Customized UI Development Through Context-Sensitive GUI Patterns

Enes Yigitbas, Stefan Sauer

Paderborn University, s-lab – Software Quality Lab

Abstract

Developing highly flexible and easy to use user interfaces (UIs) for software applications that satisfy different usage contexts is a challenging task. Model-driven UI development (MDUID) approaches support the generation of multiple variants of a UI for different target users, platforms and environments. However, these approaches are not always sufficient because of usability issues which require manual changes for UI customizations. To overcome this deficit, we present a MDUID process that integrates context-sensitive GUI patterns to support the development of flexible and customized UIs. The integration and situation specific application of GUI patterns enable the creation of user tailored UIs and leverage a personalized interaction. For showing the feasibility of our approach, we formalized and implemented a set of context-sensitive GUI patterns based on the Interaction Flow Modeling Language (IFML). We demonstrate our pattern application concept and tool-support based on a customized MDUID process for generating the UI of a calendar management application.

1 Introduction

An important criterion for user acceptance and user experience of interactive systems is usability. When developing multiple variants of a UI, usability issues have to be especially taken into account due to the variable contexts of use. Although MDUID approaches increase the efficiency and consistency for the generation of multiple UI variants, purely automated approaches are not optimal in terms of quality regarding the usability of the UI (Aquino et al.2010). Beside MDUID approaches a variety of HCI and usability patterns has been defined in the past (HCI Patterns).

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 (Märtin et al. 2013) GUI patterns are described as patterns that “specify one or more abstract interaction objects (AIOs), their relationships, and their interactive behavior” and that these patterns “are primarily aimed at good usability”.

Platzhalter für DOI und ggf. Copyright Text. (Bitte nicht entfernen).

Name, Vorname (2016): Titel. Tagungsband Mensch und Computer 2016. Gesellschaft für Informatik. DOI: xxxxxx

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. Also it can offer new possibilities to leverage personalized interaction, because an intelligent selection and application of GUI Patterns can help to create user tailored/specific UIs. However, this solution entails several challenges.

The first is that GUI patterns are mostly described informally in practice. However, model-driven approaches are based on formalisms like MOF metamodels or XML schemes. These formalisms are needed for automated model-to-model and model-to-code transformations. For an automated application of GUI patterns through transformation methods it is also important that application conditions for GUI patterns are defined. GUI pattern catalogs often make suggestions about pattern usage, e.g. in which situation to use which pattern. But these suggestions are in most cases informally specified so that automatic reasoning or context-sensitive GUI pattern application is not supported. The last challenge is that there is barely no tool support for applying or instantiating context-sensitive GUI patterns in practice. In (Breiner et al. 2010) it is reasoned that the lack of tools "hinders the use of patterns within fully automated processes", like the MDUID approach.

In this paper we design and implement a customized MDUID process that integrates context-sensitive GUI patterns. The remainder of this paper is structured as follows: The next section describes related work in the area of MDUID and pattern integration approaches. After that, we introduce our extended pattern integration concept. Then, we present the idea of context-sensitive GUI patterns and their formalization based on the abstract user interface language IFML. Afterwards, the implementation of our approach is described and its application is demonstrated based on a case study. Finally, we conclude our contributions and outline future research activities.

2 Related Work

In the following we will briefly sum up related background information and work on MDUID approaches as well as pattern-based approaches and set them in relation to our own solution.

2.1 MDUID Approaches

MDUID covers two subareas of software development, which are model-driven development (MDD) and user interface development (UID). The main idea behind MDUID is to automate the development process of UI development by making the models the primary artifact in the development process rather than application code. A 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) (Calvary et al. 2003) 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) (Paternò et al. 2009) and IFML (Interaction Flow Modeling Language) (IFML Specification) 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 context of use for a UI can be defined by user, platform, and environment (models). A UI can be adapted to the context of use either at development time (e.g., developing multiple UIs for different target platforms) or at runtime (i.e., context adaptive UIs). In this paper we focus on design-time adaptations/customizations by automated application of context-sensitive GUI patterns during the development process.
The described MDUID approaches enable the specification and also support the generation of UIs, but they do not offer explicit mechanisms for specifying GUI patterns and pattern application conditions based on the context. Therefore, the existing MDUID tools show a lack of pattern formalization, instantiation and tight integration in the development process.

