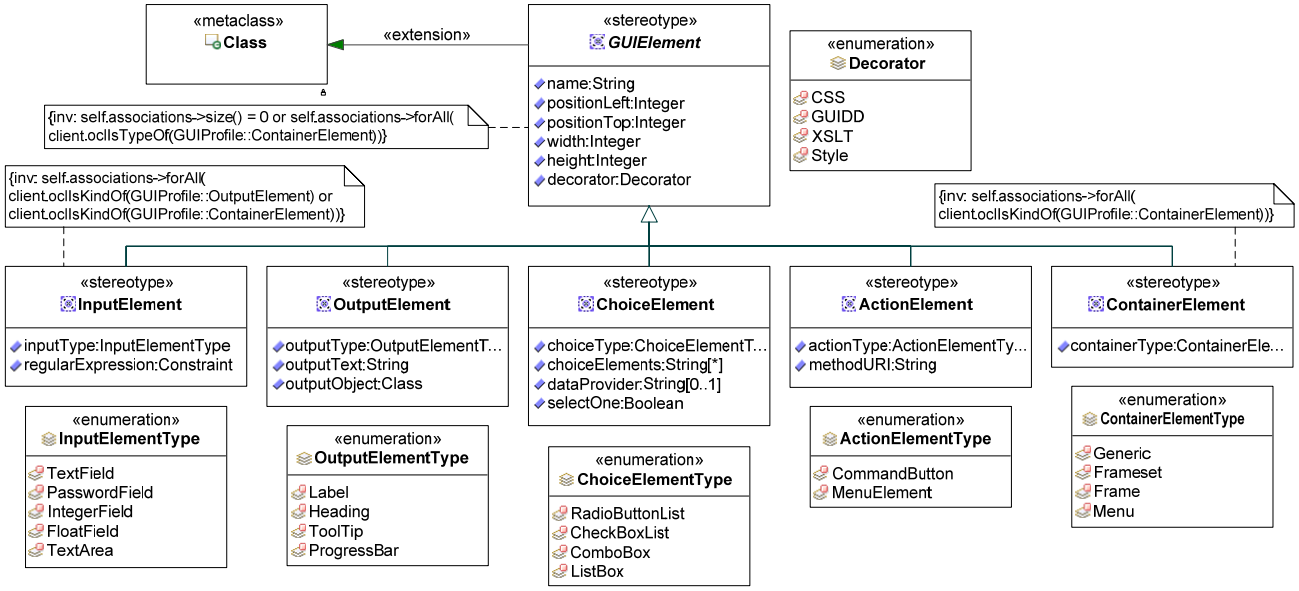


Fig. 2. GUI metamodel *GUIProfile* used for modeling the assembly of a GUI (note that not all needed OCL expressions are displayed)

most common elements contained in currently available libraries like [25]. As mentioned in [12], the *GUIProfile* is constructed to be extendable to support further elements and types that might be useful in future versions of that model, for example by adding new elements to the corresponding enumeration.

We use the *GUIProfile* as the target model for a first model-to-model transformation within which our extended activity diagram presented in subsection A is the source model. Each *GUIInputPin* of a *UserAction* is transformed to an instance of one stereotype of the *GUIProfile*. For example a *GUIInputPin* of *GUIType Text* is transformed to an *InputElement* of *InputType TextField* or *TextArea* whereas a *GUIInputPin* of *GUIType Boolean* would be transformed to a *ChoiceElement* and so on. Some of our transformation rules that do a mapping between *GUIActivityProfile* and *GUIProfile* are depicted in Figure 3. The shown QVT rules describe partially how a *UserAction* is mapped to a dialogue module. The *UserAction* is mapped to a package which symbolizes a dialogue module. The when-clause of the according mapping rule (**`action2Package`**) restricts the transformation to *UserAction*s only. In order to map all inputs to GUI elements as needed another mapping (**`guiPin2Class`**), not displayed, is called to choose the relevant model members (e.g. an *OutputElement*).

```
mapping main(in model: uml20::activities::Activity):
uml::together::Model {
 object {
  nestedPackages :=
    model.nodes.oclAsType(uml20::activities::Action)
      ->collect(act | act.action2Package())
      ->asOrderedSet();
  }
}

mapping uml20::activities::Action::action2Package():
uml::kernel::packages::Package
 when {
  self.getStereotypeInstances().
  oclIsTypeOf(GUIActivityProfile::UserAction)-> any(true)
 }
 ...
 ownedMembers += inputs->collect(pin |
                        pin.guiPin2Class());
 }
}
```

Fig. 3. Used transformation rules in QVT

The result of the first transformation is one GUI model for each *UserAction* as an instance of *GUIProfile*. In a second iteration during a design phase of the software development process the GUI model can be manually enriched with further information that is not forthcoming by the extended activity diagram. For example the correct order of each GUI element, their sizes, colours etc. have to be specified as this information is of no concern during the preceding analysis phase. After the refinement of the GUI model, a second transformation is applied. It transfers the GUI model either to a platform specific model (PSM) like a Java Swing [28] or XUL model [16] and finally source code. The whole transformation process will be exemplified in section 4.

