Model-driven UI Development integrating HCI Patterns

Enes Yigitbas
University of Paderborn
s-lab – Software Quality Lab
eyigitbas@s-lab.upb.de

Bastian Mohrmann
University of Paderborn
basti86@mail.upb.de

Stefan Sauer
University of Paderborn
s-lab – Software Quality Lab
sauer@s-lab.upb.de

ABSTRACT
An important criterion for user acceptance of interactive systems is software ergonomics. Therefore, a variety of HCI or usability patterns has been defined in the past. Although HCI patterns promise reusable best-practice solutions, the lack of formalization and effective tool support hinder their usage in a model-driven development process. To overcome this deficit, we propose a model-driven user interface development (MDUID) process that integrates HCI patterns. For showing the feasibility of our approach, we formalized and implemented a set of GUI patterns, a particular category of HCI patterns, based on IFML. We present our pattern application concept and our tool-support based on a customized MDUID process for generating rich internet applications (RIAs).

Author Keywords
HCI, Model-driven UI Development, Pattern, GUI Pattern

ACM Classification Keywords
H.5: Information interfaces and presentation (e.g., HCI): H.5.2: User Interfaces.

INTRODUCTION
An important criterion for user acceptance and user experience, particularly in the context of interactive systems, is software ergonomics. Therefore, a variety of HCI and usability patterns has been defined in the past [1]. Similar to software development patterns, HCI patterns are reusable best-practice solutions. The difference is that HCI patterns address the usability domain and the improvement of software ergonomics rather than general software architecture or code structure. One particular category of HCI patterns are GUI patterns. In [2] GUI patterns are described as patterns that "specify one or more abstract interaction objects, their relationships, and their interactive behavior" and that these patterns "are primarily aimed at good usability". The integration of GUI patterns in the MDUID process appears to be a promising way to overcome the lack of usability of automatically generated user interfaces. However, this solution entails two problems.

Workshop on Large-scale and model-based Interactive Systems: Approaches and Challenges, June 23 2015, Duisburg, Germany.

The first is that HCI patterns are mostly described informally in practice (1). However, model-driven approaches are based on formalisms like MOF meta-models or XML schemes. These formalisms are needed for automatized model-to-model and model-to-code transformations. The second problem is that there is barely no tool support for applying or instantiating HCI patterns, particularly GUI patterns in practice (2). In [3] it is reasoned that the lack of tools "hinders the use of HCI patterns within fully automated processes", like the MDUID approach.

In this work, we design and implement a customized MDUID process that integrates GUI patterns. The remainder of this paper is structured as following: First, we describe related work in the area of MDUID and HCI pattern integration approaches. Then we present our GUI pattern catalog and its formalization based on the abstract user interface language IFML. Afterwards we explain the implementation of our approach and the corresponding tool-support. In the end, we conclude our own contributions and outline future research activities.

RELATED WORK
Focusing on the topic of model-driven UI development (MDUID) integrating HCI patterns, multiple aspects have to be taken into account. Therefore our work is related to and influenced by a broad range of research fields in order to overcome the gap between HCI and MDUID. In the following we will briefly sum up existing MDUID approaches and pattern integration approaches and set them in relation to our own solution.

MDUID Approaches
MDUID brings together two subareas of software development, which are model-driven development (MDD) and user interface development (UID). The core idea behind MDUID is to automatize the development process of UI development by making the models the primary artifact in the development process rather than application code. An MDUID process usually involves multiple UI models on different levels of abstractions that are stepwise transformed to the final user interfaces by model transformations.

The CAMELEON Reference Framework (CRF) [4] provides a unified reference framework for MDUID differentiating between the abstraction levels Task & Concept, Abstract User Interface (AUI), Concrete User Interface (CUI) and Final User Interface (FUI).