2.2 Pattern-based Approaches

A literature study on pattern-based UI development reveals different related approaches about pattern integration and application. Engel (Engel 2010) 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 (Radeke et al. 2007) 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 context-sensitive GUI patterns by representing them in automatic model transformations.

3 Solution Concept

In order to overcome the previously mentioned challenges, 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 automated application of context-sensitive GUI patterns within a model-driven process to support customized UI development. Therefore, the CRF was extended by *instantiation parameters* and *application conditions* of context-sensitive GUI patterns like depicted in figure 1.

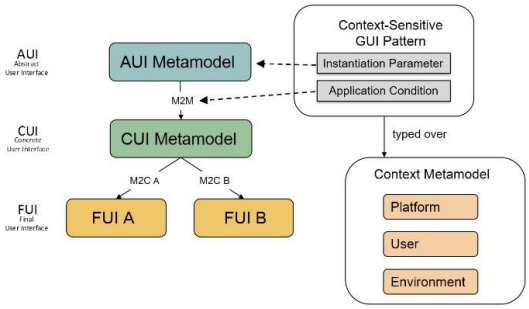

Figure 1: Integration of Context-Sensitive GUI Pattern.

Context-Sensitive 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. 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 context-sensitive GUI pattern is reasonable. The advantage of formalized 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. Figure 2 shows an overview of the pattern application workflow. After the *AUI Model Specification* step 1, where the AUI model is defined and the instantiation parameter set, the AUI model serves as input for the pattern application. In the *Pattern Application* step 2, the AUI model tree is traversed so that each abstract interaction object (AIO) is checked whether it fulfills the pattern application condition. If the pattern application condition and the contextual parameter are fulfilled, then the pattern transformation rule is applied so that the AIO is mapped to a concrete interaction object (CIO) on CUI level. Otherwise a default transformation rule is applied which means that no context-sensitive GUI pattern is applied. After introducing the pattern application workflow, we will now explain the pattern integration concept. As depicted in figure 1 the pattern integration concept is based on the CRF. It contains four abstract components: A *MDUID process implementation* with its different metamodels (AUI, CUI, and FUI), an *instantiation parameter* extension for the AUI metamodel, an *application condition* extension for the model-to-model transformation and a

Context Metamodel for specifying contextual parameter for the application of a context-sensitive GUI pattern. 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 metamodel 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.

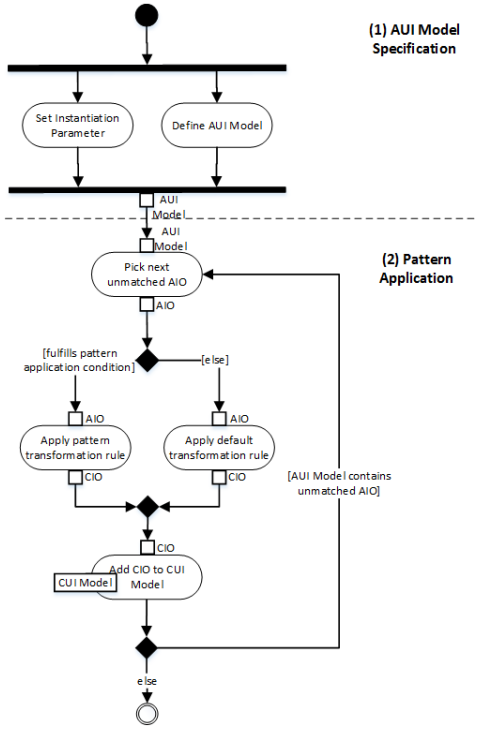

Figure 2: Pattern application workflow.

4 Context-Sensitive GUI Pattern

In order to integrate GUI patterns in the MDUID process, a choice of context-sensitive GUI patterns was identified and then formalized by instantiation parameters and application conditions conforming to the extended components, the IFML metamodel and the ATL plugin. All integrated patterns were documented in a pattern catalog comprising the pattern's general meaning, its formalized instantiation parameters and application conditions. The formalization 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 context-sensitive GUI patterns that were chosen based on their frequent use in interactive applications and their occurrence in pattern catalogs (Van Welie). Our pattern catalog consists of the following patterns:

Pattern	**Description**
Wizard	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.
Date Selector	The *Date Selector* pattern is applied when a user needs to select a date or time period.
List Entry View	The *List Entry View* pattern is used when a user needs to build a large list of typically simple items.
Input Prompt	An *Input Prompt* provides a way to make a user interface more self-explanatory by showing default values as help information.
Auto Completion	When a user types in a text field, he can be supported with suggestions arranged in a selectable list.
Primary Action	When a user shall be made aware of an important action on the user interface, the *Primary Action* pattern suggests to give this action more visual "prominence".
Object Creation	As a result of a user entering data into the user interface usually an object is created. Depending on validity of the entered data the user can be informed either with a confirmation or an error message.

