Patterns and Models for Automated User Interface Construction – In Search of the Missing Links

Christian Märtin, Christian Herdin, and Jürgen Engel

Augsburg University of Applied Sciences
Faculty of Computer Science
Automation in Usability Engineering Group
An der Hochschule 1
86161 Augsburg, Germany
{Christian.Maertin,Christian.Herdin,
Juergen.Engel}@hs-augsburg.de

Abstract. This paper starts with an analysis of current or proven model and pattern-based user interface development methods and techniques. It discusses how these approaches facilitate the construction process and enhance the overall flexibility, usability and user experience of the resulting software. It is shown that HCI patterns meanwhile can contribute heavily to all development aspects of interactive systems. In order to integrate patterns, task, dialog, and object-oriented models to further automate user interface construction, the paper tightly couples these seemingly disparate development paradigms to allow a more powerful interplay. Thereby some of the missing links are identified for letting the pattern-based automated generation of complex parts of high-quality and media-rich applications become a routine job. A well-known smart phone app is examined to demonstrate some steps of the new approach.

Keywords: Model-based user interface development (MBUID), HCI patterns, task models, object-oriented models, dialog models, embedded patterns, user interface generation.

1 Introduction

Model-based user interface development (MBUID) has affected HCI since the nineteen eighties. However, over the years, the initial confidence in the efficiency and generative power of model-based development processes and tools has given way to a more unemotional view on the advantages and shortcomings of the model-based paradigm. The wide spectrum of end-users with differing levels of expertise and preferences, the heterogeneous working contexts and workflows, the variety of development environments and languages used for interactive system development, and the different target computing platforms from desktops to smart phones, with novel interaction techniques and modalities make it quite hard to define a generic and open-ended architecture for future MBUID environments [20].

M. Kurosu (Ed.): Human-Computer Interaction, Part I, HCII 2013, LNCS 8004, pp. 401–410, 2013.

Current trends in the area of interactive system design and user-centered development seem mostly to be aimed at providing a high degree of context-awareness, designer creativity and developer flexibility in order to produce high-quality interactive software with adequate user experience levels. The new social media with their intuitive, easy to use interfaces and the massive trend towards mobile applications and ubiquitous computers has directed the focus rather on rapid and agile user interface development methods than on detailed modeling environments.

On the other hand, research in the field of model-based and model-driven development is widespread and over the years has led to various approaches for automated GUI generation by interpreting semi-abstract and concrete models written in detailed user interface description languages [1], [29]. Model-based approaches also have proven to be valuable for approaching key challenges in HCI modeling and development, e.g. in the field of detailed modeling of real time control systems [19].

It is the goal of this paper to highlight some of the major contributions of MBUID and to pave the way for an interactive system development paradigm that leaves enough room for individual media-oriented design and development practices, but at the same time keeps in mind the benefits made available by MBUID, like high quality and usability through detailed modeling, platform independence, and efficiency through automated generation of non-trivial parts of the interactive target software.

HCI patterns could play a major role to more flexibility in MBUID. Therefore we propose a combined development approach that tightly couples the MBUID paradigm with HCI patterns for various interactive system modeling aspects.

The paper is structured in the following way: Chapter 2 will examine relevant work in the area of MBUID that is time-proven and in our opinion should make it into the combined approach. In chapter 3 we review the potential of pattern-based development approaches, introduce the notion of embedded patterns, and define the various pattern categories for pattern-based user interface modeling. Chapter 4 defines the structure and the major steps of the combined new development process and demonstrates some of its features by reengineering a popular media app. Chapter 5 concludes the paper and discusses the directions of our planned future work.

2 MBUID Accomplishments

Model-based user interface development environments introduce models to the development process of interactive applications [4]. The models are the carriers of the specified target system´s functional and data requirements and may serve to map these requirements to the structure and properties of the user interface in order to allow for effective and usable interaction of the user with the final target system implementation. The role of the models varies with respect to the modeling purpose. Typically more than one model is used interactively or parsed during the development process to construct the desired solution. Some ways in which models are exploited provide information for controlling highly-automated development processes for interactive systems. In the following we examine some major directions in model exploitation.