In conclusion our approach combines several steps in a MDA process. Extended UML activity diagrams serve as linchpin, while our GUI models which are instances of the *GUIProfile* are derived automatically from these extended activity diagrams. In next iterations the GUI model is manually enriched, transformed to a platform specific GUI model and finally to its underlying source code. Distinguishing between our platform independent and a platform specific GUI model is crucial in order to keep device and platform independent as long as possible. Through this indirection we obtain reusable transformations [5] and are able to increase the degree of automation in a software development process. Changing the GUI's target platform will only result in a swap of the last set of transformation rules with another appropriate set instead of writing the source code again from scratch.

IV. Model-Driven Development of Graphical User Interfaces – Implementation Experience

In this section we present how to put our approach into practice based on a case study following a common software development process [9]. As a toolkit for our approach we chose the latest version of Borland's Together Architect [30]. We commenced by implementing the *GUIActivityProfile* and the *GUIProfile* (cf. section 3) with Together and contributed the new stereotypes to Together's palette of stereotypes. Then we implemented all necessary QVT expressions. Note that this setup has to be done only once. After this setup all new stereotypes like *GUIInputPin* are available for modeling.

A. Case Study

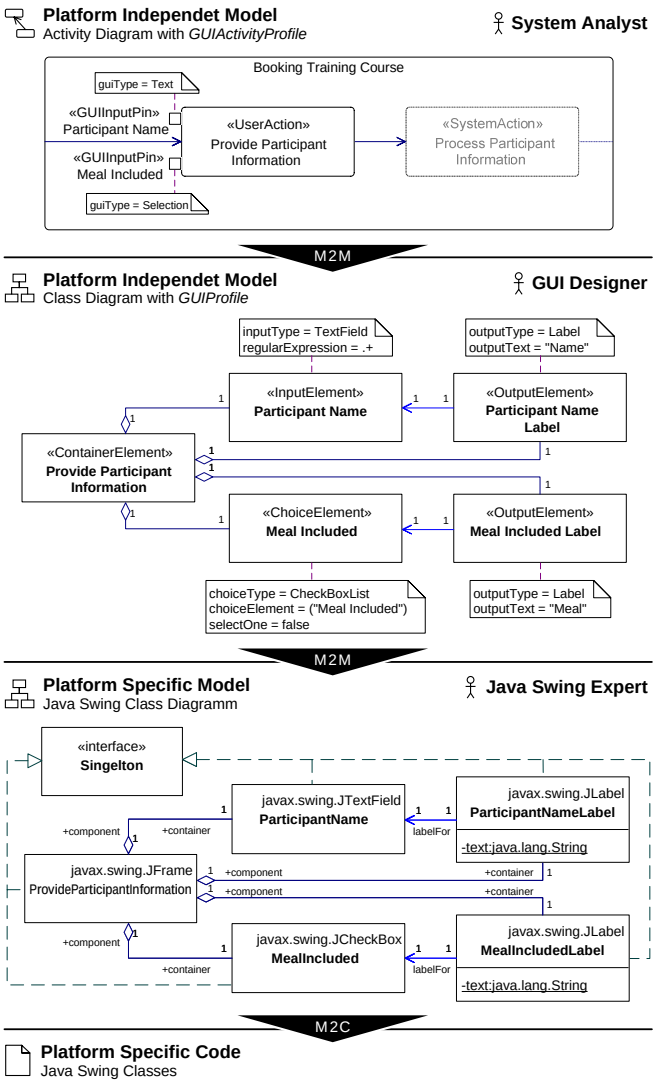
Booking a professional training course will serve as our case study. The pattern of booking is usually very similar, either using a telephone or the Web: a participant picks a desired course, checks if the course has free places, books some extras like meals and so on. In this small business process the training participant has to provide some information during the booking process. Progressive training companies allow for booking training courses via a Web front-end, some others have the training participant make a phone call and an employee enters the provided information via a local client of their training management system. In both cases a GUI is needed. We use this scenario to demonstrate how our approach easily allows for developing a GUI on the one hand for a Web front-end and on the other for a traditional client application.

