Model-based UI Modernization: From Legacy UIs to Self-adaptive UIs

Ivan Jovanovikj, Enes Yigitbas, Stefan Sauer
Software Innovation Lab, Paderborn University, Germany
Zukunftsmeile 1, 33102 Paderborn
{ivan.jovanovikj|enes.yigitbas|stefan.sauer}@upb.de

Abstract

Due to constantly changing conditions, either business-driven or legal-driven, software systems often need to be changed or adapted to a new environment. Such a software modernization process of a legacy system has to address three main aspects: data access, business logic and user interface. As the user interfaces of interactive systems become increasingly complex due to new interaction paradigms, required adaptability, use of innovative technologies, multi-media, and interaction modalities, the topic of UI modernization demands for sophisticated processes and methods to systematically transform the UI of a legacy software system to a more flexible UI in the target platform. In this paper, we present a model-based UI modernization approach for enhancing legacy UIs towards self-adaptive UIs that are able to automatically adapt to the context-of-use at runtime.

1 Introduction

The user interface (UI) is a key component of any interactive software application and is crucial for the acceptance of the application as a whole. However, a UI is not independent from its context-of-use, which is defined in terms of the user, platform and environment [1]. As today's UIs of interactive systems become increasingly complex since many heterogeneous contexts of use have to be supported, it is no longer sufficient to provide a single "one-size-fits-all" UI. However, most of the existing software systems are providing UIs that are static "one-size-fits-all" UIs. In some cases they consider manually adaptable UIs for personalization purposes or device sensible UIs based on the widespread paradigm of responsive web design in the context of web. Nonetheless, existing software systems, do not incorporate self-adaptive UIs that can automatically react to context changes regarding user characteristics (e.g., age, role, skills, preferences etc.), platform characteristics (e.g., screen size, resolution, sensors etc.) and environmental factors (e.g., light, loudness, weather etc.) by adapting the UI through layout, navigation, task-feature set minimization/maximization changes at runtime.

In our previous work, we have already addressed the problem of model-driven engineering of self-adaptive UIs [2]. In this context, we have presented different complementary domain specific languages (ContextML and AdaptML) to OMGs UI modeling language IFML[1], which support the specification of various context-of-use situations and UI adaptation rules.

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

While this existing approach primarily focuses on the forward engineering process of self-adaptive UIs, it does not consider existing legacy UIs. Therefore, to close the gap and support an automatic transition of non-adaptive legacy UIs to self-adaptive UIs, in this paper we address the reverse engineering process to enable the extraction of IFML models from legacy UIs. Once the IFML model is extracted for legacy UIs, we restructure and enrich it with self-adaptivity features by applying our existing forward engineering approach for self-adaptive UIs [2].

2 Challenges in UI Modernization

Figure 1 depicts the general solution idea for a model-based UI modernization process of *Legacy UIs* towards a *Final UI* which is self-adaptive. In the following, we discuss the related challenges in tackling this complex task of UI modernization.

- Challenges in Reverse-Engineering: Based on the *Legacy UI* code base, a user interface model has to be discovered in terms of a *Legacy Concrete UI Model*. As the extracted *Legacy Concrete UI Model* contains platform specific details, a further abstraction is needed to obtain an *Abstract UI Model*.
- Challenges in Restructuring: Based on the *Abstract UI Model*, a platform-independent description of the UI, further enrichments and changes regarding content, structure and navigation of the UI can be pursued. Moreover, it is possible to abstract to a *Tasks&Concepts Model* from the *Abstract UI Model* by applying *Tasks&Concepts Recovery* to adjust the task-feature set of the envisioned UI. These changes have to be reflected back to the *Abstract UI Model* when applying *UI Derivation*.

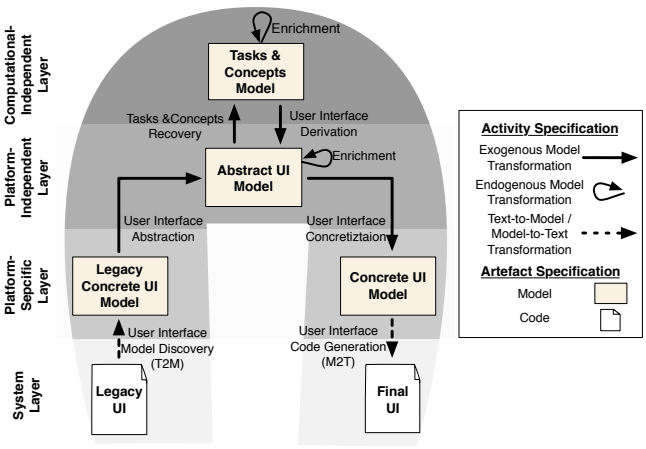

Figure 1: Model-based UI modernization approach

The restructured *Abstract UI Model* is then transformed to a *Concrete UI Model* based on which the *Final UI* for a specific target platform is generated. Beside UI code generation, the forward engineering approach requires also the generation of code for context services (components to observe context-of-use properties mainly through context sensors) and adaptation services (components to realize adaptation logic for runtime UI adaptation) for delivering self-adaptive UIs (for further details the interested reader may refer to [2]).