2.1 Task Models

Task models specify interaction goals, the sequence of user actions, and the collaboration of tasks and subtasks between users and the modeled system. They are in the center of many model-based development approaches. The CTT notation [23] is widely used in the HCI community for designing graphical tree specifications of the task structure and substructure of interactive applications. It depends on the level of detail, how complex task models may become for non-trivial applications.

As the static notation of task models to a certain degree describes the interactive behavior of the system under development, it is common to enable the execution of the task model together with a separately developed prototype or a mockup of the user interface in order to give an assessment of the quality of the modeled user-system interaction [2]. An executable task model can also be used as a starting point for deriving the user interface structure. In [25] so-called dialog graphs are derived directly from the task model. They allow for dynamic simulation of semi-abstract representations of the interactive application within various device contexts. The lifespan of task models can also encompass the runtime of the interactive application. Such task models can be used to implement context-aware dynamic user interfaces that behave differently at runtime, depending on the device type on which the application is executed [14].

In order to specify the detailed interactive behavior of highly sophisticated user interfaces and to design novel interaction techniques precisely, the approach presented in [22] combines task models, scenarios extracted from the task models, and detailed ICO models. ICO models are based on Petri nets [19] and allow the modeling of asynchronous and concurrent input/output actions as well as event-based state changes of interactive applications. Executable prototypes can be constructed from ICO models.

A recent approach for automatic generation of device-specific user interfaces from discourse models is presented in [24]. Discourse models are device and modality independent and define the sequence of questions and answers, i.e. the dialog, to solve a given problem. Together with a Domain-of-Discourse model that provides domain classes and their attributes, and the Action-Notification Model they form a communication model that represents the tasks and concepts of the interaction and serves as the basis for user interface generation.

2.2 Object-Oriented Models for User Interface Construction

Like task models, object-oriented models have a long history in MBUID environments. The survey in [4] compares several development environments of the first and second generation. In the last decade of the 20th century software engineering methods and tools were mainly based on object-oriented technologies. Some of the MBUID systems of this time-span, e.g. AME [15] and TADEUS [26] therefore chose the object-oriented modeling paradigm and notations for defining models of the interactive target system during its different lifecycle phases. Several systems were able to automatically define the window structure, choose abstract and concrete interaction objects for the user interface, and generate layout prototypes [28], [12], [15].

One aim of these MBUID environments was the tight coupling of the software engineering lifecycle with the user interface development process. Although most systems were targeted at the construction of traditional GUI-based applications, the technology and philosophy behind some of these environments today could again serve as one key technology for efficiently creating and generating web services and mobile apps.

The *Application Modeling Environment (AME)* [15], [16], e.g., applied a model refinement process and started modeling with an object-oriented analysis model that contained the domain classes (business logic) of the application, class attributes and attribute types, names and calling parameters of the class methods, and inter-class relations. The OOA model could also include the inter-class communication structure, i.e. the message-based calling structure of methods between domain classes that could later automatically be transformed into platform independent dialog sequencing code.

The OOA model was exploited by parsing the classes and inter-class relations (inheritance, aggregation, different types of associations) and by applying rules in order to define the hierarchical window structure of the user interface, abstract interaction objects, and the dynamic invocation structure for activating user interface elements and accessing the business logic. This information was collected in an OOD model. The OOD model included both, domain classes and their objects, and user interface classes and their objects (e.g. abstract windows, dialog boxes, and interaction objects). Typically one domain class was mapped to many user interface classes. Associations between the domain and user interface classes were used for communication and for transferring data between domain classes and the still abstract user interface.

The OOD model could be manually enhanced by browsers and editors. It served as the data model for an executable and fully functional prototype of the user interface. The UI and layout prototype was generated automatically by mapping the abstract interaction objects onto concrete interaction objects available within the simulation environment and by applying rules for interaction object grouping.

In order to arrive at a user interface prototype with satisfying usability characteristics, a user profile as well as a profile of the target environment could be interpreted by the generator. The resulting prototype could then be inspected with respect to correctness of the interaction design and usability and visual appearance of the user interface. From this, it was only a small remaining step to generate the C++ code for the user interface and the bindings to the business logic for different then state-of-the-art target platforms, e.g. Windows 95 and X-Window.