B. Specifying Use Cases with Activity Diagrams

Because necessary requirements are again assumed to be already captured, we skip this phase and start with a use case named *Booking Training Course* for further investigation. With the help of the *GUIActivityProfile*, a system analyst specifies this use case by providing an activity diagram consisting of several *User*- and *SystemActions*. During the *UserAction Provide Participant Information* (cf. figure 4) the participant enters the information needed like his name, his desired course or if he wishes to have meals etc. The GUI has to provide the corresponding GUI elements so the role system analyst simply adds *GUIInputPins* to the *UserAction* and assigns them with an adequate *GUIType*. In the upper part of figure 4 we display only a small extract of the activity diagram with two *GUIInputPins*. The first is of *GUIType Text* for the participant's name and the second of *GUIType Boolean* for choosing whether he wants to have meals included or not.

C. Activity Diagram to GUI Model Transformation

Although the system analyst does provide information about the attributes the user enters, he neither cares about the design nor the layout of the corresponding GUI elements. This design task is performed by e.g. the role GUI expert during a design phase. For this purpose the activity diagram is transformed by a model-to-model transformation to the corresponding platform-independent GUI model. The role GUI expert improves this GUI model by adding information




```
public class ProvideParticipantInformation extends javax.swing.JFrame
                                           implements Singelton {

   private static final java.awt.Component[] components = {
         ParticipantNameLabel.getInstance(),
         ParticipantName.getInstance(),
         MealIncludedLabel.getInstance(),
         MealIncluded.getInstance()};

   private static ProvideParticipantInformation inst;
   public static ProvideParticipantInformation getInstance() {
      if(inst == null) inst = new ProvideParticipantInformation();
      return inst;
   }

   private ProvideParticipantInformation() throws HeadlessException {
      // add components to content pane
      ...
```

Fig. 4. Case study *Booking Training Course*

e.g. about the size of the input field *Participant Name* (c.f. figure 4) or the display order of the different input elements and so on. At this stage our GUI model is still independent of any platform or technology. Starting from the same refined and platform-independent GUI model, it is now possible to apply different sets of model-to-model transformation rules to achieve platform-specific models for Java Swing, XUL or any desired platform. Finally the platform-specific model is transformed to source code by a model-to-text transformation. Figure 4 depicts the whole approach exemplified for Java

Swing [29] as the target platform. To achieve the same GUI in XUL [16] source code we use the platform-independent GUI model, implement the transformation rules and again execute the transformation process. Finally we created two GUIs for two different platforms derived from one extended activity diagram. Although the initial effort to implement the needed UML profiles and the sets of transformation rules is not negligible the benefit of the initial complexity quickly pays off while reusing the set of transformation rules.

V. Conclusion and Future Work

In this paper we were able to demonstrate that a model-driven development of graphical user interfaces is feasible and applicable to common software development processes. The focus of this work has, in particular, been on two UML profiles enabling a tool-supported modeling of GUI-related aspects and on a multi-level approach to transform these aspects stepwise down to source code following a common software development process. Using a case study we were able to demonstrate that transformations to several target platforms are possible.

The benefit of this work comes with an added value to the modeling of graphical user interfaces. Following our approach the modeling of GUIs does not only serve the purpose of communication with the customer or for documentation during a software development process; the developed models are also used to generate source code. The usage of several models with different levels of abstraction as suggested by MDA has two major advantages. First it makes our approach applicable to any common software development process and second it gives consideration to the diversity of platforms and devices available today. Whether the GUI has to be developed for a handheld with a Java client or for a common computer with a Web browser, both GUIs can be developed using the same approach.

Our approach focuses on modeling one GUI to one user interaction. Yet another aspect to be addressed comes with the number of involved users in a business process. There are business processes involving many different users. As a next step we will pursue extending our approach of modeling user interactions to involve two or more different roles.

Furthermore there are user interactions spanning over several GUIs (like for example installation wizards) with dependencies in-between the individual GUIs. So the platform-independent modeling of navigational aspects between GUIs is a next step we want to investigate. First promising results which also already influenced our approach can be found in [28].

References

[1] P. Chowdhary, K. Bhaskaran, N. S. Caswell, H. Chang, T. Chao, S.-K. Chen et al., "Model Driven Development for Business Performance Management," *IBM Systems Journal*, vol. 45, no. 3, 2006. http://www.research.ibm.com/journal/sj/453/chowdhary.html

[2] G. Cernosek and E. Naiburg, "The Value of Modeling," *IBM developerWorks*, June 2004. http://www-128.ibm.com/developerworks/rational/library/6007.html

[3] I. Sommerville, „Software processes" in *Software Engineering*, 7th ed. Harlow, UK: Pearson Education, 2004, pt. 1, ch. 4.

[4] A. Kleppe, J. Warmer and W. Bast, *MDA Explained: The Model Driven Architecture: Practice and Promise*, Amsterdam, Netherlands: Addison-Wesley Longman, 2003.

[5] J. Mukerji and J. Miller, "MDA Guide Version 1.0.1," OMG, 2003. http://www.omg.org/cgi-bin/doc?omg/03-06-01.