There are various state-of-the-art modeling languages for covering the different abstraction levels of the CRF. For example MARIA XML (Model-based lAnguage foR Interactive Applications) [5] and IFML (Interaction Flow Modeling Language) [6] provide both an AUI modeling language and a tool-support to create and edit AUI models. Based on these AUI models further transformations can be performed to transform them into platform-specific CUI models which eventually are needed for generating the final user interfaces (FUI). The described MDUID approaches enable the specification and also support the generation of UIs, but they do not offer explicit mechanisms for specifying HCI patterns like GUI patterns. Therefore the existing MDUID tools show a lack of pattern formalization, instantiation and tight integration in the development process.

Pattern Integration Approaches

Engel [7] presents the concept of the PaMGIS (Pattern-Based Modeling and Generation of Interactive Systems) framework for pattern-based modeling. The PaMGIS framework combines model-based and pattern-based approaches on different levels of abstraction. The core component of the framework is the pattern repository, a collection of ``different types of patterns and pattern languages". Within the repository, the patterns are described by the PPSL (PaMGIS Pattern Specification Language). Beside the definition of HCI patterns, their meaning, their idea etc., PPSL also provides means to define relations between pattern models and other models. Such relations contain information about the particular pattern, the related FUI, (hierarchical) relationships to other patterns and back links to other object-oriented models, e.g. an AUI or CUI model of the interactive system. This information is necessary for model-to-model and model-to-code transformations. However, the PaMGIS approach leaves two issues open. First, it does not become completely clear if the mentioned model-to-code transformation can be defined on the model level or has to be defined for each instance over and over again. Secondly, no concepts for data binding have been discussed in this approach.

Radeke [8] proposes in his work a pattern application framework that describes a general concept of how patterns can be integrated in model-based approaches. This framework relies on three phases. In the first phase the user selects the pattern from the pattern repository that he wants to apply. The pattern repository contains hierarchically structured patterns and sub-patterns defined in a common pattern language. The generic part of the pattern is instantiated in the pattern instantiation phase with regard to the context of use. The outcome is an instantiated pattern that can be integrated in the development process. Although this approach suggests an interesting pattern instantiation concept, it integrates HCI patterns in a model-based rather than model-driven way. We overcome this deficit in our approach through a tight integration of the formalized GUI patterns by representing them in automatic model transformations.

PATTERN INTEGRATION CONCEPT

In order to overcome the previously mentioned problems (1) and (2), a general concept for integrating patterns in MDUID was developed that aims at increasing the usability of generated user interfaces. The main goal of this concept is the automatized application of GUI patterns within a model-driven process. Therefore, the CRF was extended by *instantiation parameters* and *application conditions* of GUI patterns like depicted in figure 1. Let us start with a short explanation concerning these two terms.

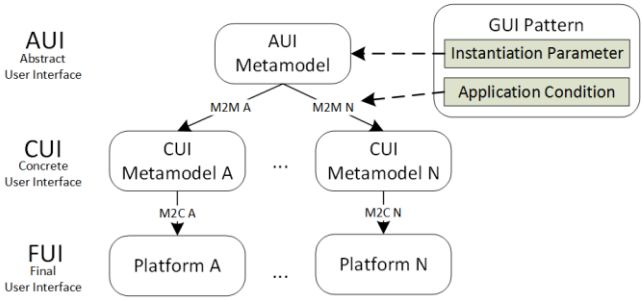

Figure 1. Overview of the pattern integration concept

Following the concepts of Wendler [12] and Radeke et al. [8], GUI patterns consist of a static and a dynamic part. The static part of a pattern describes the core solution idea of the pattern and can contain information about navigation, user interface elements or layout. It does not change among application scenarios. The dynamic part, however, depends on the prevailing pattern application context and therefore has to be set during the user interface modelling process. Since these dynamic parts determine the instantiation of a pattern, Wendler defines them as the instantiation parameters. The second important aspect is given by the conditions under which a pattern is advisable. In order to decide, when which pattern shall be applied, so-called pattern application conditions are helpful. Pattern application conditions are formal and describe situations in which a specific GUI pattern is reasonable. The advantage of formalised conditions is that they can be validated automatically, e.g. in the model-driven transformation process. Such a validation determines if a pattern is applied or not. After introducing the relevant terms, we will now explain the concept.