AME therefore was a quite flexible system that combined automation with explicit developer interaction during all phases of the development life cycle. One advantage of AME was the representation of the user interface as abstract OOD classes. These classes could serve as a stable user interface model and for communication with the business logic, without selecting concrete interaction objects too early. Instead, they were only defined during the final generation steps. The model was flexible enough to undo the selection decision and assign other concrete IOs with their inherent interaction techniques when the prototype was already being executed. This allowed the simulation and evaluation of different versions of the user interface before the target code was generated.

Model-driven architecture (MDA), a software engineering trend that evolved some years after MBUID [21], also is built on the transformation of abstract into more and more concrete model-representations with the final generation of domain and user interface code. As a starting point, typically, different UML models of the application are created and then exploited by automated model-transforming tools. MDA approaches provide UML models for all parts of the system under development. In contrast, in an MBUID environment like AME, as a starting point only such parts of the business logic that later would interact with the user interface had to be included into the OOA model [16]. Non-interactive parts of the application kernel could be developed independently by using the interactive business classes as their interface to the OOD user interface classes.

3 HCI Patterns

HCI patterns and pattern languages [18], [3], [27], [17], [5] recently have raised much interest in the MBUID community. Structural and behavioral information contained in HCI patterns can be used for modeling many different aspects of interactive systems or fine-tuning the properties of user interfaces derived from task or object-oriented models. Patterns also can be transformed into executable user interfaces by automated transformation processes [30], [9].

Within the PaMGIS framework [7] we have developed the *PaMGIS Pattern Specification Language (PPSL),* a powerful pattern description language based on PLML that can directly be exploited for user interface generation and for linking patterns with other modeling components. Table 1 shows the structure of PPSL pattern descriptions. There are many categories of HCI patterns [13], some of which are briefly discussed here. All pattern types can be specified with PPSL [6], [8], embedded into object-oriented models and linked to task and dialog models.

Structural Patterns. Such patterns resemble classic design patterns [11] and specify the class/object structure and the static and dynamic inter-class relations of parts of the interactive application.

Domain Patterns. Such patterns specify those parts of the business model that directly interact with the user interface.

GUI Patterns. Such patterns specify one or more abstract interaction objects, their relationships, and their interactive behavior. GUI patterns are primarily aimed at good usability.

Media Patterns. Such patterns specify (reusable groups of) media-specific interaction objects and their interaction techniques (e.g. multi touch input, speech input/output).

Infrastructural Patterns. Such patterns specify context and platform-specific reusable design information.

Table 1. Pattern description elements of the PaMGIS Framework

Specification Element	Brief Description
UPID	Unique pattern identifier as defined in [6]
Name	Name of the pattern
Alias	Alternative names, also known as
Illustration	Good example of instantiation of the pattern
Problem	Description of the problem to be solved
Context	Situations and circumstances in which the pattern can be applied
Forces	Description of forces in the environment that the use of the pattern will resolve
Solution	Description of how to resolve the problem
Synopsis	Summary of the pattern description
Diagram	Schematic visualization of the pattern
Evidence	Verification that it is in fact a pattern by
Example	at least three known uses of the pattern
Rationale	discussion and any principled reasons
Confidence	Rating of how likely the pattern provides an invariant solution for the given problem
Literature	References to related documents or papers
Implementation	Code or model fragments or details of technical realization as defined in [6]
Pattern-link	Relationship to other patterns or pattern instances as defined in [6]
Embedding-link	Back-link to object-oriented analysis and/or design model
ELinkID	Unique embedded link identifier
Reference class ID list	OO model classes from which the pattern is referenced
UML relationship-type list	Relation types between classes and referenced pattern
Label	Name of the embedding link
Management Information	Authorship and change management
Authors	Name of the pattern author
Credits	Merits
Creation-Date	Date of pattern compilation
Last-modified	Date of last change
Revision-number	Version of the pattern definition

UX Patterns. Such patterns specify rule-based or algorithmic knowledge for designing specific user experience characteristics for the interactive system.