3 Solution Idea

For tackling the described challenges of model-based UI modernization, we envision a concrete solution idea in the context of web UIs. Nowadays, there is still a huge amount of web applications around providing web UIs, which are using outdated web technologies and UI frameworks, thus suffering from usability and maintainability problems.

Figure 2 shows an instantiation of our model-based UI modernization approach for web UIs. As a starting point, we consider legacy web UIs based on older web-technologies (HTML 1.0, JS 1.0, CSS 1.0). At that point, those web UIs are not providing self-adaptivity features, meaning that they are not able to automatically adapt to the dynamically changing context-of-use parameters. By using an existing JavaScript *Parser*[2] (*T2M*), we extract an abstract syntax tree as an intermediate representation. This is used as a basis for obtaining a platform-specific model of the legacy UI which in our case is represented as a rich internet application model (*RIA Model*)[3]. We use a *RIA Model*, as it is well-suited for representing web applications and especially web UIs. The extraction of abstract UI models based on the RIA model is established through a model-to-model (*M2M*) transformation, where relevant UI model elements from the RIA model are mapped to specific representations in the IFML model. The *M2M* transformations, specified by a *Software Developer* are executed by a *Model-Transformation Engine*. Compared to the general approach in Figure 1, we have decided to exclude the abstraction to the *Tasks&Concepts Model* in the instantiation for the sake of simplicity.

After extracting an *IFML Model*, the restructuring of the legacy UI begins. In this step, the *Web/UI Designer* can make adjustments on the IFML Model by manipulating the structure, content and navigation of the UI. In addition to that, the *Web/UI Designer* is able to specify a context model (*CML Model*) characterizing various potential context-of-use situations and an adaptation model (*AML Model*) characterizing UI adaptation rules that will be triggered at runtime through a rule-based execution environment to react to the possible context-of-use changes. The *CML Model* and the *AML Model* are instances of our domain specific languages ContextML[2] and AdaptML[2] respectively.

[2]https://github.com/acornjs/acorn

[3]https://uwe.pst.ifi.lmu.de/publications/maewa_rias_report.pdf

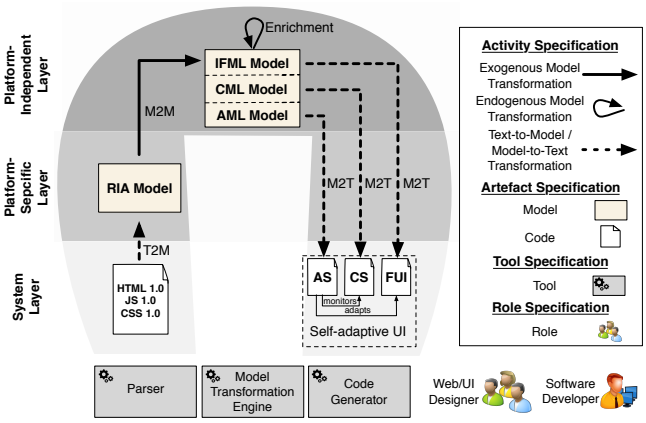

Figure 2: Modernization of web UIs towards self-adaptive UIs

In the forward engineering process, our approach provides specific code generators to generate the final UI (*FUI*), context service (*CS*), and adaptation service (*AS*). Please note, that we generate code directly from the platform-independent layer in the instantiated approach as we focus on web UIs that may be run as a browser-based application on different platforms. The code generators realize a model-to-text (*M2T*) transformation and are mainly template based. The generated artefacts of the *Self-adaptive UI* are based on Angular 2 and TypeScript. At runtime, we have an interplay between the components *AS*, *CS* and *FUI*. The generated *FUI* consists of an HTML template, which is used to render the UI in the browser, and an Angular 2 component, which is implemented in TypeScript and manages the view. Likewise, the *AS* is generated as Angular 2 service and is also implemented in TypeScript. The *AS* uses Nools, a JavaScript based rule engine, for monitoring the context information provided by the *CS*. At runtime, the *AS* monitors the context information and executes those adaptation rules whose conditions are satisfied.

4 Conclusion and Future Work

In this paper, we discussed the challenges and ideas for a model-based UI modernization approach. Such an approach enhances legacy UIs towards self-adaptive UIs that are able to automatically adapt to the context-of-use at runtime. Future work will cover the improvement of our approach regarding efficiency and effectiveness in extracting IFML models from legacy UI code and its application for diverse legacy interactive software systems.

References

[1] G. Calvary et al. A unifying reference framework for multi-target user interfaces. *INTERACTING WITH COMPUTERS*, 2003.

[2] E. Yigitbas et al. Adapt-UI: An IDE supporting model-driven development of Self-adaptive UIs. In *EICS 2017*.