Improving the Flexibility of Model Transformations in the Model-Based Development of Interactive Systems

Christian Wiehr[1], Nathalie Aquino[2], Kai Breiner[1], Marc Seissler[3], Gerrit Meixner[4]

[1]University of Kaiserslautern, Software Engineering Research Group,
Gottlieb-Daimler Str. 42, 67663 Kaiserslautern, Germany, {c_wiehr, Breiner}@cs.uni-kl.de
[2]Research Center on Software Production Methods, Universitat Politècnica de València,
Camino de Vera s/n, 46022 Valencia, Spain, naquino@pros.upv.es
[3]University of Kaiserslautern, Center for Human-Machine-Interaction,
Gottlieb-Daimler Str. 42, 67663 Kaiserslautern, Germany, Marc.Seissler@mv.uni-kl.de
[4]German Research Center for Artificial Intelligence (DFKI),
Trippstadter Str. 122, 67663 Kaiserslautern, Germany, Gerrit.Meixner@dfki.de

Abstract. This paper presents an approach that adds flexibility in the varieties of user interfaces that can be generated by processes of model-based user interface development. This approach is used at design time. Ideas from this approach have been extended for use at runtime and have been applied to SmartMote, a universal interaction device for industrial environments.

1 Introduction

Developers are faced with the problem of having to build user interfaces (UIs) for a plethora of target devices and usage situations. Model-based user interface development (MBUID) methodologies promise to reduce the complexity of this problem by leveraging different layers of abstraction, the respective models for expressing aspects of UIs on these levels, and transformation tools for the development and semi-automatic generation of UIs. Frameworks, such as the Cameleon Reference Framework (CRF) [6], have been proposed, which define the different abstraction layers and models to be used for the systematic, user-centered design of multi-target and context-aware UIs.

However, the model-based development of context-aware and runtime adaptive UIs still presents relevant challenges regarding model transformations. When an approach uses a transformation language/model to specify transformations, we say that it has *explicit* transformation logic. In concepts like DynaMo-AID [7] and MASP [4], model transformations and adaptations during runtime are implemented in the underlying renderers, that is, using *implicit* transformation logic. While the MARIA language [9] uses XSLT for the model transformations, it is unclear how the model adaptation during runtime is specified in this concept.

Approaches with implicit transformation logic have limitations regarding the diversity of user interfaces that can be generated and they might require manual modifications on the generated code in order to deal with UI requirements that are not supported by the automated transformation process. Approaches with explicit

transformation logic can be considered complex to be used by UI designers and there is a lack of suitable support for manipulation of models at runtime. Therefore, new transformation mechanisms are needed that support the explicit specification of model transformations and manipulations in order to increase the flexibility of today's engineering processes for context-aware UIs.

In Section 2, we present the Transformation Templates approach, which serves as the initial mapping concept for our transformation approach. In Section 3, we extend our runtime generation approach to improve the flexibility of its transformation process. In Section 4, we discuss results and conclude.

2 Flexibility at Design Time: Transformation Templates

A Transformation Template (TT) [2] aims to explicitly specify the structure, layout, and style of a UI according to the preferences and requirements of the end users as well as in line with the different hardware and software computing platforms and environments in which the UI will be used, i.e., different contexts of use.

A *parameter type* represents a design or presentation option. Defining a parameter type subsumes specifying the types of elements of a UI model that are affected by it, as well as its *value type*, which refers to a specific data type or to an enumeration of possible values. A parameter type can be implemented for different contexts of use and usability guidelines are provided in order to support the selection of suitable values by UI designers in different situations. A TT gathers a set of parameters for a specific context of use. Each *parameter* corresponds to a parameter type and has both a value and a selector. The *value* of a parameter corresponds to a possible value of the corresponding parameter type. A *selector* delimits the set of UI elements that are affected by the value of a parameter. We have defined different types of selectors that allow the designer to choose different sets of UI elements.

TTs are used to parameterize model-to-model or model-to-code transformations. A model compiler takes the source UI model and a TT as input. The values and selectors of the parameters of the TT specify how to transform the source UI model into the target UI model. It is important to note that TTs do not replace any implicit transformation logic or explicit transformation languages; instead, they provide a higher-level tier for UI designers to easily specify UI transformations at design time.

TTs add flexibility in MBUID approaches because they externalize the design knowledge and presentation guidelines and make them customizable according to the characteristics of the project being carried out. TTs can then be reused in other projects with similar characteristics. Furthermore, TTs aim to diversify the kinds of UIs that a MBUID approach can generate.

3 Using TTs to Increase Flexibility at Runtime

In our MBUID approach for runtime adaptive systems, three core models can be identified (see Fig. 1): 1) the Useware Markup Language (useML) [8], which is used to structure the user's tasks; 2) the Dialog and Interface Specification Language

(DISL) [3], which is used for describing the dialog behavior of the UI; and 3) the User Interface Markup Language (UIML) [1], which is used to define how the content is presented to the user in terms of concrete interaction objects and their layout.