Pattern Transformation Patterns (PTP). Such patterns contain rule-based or algorithmic descriptions for transforming patterns to different device contexts (e.g. from desktop to mobile device) [9].

4 Coupling HCI Patterns with Task- and Object Modeling

The discussion of important contributions to the fields of task modeling, object-oriented modeling, and patterns for interactive application development clarifies that it does not seem possible to arrive at an effective solution for user interface design automation, if one of the mentioned areas is neglected. Therefore we propose a tightly-coupled modeling paradigm which uses HCI patterns – specified in PPSL – as pivotal points to both integrate task modeling and object-oriented modeling aspects into the final design. This allows three different views onto the interactive system under construction: the pattern view, the task model view, and the object model view. The "missing links" can now be identified as the attributes `Implementation`, `Pattern-link`, and `Embedding-link` of PPSL specifications that relate each pattern with modeling components in the task and object models. Figure 1 shows the resulting combined model structure.

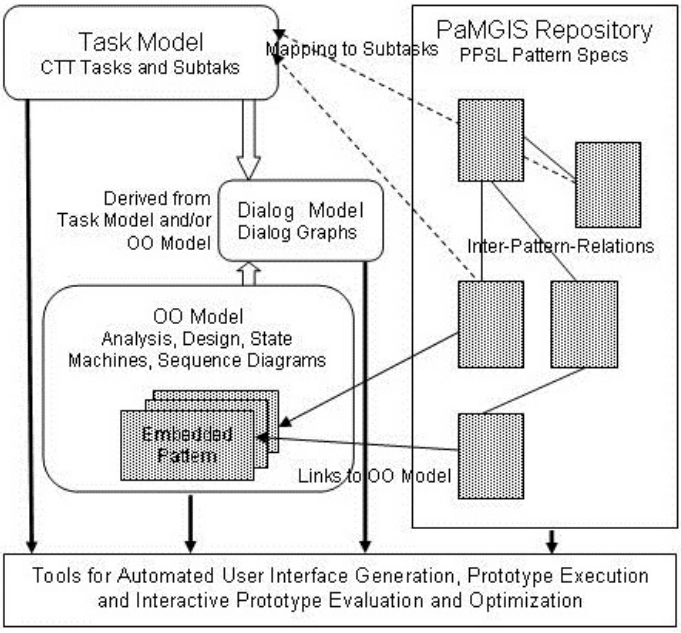

Fig. 1. A tightly-coupled modeling approach

As an example, we use the CNN smart phone application for Android to briefly demonstrate a small part of the proposed approach. Figure 2 shows the CTT task model for the *News Sharing pattern* found in many news apps for smart phones and tablets. In the user scenario the user wants to share some news from the CNN-App with his or her follower at google+. In the repository the pattern could be represented as a domain pattern with several domain classes each of which would be related to a user interface pattern. The domain pattern could be embedded into the OO model where it would be associated with one or more domain classes defining the actual aggregation and intra class structures.