Referring again to figure 1, the pattern integration concept based on the CRF is depicted. It contains three abstract components: An *MDUID process implementation* with its different meta-models (AUI, CUI, Platform), an *instantiation parameter* extension for the AUI meta-model, and an *application condition* extension for the model-to-model transformation. These components have to be specified when the pattern integration concept is

implemented. As explained above, instantiation parameters depend on a pattern's application context. Because of that, they have to be set during the initial user interface specification. In our case, the user interface is initially specified on the AUI layer and hence the instantiation parameters are integrated in the AUI meta-model by additional types and/or features. The application conditions are integrated in the transformation from the AUI to the CUI model by means of transformation rules. They are validated on the AUI model and therefore reusable for any target platform, like the AUI model itself. If the conditions are valid, the pattern is applied and the according platform-dependent CUI elements are generated.

GUI PATTERN CATALOG

The developed pattern integration concept was implemented for a choice of GUI patterns. Therefore, the abstract components introduced in the previous section were instantiated. The resulting customized MDUID process is depicted in figure 2. The AUI layer is realized with IFML and the model-to-model transformation is realized with an ATL [13] plugin. In order to integrate GUI patterns, a choice of GUI patterns was identified and then formalized by instantiation parameters and application conditions conforming to the extended components, the IFML meta-model and the ATL plugin. The formalized patterns are represented by the *extension* components in figure 2.

All integrated patterns were documented in a pattern catalog comprising the pattern's general meaning, its formalized instantiation parameters and application conditions. The formalisation of the instantiation parameters is described by means of an extension of IFML while the formalization of application conditions is described by means of transformation rules extending the ATL model-to-model transformation. Currently, the pattern catalog includes seven GUI patterns that were chosen based on their frequent use in interactive applications and their occurrence in pattern catalogs [1]. Further, the patterns in the pattern catalog are structured according to pattern categories taken from [9] and presented in a defined description scheme.

In the following, we want to present the *Wizard* pattern entry according to this description scheme in order to give an example of the pattern formalization:

Wizard

Description

The *Wizard* pattern is used when a user "wants to achieve a single goal but several decisions need to be made before the goal can be achieved completely" ([11]). Regarding a complex task inside a software system that is performed rather rarely and that is too long to fit into a single page, the Wizard pattern suggests to separate the complex task into several steps that are organized in a prescribed order. The user can deal with each of these steps in a discrete *mental space* and therefore has a simplified view on this task ([10] p.55).