Table 1: Context-Sensitive GUI Pattern Catalog.

In the following, we present the context-sensitive GUI patterns *Wizard* in more detail in order to give an example of the pattern formalization:

Context-Sensitive GUI Pattern: 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" (Van Welie). 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. This context-sensitive GUI pattern can be especially helpful in the following contexts of use: The user is a novice who has little knowledge on the application domain. A Wizard could help to guide this novice user through the application.

Formalization
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 formalized 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 metamodel while the white class is a custom extension.

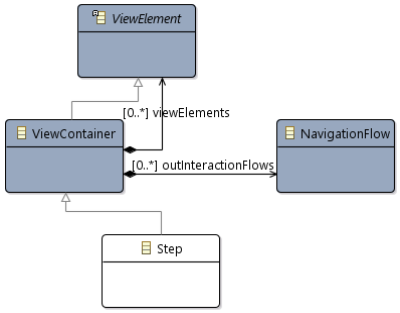

Figure 3: Wizard pattern extension.

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.

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

5 Implementation

In this section, the implementation of the pattern integration approach is presented. The architecture of the implemented approach is depicted in figure 4. This architecture partially implements the four abstraction layers (Task & Concept, AUI, CUI, and FUI) of CRF indicated by the colored rectangles. The UML 2.0 language on the *Concept* layer enables the modeling of the application's domain, e.g. by a class diagram. As can be seen, the *Abstract UI* layer is realized by IFML[1]. 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 metamodel by a choice of specific AUI elements and GUI pattern instantiation parameters. IFML provides dedicated extension points for this purpose. We realized the *Concrete UI* layer

[1] http://www.ifml.org

with a custom metamodel, *RIACUI.ecore*, which is specific for rich internet applications. The *RIACUI.ecore* enables to describe user interfaces as they are perceived by the end user including the layout, colors and concrete interaction types. On the *Final UI* layer, the user interface was finally represented by two different target FUIs: JavaServerPages, JavaScript code and CSS style sheets for rich internet applications and Java for standalone applications. The *Transformation Workflow* component manages the model-to-model and the model-to-code transformation. As can be seen in figure 4, the model-to-model transformation is realized with ATL[2] 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[3], 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 organized in different files and programming languages as it is the case for the FUI of rich internet applications.

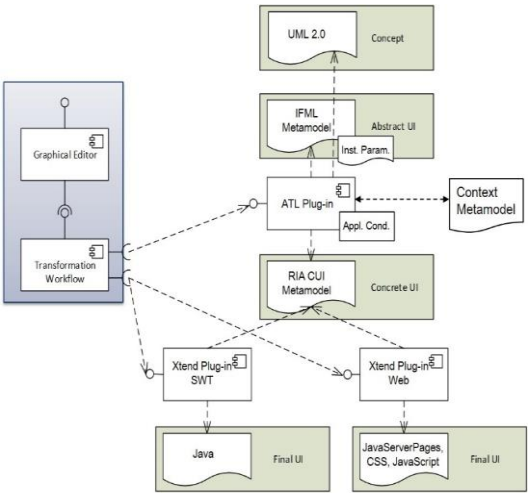

Figure 4: Architectural Overview.

6 Case-Study

For demonstrating the practicability of the implemented MDUID process, we have applied it for the modeling, transformation and generation of a calendar management application's UI that supports the creation, management and analysis of meetings. First we have specified an

[2] https://eclipse.org/atl

[3] http://www.eclipse.org/xtend

abstract UI model (see Figure 5) for the calendar management application using our extended IFML Graphical Editor that is based on the open source IFML Editor.