[6] *Unified Modeling Language (UML), Version 2.1.1: Superstructure*, OMG Standard, 2007. http://www.omg.org/cgi-bin/doc?formal/07-02-03

[7] J. Rumbaugh, I. Jacobson and G. Booch, "The Unified Modeling Language Reference Manual," 2nd Ed., Addison-Wesley, 2004.

[8] R. Petrasch and O. Meimberg, *Model Driven Architecture – Eine praxisorientierte Einführung in die MDA*, Heidelberg, Germany: dpunkt, 2006.

[9] P. Kruchten, *The Rational Unified Process, An Introduction*, 2nd ed., Addison-Wesley, 2000.

[10] B. Hailpern and P. Tarr, "Model-driven development: The good, the bad, and the ugly," *IBM Systems Journal*, vol. 45, no. 3, 2006. http://www.research.ibm.com/journal/sj/453/hailpern.html

[11] R. France and B. Rumpe, "Model-driven Development of Complex Software: A Research Roadmap," *2007 Future of Software Engineering*, pp. 37–54.

[12] S. Link, T. Schuster, P. Hoyer and S. Abeck, "Modellgetriebene Entwicklung von Benutzerschnittstellen," *Jahrestagung der Gesellschaft für Informatik*, Bremen, Germany, 2007.

[13] *Meta Object Facility (MOF) 2.0 Query/View/Transformation (QVT) Specification, Final Adopted Specification*, OMG Standard, 2005. http://www.omg.org/cgi-bin/doc?ptc/2005-11-01

[14] A. Arsanjani, "Service-oriented modeling and architecture," *IBM developerWorks*, 2004. http://www.ibm.com/developerworks/library/ws-soa-design1/

[15] J. Bishop, "Multi-platform User Interface Construction – a Challenge for Software Engineering-in-the-Small," *Proc. 28th Int. Conf. on Software engineering*, Shanghai, China, 2006.

[16] *XML User Interface Language (XUL) 1.0*, Mozilla.org Specification. http://www.mozilla.org/projects/xul/

[17] *Extensible Application Markup Language (XAML)*, Microsoft Specification. http://msdn2.microsoft.com/en-us/library/ms747122.aspx

[18] J. Bishop and N. Horspool, "Developing principles of GUI programming using views," SIGCSE Bulletin, ACM Press, 2004

[19] F. J. Martinez-Ruiz, J. M. Arteaga, J. Vanderdonckt, J. M. Gonzalez-Calleros, R. Mendoza, "A First Draft of a Model-driven Method for Designing Graphical User Interfaces of Rich Internet Applications," *Proc. 4th Latin American Web Congr.*, 2006, pp. 32–38.

[20] G. Calvary, J. Coutaz, L. Bouillon, M. Florins, Q. Limbourg, L. Marucci et al., "The CAMELEON Reference Framework," 2003. http://giove.cnuce.cnr.it/cameleon/deliverable1_1.html

[21] P. Pinheiro da Silva, N. W. Paton, "User Interface Modelling with UML," *Proc. 10th European-Japanese Conf. on Information Modelling and Knowledge Bases*, Saariselk, Finnland, 2000.

[22] P. Pinheiro da Silva, N. W. Paton, "Improving UML Support for User Interface Design: A Metric Assessment of UMLi," *Proc. 2003 Int. Conf. on Software Engineering*.

[23] J. Almendros-Jimenez, L. Iribarne, "Describing use cases with activity charts," *Proc. 2004 Metainformatics Symposium*, pp. 141–159.

[24] J. Almendros-Jimenez, L. Iribarne, "Designing GUI components from UML Use Cases," *Proc. 12th Int. Conf. and Workshop on the Engineering of Computer Based Systems*, 2005, pp. 210–217.

[25] A. Lorenz, "Anpassung von UML-Aktivitäten an den Prozess der Webapplikationsentwicklung," *Proc. 36. Jahrestagung der Gesellschaft für Informatik*, Bremen, Germany, 2006, pp. 178–184.

[26] P. Forbrig, *Objektorientierte Softwareentwicklung mit UML*, Munich, Germany: Hanser Fachbuchverlag, 2006.

[27] *XML Schema 1.0*, W3C Recommendation, 2002. http://www.w3.org/XML/Schema

[28] N. Koch, "Transformation Techniques in the Model-Driven Development Process of UWE," *Proc. 6th Int. Conf. on Web Engineering*, Palo Alto, USA, 2006.

[29] M. Loy, R. Eckstein, D. Wood, J. Elliott and B. Cole, *Java Swing*, 2nd ed., O'Reilly, 2002.

[30] Borland Together 2006 Release 2. http://www.borland.com/us/products/together/index.html

All web references were verified on September 5th, 2007.