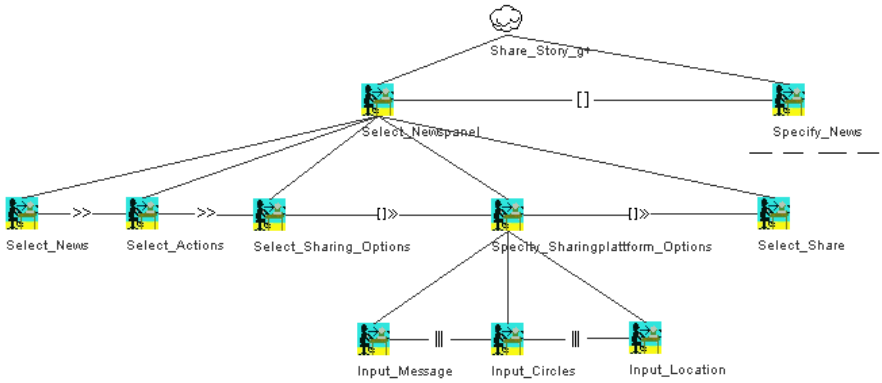

Fig. 2. Excerpt of the task model of the *News Sharing* pattern

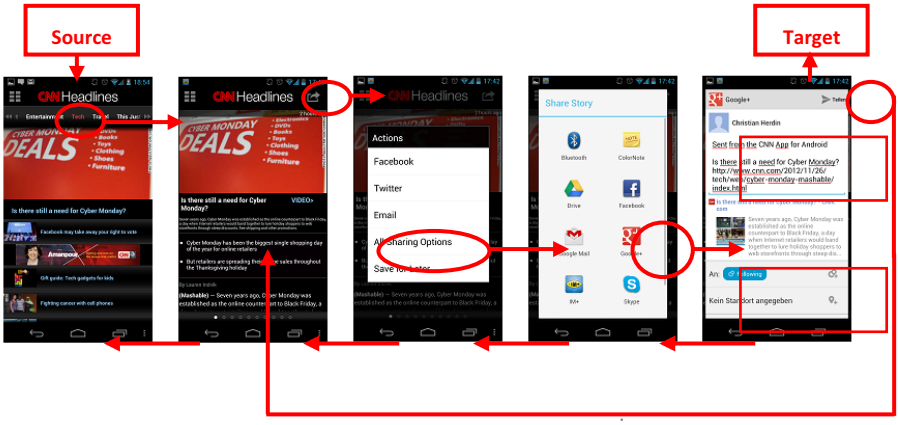

Fig. 3. Resulting UI of the Sharing dialog

The dialog structure for the pattern could be derived from the task model and/or the OO models in the form of dialog graphs or message-based activation sequences. The visual screen design that varies between the tablet and the smart phone app could be available as platform-independent and/or device-specific user interface patterns in the repository. These patterns could be directly referenced by the domain pattern embedded in the OO model during the refinement process executed by the interactive prototyping and automated generation tools. This could lead to the resulting user interface shown in figure 3.

5 Conclusion

This paper has reviewed proven approaches in task-, object-oriented, and pattern-based modeling of interactive systems. All three areas contribute heavily to the

quality and flexibility needed for real-world, state-of-the art interactive applications. By centering a combined modeling approach around powerful PPSL HCI pattern specifications and defining the most important inter-model-relationships, it becomes possible to integrate the strengths of the three modeling paradigms and to allow automated prototyping and generation tools an easier exploitation of the so far distributed modeling knowledge.

Thus, in future implementations of such combined modeling approaches, even detailed system modeling information, low-level interaction specifications, and creative user experience design suggestions could contribute to widely automatically generated prototypes of the interactive target applications.