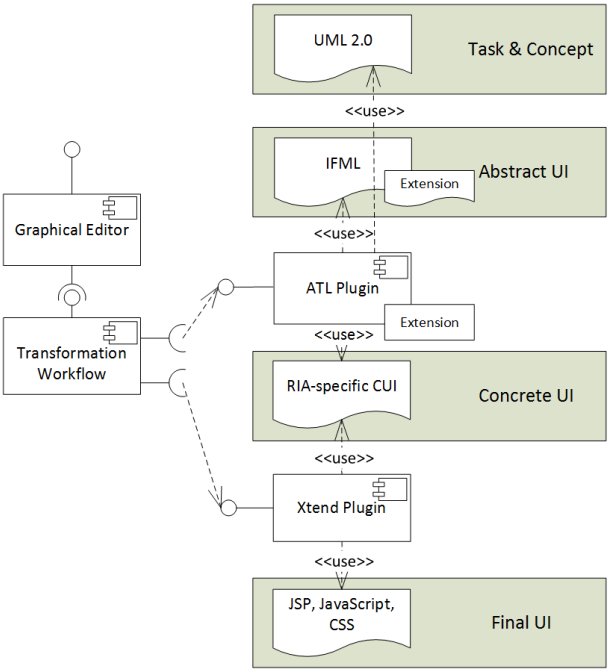

Figure 2. Architecture of the customized MDUID process

Instantiation Parameters

From the above description we can derive the following instantiation parameter when a task is separated into several decision steps: The *amount* of steps, the *order* of steps and the *content* of the particular steps. Like illustrated in figure 3, a step is formalised as a *Step* class that inherits from the *ViewContainer* class. Hence, the amount of steps and any view elements, like Events, Fields or Lists that are the content of a step can be defined. Furthermore, the inherited *outInteractionFlow* association enables the definition of *NavigationFlows* between steps and thus the order of the steps. In the related figure, the coloured classes are part of the IFML meta-model while the white class is a custom extension.

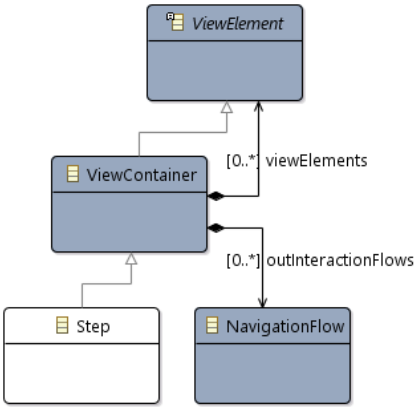

Figure 3. Simplified Wizard extension

Pattern Application Condition

The Wizard pattern is applied whenever a *ViewContainer* element with at least two containing *Steps* is modelled. All contained *Steps* must be connected with *NavigationFlows*, so their order can be determined. Below, these conditions are implemented by means of an ATL transformation rule code snippet with a source pattern and a guard.

```
from
    s : AUI!ViewContainer(s.viewElements
            ->select(child | child.oclIsTypeOf(AUI!Step)
                and child.hasStepNavigationFlow())
            ->size() > 2)
```

IMPLEMENTATION AND TOOL-SUPPORT

In this section, the implementation of the pattern integration approach and the corresponding tool-support is presented in detail. The implementation is in a state where it already could be successfully applied in an industrial setting. The architecture of the implemented approach is depicted in figure 2. This architecture partially implements the four abstraction layers (Task & Concept, AUI, CUI, FUI) of CRF indicated by the colored rectangles. The UML 2.0 language on the Task & Concept layer enables the modeling of the application's domain, e.g. by a class diagram. As can be seen, the AUI layer is realized by IFML. In particular, we reused the *IFML-metamodel.ecore*, an implementation of the *IFML standard*, which can be downloaded from the official website and extended this meta-model by a choice of specific AUI elements and GUI pattern instantiation parameters. IFML provides dedicated extension points for this purpose. We realized the CUI layer with a custom meta-model, *RIACUI.ecore*, which is specific for rich internet applications. The *RIACUI.ecore* enables to describe user interface as they are perceived by the end user including the layout, colors and concrete interaction types. On the FUI layer, the user interface is finally represented by JavaServerPages, JavaScript code and CSS style sheets. The *Transformation Workflow* component manages the model-to-model and the model-to-code transformation. As can be seen in figure 2, the model-to-model transformation is realized with ATL and produces a RIA-specific CUI model from an IFML model and the related UML 2.0 domain model. ATL provides a feature called *rule inheritance*. Rule inheritance helps to reuse transformation rules and is similar to inheritance relations in the object oriented domain. Subsequently, the model-to-code transformation, realized in Xtend [16], generates application code from a previously produced RIA-specific CUI model. The advantage of Xtend is, since it is based on Java, a statically-typed programming language which employs template-based text generation. This is particularly helpful when it comes to code generation for application code organised in different files and programming languages as it is the case for the FUI of rich internet applications.