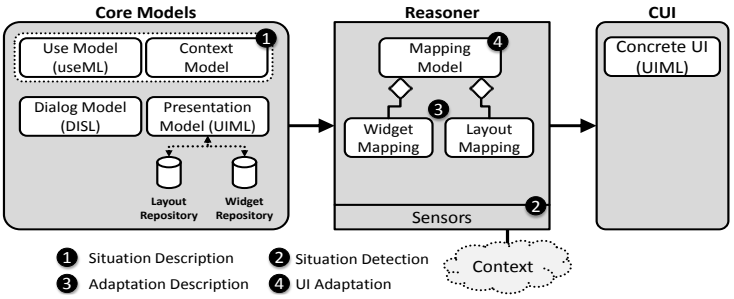

Fig. 1. Transformation process in our MBUID approach for runtime adaptive systems

These models together with runtime transformation specifications coded in the generator software (implicit transformations) were enough to obtain functional but basic UIs for runtime adaptive systems. In order to improve the flexibility of model transformations in our approach, we took the idea of the TTs as an initial mapping concept and we extended it to refer to dynamic, runtime model data, so the UI can be automatically generated and react to adjusted models.

Therefore, a *context model* has been integrated to provide access to static context information (such as information about the user or environment that may have a direct influence on the interaction) (see Fig. 1). A *mapping model* with mapping rules was also integrated as an extension to TTs (see Fig. 1) to refer to model data in addition to the fixed values of the TTs. It is composed of *layout mappings*, which allow the structure of the UI to be defined according to a hierarchical structure of UI containers and widgets; and *widget mappings*, which allow the mappings from abstract interaction objects to concrete widgets to be described. Further, in order to support the transformation process in terms of reuse and performance, a repository containing frequently occurring widgets (*widget repository*) and container-widget configurations (*layout repository*) was integrated (see Fig. 1), that can be extended easily. Both, the containers and the widgets are specified using UIML – featuring parameters to customize their presentation. At runtime, these parameters are set according to the mapping rules, using either fixed values or references to model data. All of the transformation specification previously coded in the generator software can be formalized using the mappings, which guarantee by definition that the generation will be successful. In order to demonstrate the feasibility of our approach, we developed a functional prototype as an extension to the SmartMote approach [5].

4 Discussion and Conclusion

In this paper, we presented a new mapping concept for improving the flexibility of an approach for developing context-aware UIs on the basis of UI models according to the levels of abstraction in the CRF.

For the development of such UIs, the underlying models have to be automatically manipulated during runtime. The TT approach can serve as the underlying concept, but it had to be extended. An extended TT concept was developed that supports the interlinking and mapping of different models during runtime. Mapping rules can use model values as input and manipulate the UI generation process during runtime. To test the feasibility of this concept, a first prototype was developed.

Acknowledgments. This work has been developed with the support of MICINN under the project PROS-Req (TIN2010-19130-C02-02), and GVA under the project ORCA (PROMETEO/2009/015) and the BFPI/2008/209 grant, and co-financed with ERDF. We also acknowledge the support of the ITEA2 Call 3 UsiXML project under reference 20080026. Parts of the presented work are result of the GaBi project funded by the German Research Foundation (DFG) which is part of the AmSys research focus at the University of Kaiserslautern funded by the Research Initiate Rhineland-Palatinate.

References

1. Abrams, M., Phanouriou, C., Batongbacal, A.L., Williams, S.M., Shuster, J.E. UIML: An Appliance-Independent XML User Interface Language. 31, 1695-1708 (1999).
2. Aquino N, Vanderdonckt J, Pastor O (2010) Transformation templates: adding flexibility to model-driven engineering of user interfaces. Proceedings of the 25th ACM symposium on applied computing, SAC 2010, Sierre, March 2010. ACM Press, New York, pp. 1195–1202.
3. Bleul, S., Schäfer, R., Müller, W.: Multimodal Dialog Description for Mobile Devices. Gehalten auf der Workshop on XML-based User Interface Description Languages at AVI 2004 , Gallipoli, Italy (2004).
4. Blumendorf, M., Feuerstack, S., Albayrak, S. Multimodal user interfaces for smart environments: the multi-access service platform. Proceedings of the working conference on Advanced visual interfaces. pp. 478-479 ACM, Napoli, Italy (2008).
5. Breiner, K., et al., 2011. Automatic adaptation of user workflows within model-based user interface generation during runtime on the example of the SmartMote. In Proceedings of the 15th International Conference on Human-Computer Interaction (HCII 2010), Orlando, FL.
6. Calvary G, et. al. (2003) A unifying reference framework for multi-target user interfaces. Interact Comput 15(3):289–308.
7. Clerckx, T., Luyten, K., Coninx, K. DynaMo-AID: a Design Process and a Runtime Architecture for Dynamic Model-Based User Interface Development. In The 9th IFIP Working Conference on Engineering for Human-Computer Interaction Jointly with the 11th International Workshop on Design, Specification and Verification of Interactive Systems. pp. 11–13 Springer-Verlag (2004).
8. Meixner, G., Seissler, M., Breiner, K. Model-Driven Useware Engineering. In Hußmann, H., Meixner, G., Zühlke, D. (eds.): Model-Driven Development of Advanced User Interfaces. 1-26 Springer, Heidelberg (2011).
9. Paternó, F., Santoro, C., Spano, L.D. MARIA: A universal, declarative, multiple abstraction-level language for service-oriented applications in ubiquitous environments. ACM Trans. Comput.-Hum. Interact. 16, 1-30 (2009).