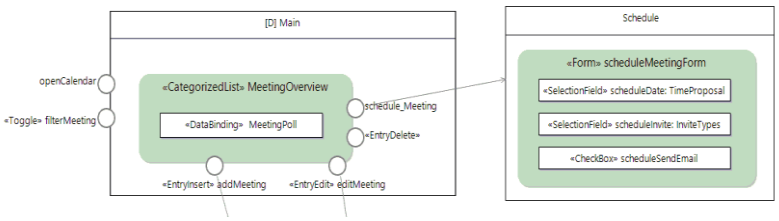

Figure 5: Excerpt of the Calendar Management abstract UI.

In the second step, the automated transformation workflow was executed on the specified abstract UI model. The transformation starts with the model-to-model transformation realized by the ATL plug-in. During the model-to-model transformation, several pattern transformation rules were applied. The result of the model-to-model transformation is a CUI model that serves as input for the subsequent model-to-code transformation. Finally, the FUI representation was generated based on the CUI model. During the generation step different Xtend templates are used to transform the CUI model elements to corresponding code snippets in the target technology. An excerpt of the resulting final user interfaces for the Java target language are shown in figure 6. Here, we can see that several patterns like the *Wizard*, *Date Selector* or *List Entry View* pattern were successfully applied during the generation of the final user interfaces of the calendar management application.

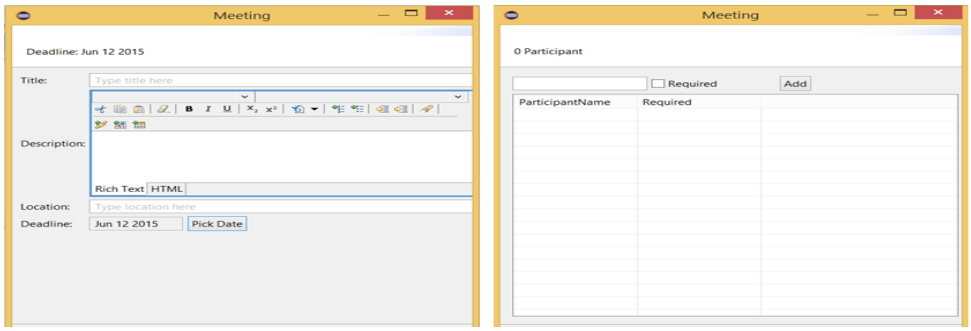

Figure 6: Screenshot excerpts of the generated UI for the calendar management application where different GUI patterns were automatically applied during the transformation process.

7 Conclusion and Outlook

In this paper, we presented the design and implementation of a MDUID process that integrates context-sensitive GUI patterns to support customized UI development. As a basis of our solution concept we first described our general pattern integration concept. Then, we presented the idea of context-sensitive GUI pattern 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 based on a case study where we generated the UI of a calendar management application. As an ongoing task we are working on the evaluation of our implemented solution in an industrial case study with real end-users. In the evaluation we will especially focus on the influence of the integrated context-sensitive GUI patterns to the usability of the automatically generated UIs. Furthermore, we plan to extend our catalog by more context-sensitive GUI patterns on different abstraction levels (e.g. CUI level) that support different target platforms (e.g. mobile). With regard to future work we intend to extend the application rules for GUI patterns to dynamic aspects, so that situation specific pattern application is supported at runtime.

8 References

Aquino, N.et al.: Usability evaluation of multi-device/platform user interfaces generated by model-driven engineering. In *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement* (ESEM '10).

Breiner, K. et al.: 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).

Calvary, G. et al. (2003): A Unifying Reference Framework for Multi-target User Interfaces. In: *Interacting with Computers*, 289-308.

Engel, J.: A model- and pattern-based approach for development of user interfaces of interactive systems. In *Proc. of the 2nd ACM SIGCHI symposium on Engineering interactive computing systems* (EICS '10).

HCI Patterns, http://www.hcipatterns.org/patterns

IFML Specification, http://www.omg.org/spec/IFML

Märtin, C. et al.: Patterns and models for automated user interface construction: in search of the missing links. In *Proc. of the 15th int. conference on Human-Computer Interaction: human-centred design approaches, methods, tools, and environments - Volume Part I* (HCI'13), Masaaki Kurosu (Ed.), Vol. Part I.

Paternò, F. et al.: 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).

Radeke, F. et al.: Patterns in task-based modeling of user interfaces. In *Proc. of the 6th international conference on Task models and diagrams for user interface design* (TAMODIA'07), Marco Winckler, Philippe Palanque, and Hilary Johnson (Eds.).

Van Welie, M.: A pattern library for interaction design. http://www.welie.com/patterns