The tool support is given by a *graphical editor* that is an extension to the IFML open source editor based on EMF [15] and the *Sirius* [14] framework. The editor is available at Github and was extended within this work by graphical representations and create/read/update/delete operations for the IFML extensions. Figure 4 depicts a screenshot of the editor showing the working area, the palette and the properties tab. This editor is an *eclipse* plug-in [17]. In the working area the current IFML model is displayed,

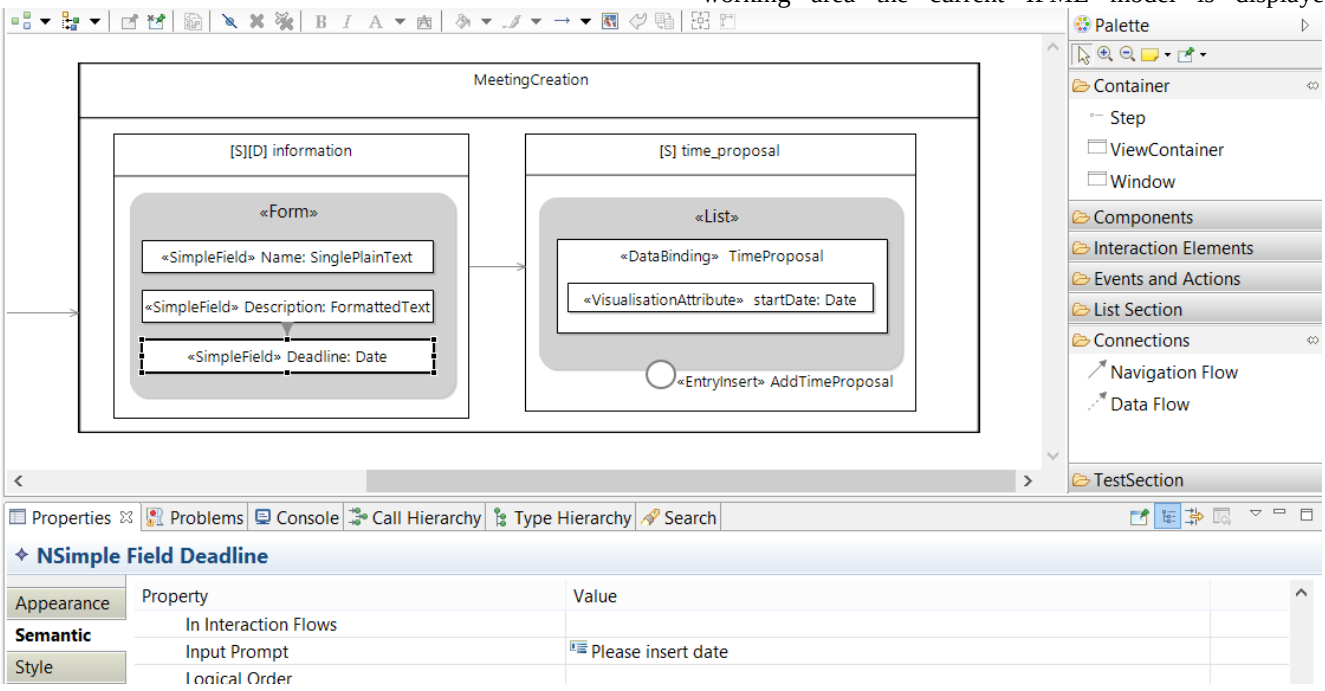

Figure 4. Screenshot of the extended IFML editor

represented in its concrete syntax. The use of meaningful icons and graphical representations helps for a better and faster understanding of the editor. The user can create new IFML model elements via Drag & Drop from the palette on the right hand side. The palette is structured in multiple sections where different V*iewElements* like *List*, *Window* and *NavigationFlow* are available. The *Step* entry in the palette also indicates that instantiation parameters of GUI patterns are configurable. The editing of IFML model elements mostly takes place in the properties tab located at the bottom. Here, all attributes and associations of model elements can be set, modified or deleted. Once such an IFML model is specified, it serves as the input of the transformation chain which can be triggered manually from the editor's context menu. The outcome is a RIA-specific CUI model in XML format and the FUI represented by multiple JavaServerPages, JavaScript files and CSS style sheets.

CONCLUSION AND OUTLOOK

In this paper, we presented the design and implementation of a customized MDUID process that integrates GUI patterns. As a basis of our solution concept we first described our general pattern integration concept. Then we presented our GUI pattern catalog and its formalization based on the abstract user interface language IFML. The feasibility of our approach was then shown by a tool-support which extends the existing IFML editor by integrated GUI patterns. The implementation of the customized MDUID process and the practical usage of the tool-support was shown in the context of generating rich internet applications (RIAs). With regard to future work we intend to evaluate our implemented solution in an industrial case study. In the evaluation we will especially focus on the influence of the integrated GUI patterns to the usability of the automatically generated RIAs.

REFERENCES

1. HCI Patterns. Retrieved April 2, 2015 from http://www.hcipatterns.org/patterns
2. Christian Märtin, Christian Herdin, and Jürgen Engel. 2013. Patterns and Models for Automated User Interface Construction – In *Search of the Missing Links, in: M. Kurosu (Ed.), Human-Computer Interaction, Part I, HCII 2013, LNCS 8004*, 401-410.
3. Kai Breiner, Marc Seissler, Gerrit Meixner, Peter Forbrig, Ahmed Seffah, and Kerstin Klöckner. 2010. PEICS: towards HCI patterns into engineering of interactive systems. In *Proc. of the 1st International Workshop on Pattern-Driven Engineering of Interactive Computing Systems* (PEICS '10).
4. Gaëlle Calvary, Joëlle Coutaz, David Thevenin, Quentin Limbourg, Laurent Bouillon, and Jean Vanderdonckt. 2003. A Unifying Reference Framework for Multi-target User Interfaces. In: *Interacting with Computers*, 289-308.
5. Fabio Paterno', Carmen Santoro, and Lucio Davide Spano. 2009. MARIA: A universal, declarative, multiple abstraction-level language for service-oriented applications in ubiquitous environments. *ACM Trans. Comput.-Hum. Interact.* 16, 4, Article 19 (November 2009), 30 pages.
6. IFML Spec. Retrieved April 2, 2015 from http://www.omg.org/spec/IFML/
7. Jürgen Engel. 2010. A model- and pattern-based approach for development of user interfaces of interactive systems. In *Proceedings of the 2nd ACM SIGCHI symposium on Engineering interactive computing systems* (EICS '10). ACM, New York, NY, USA, 337-340.
8. Frank Radeke and Peter Forbrig. 2007. Patterns in task-based modeling of user interfaces. In *Proceedings of the 6th international conference on Task models and diagrams for user interface design* (TAMODIA'07), Marco Winckler, Philippe Palanque, and Hilary Johnson (Eds.). Springer-Verlag, Berlin, Heidelberg, 184-197.
9. Marco Brambilla and Piero Fraternali. *Interaction Flow Modeling Language: Model-Driven UI Engineering of Web and Mobile Apps with IFML*. Morgan Kaufmann, 2014.
10. Jenifer Tidwell. *Designing interfaces – patterns for effective interaction design* (2. ed.). O'Reilly, 2011.
11. Matijn Van Welie. A pattern library for interaction design. Retrieved April 2, 2015 from http://www.welie.com/patterns/
12. Stefan Wendler, Danny Ammon, Ilka Philippow, and Detlef Streitferdt. A factor model capturing requirements for generative user interface patterns. In *PATTERNS 2013, the Fifth Int. Conf. on Pervasive Patterns and Applications, Valencia, Spain, IARIA*, Lecture Notes in Computer Science, pages 34–43, 2013.
13. ATL. Retrieved April 2, 2015 from https://eclipse.org/atl/
14. Sirius. Retrieved April 2, 2015 from https://eclipse.org/sirius/
15. EMF. Retrieved April 2, 2015 from https://www.eclipse.org/modeling/emf/
16. Xtend. Retrieved April 2, 2015 from https://eclipse.org/xtend/
17. Eclipse. Retrieved April 2, 2015 from https://eclipse.org/