References

1. Abrams, M., et al.: UIML: An Appliance-Independent XML User Interface Language. In: Proc. Eighth International World Wide Web Conference (WWW 1998). Elsevier Science Pub. (May 1999)
2. Bomsdorf, B., Szwillus, G.: Early Prototyping Based on Executable Task Models. In: Proc. CHI 1996, pp. 254–255. ACM, Vancouver (1996)
3. Breiner, K., et al. (eds.): Proc. of the 1st Int. Workshop on Pattern-Driven Engineering of Interactive Computing Systems (PEICS 2010), Berlin, Germany, June 20. ACM Int. Conf. Proc. Series (2010)
4. da Silva, P.P.: User Interface Declarative Models and Development Environments: A Survey. In: Paternó, F. (ed.) DSV-IS 2000. LNCS, vol. 1946, pp. 207–226. Springer, Heidelberg (2001)
5. Engel, J., Märtin, C., Forbrig, P.: Tool-support for Pattern-based Generation of User Interfaces. In: [3], pp. 24–27 (2010)
6. Engel, J., Märtin, C., Herdin, C., Forbrig, P.: Formal Pattern Specifications to Facilitate Semi-Automated User Interface Generation. In: Kurosu, M. (ed.) Human-Computer Interaction, Part I, HCII 2013. LNCS, vol. 8004, pp. 300–309. Springer, Heidelberg (2013)
7. Engel, J., Märtin, C.: PaMGIS: A Framework for Pattern-Based Modeling and Generation of Interactive Systems. In: Jacko, J.A. (ed.) Human-Computer Interaction, Part I, HCII 2009. LNCS, vol. 5610, pp. 826–835. Springer, Heidelberg (2009)
8. Engel, J., Herdin, C., Märtin, C.: Exploiting HCI Pattern Collections for User Interface Generation. In: Proc. Patterns 2012, Nice, France, pp. 36–44. IARIA (2012)
9. Engel, J., Märtin, C., Forbrig, P.: HCI Patterns as a Means to Transform Interactive User Interfaces to Diverse Contexts of Use. In: Jacko, J.A. (ed.) Human-Computer Interaction, Part I, HCII 2011. LNCS, vol. 6761, pp. 204–213. Springer, Heidelberg (2011)
10. Fincher, S., et al.: Perspectives on HCI Patterns: Concepts and Tools (Introducing PLML). In: CHI 2003 Workshop Report (2003)
11. Gamma, E., et al.: Design Patterns. Elements of Reusable Object-Oriented Software. Addison-Wesley, Reading (1995)
12. Janssen, C., Weisbecker, A., Ziegler, J.: Generating User Interfaces from Data Models and Dialog Net Specifications. In: Proc. INTERCHI 1993, pp. 418–423. IOS Press, ACM, Amsterdam (1993)
13. Kaelber, C., Märtin, C.: From Structural Analysis to Scenarios and Patterns for Knowledge Sharing Applications. In: Jacko, J.A. (ed.) Human-Computer Interaction, Part I, HCII 2011. LNCS, vol. 6761, pp. 258–267. Springer, Heidelberg (2011)

14. Klug, T., Kangasharju, J.: Executable Task Models. In: Proc. TAMODIA 2005, Gdansk, Poland, pp. 119–122. ACM (2005)
15. Märtin, C.: Software Life Cycle Automation for Interactive Applications: The AME Design Environment. In: Vanderdonckt, J. (ed.) Proc. of CADUI 1996, pp. 57–73. Presses Universitaires de Namur (1996)
16. Märtin, C.: Model-Based Software Engineering for Interactive Systems. In: Albrecht, R. (ed.) Systems: Theory and Practice, pp. 187–211. Springer, Wien (1998)
17. Märtin, C., Roski, A.: Structurally Supported Design of HCI Pattern Languages. In: Jacko, J.A. (ed.) Human-Computer Interaction, Part I, HCII 2007. LNCS, vol. 4550, pp. 1159–1167. Springer, Heidelberg (2007)
18. Marcus, A.: Patterns within Patterns. Interactions, 28–34 (March+April 2004)
19. Martinie, C., et al.: Formal Tasks and Systems Models as a Tool for Specifying and Assessing Automation Designs. In: Proc. ATACCS 2011, Barcelona, Spain, pp. 50–59 (2011)
20. Meixner, G., Paterno, F., Vanderdonckt, J.: Past, Present, and Future of Model-Based User Interface Development. i-com (3), 2–11 (2011)
21. Mellor, S.J., Scott, K., Uhl, A., Weise, D.: Model-Driven Architecture. In: Bruel, J.-M., Bellahsène, Z. (eds.) OOIS 2002. LNCS, vol. 2426, pp. 290–297. Springer, Heidelberg (2002)
22. Palanque, P., et al.: A Model-based Approach for Supporting Engineering Usability Evaluation of Interaction Techniques. In: Proc. EICS 2011, Pisa, Italy, pp. 21–29. ACM (2011)
23. Paternò, F.: Model-Based Design and Evaluation of Interactive Applications. Springer (2000)
24. Raneburger, D., et al.: Automated Generation of Device-Specific WIMP UIs: Weaving of Structural and Behavioral Models. In: EICS 2011, Pisa, Italy, pp. 41–46. ACM (2011)
25. Reichart, D., Forbrig, P., Dittmar, A.: Task Models as Basis for Requirements Engineering and Software Execution. In: Proc. TAMODIA 2004, Prague, Czech Republic, pp. 51–58. ACM (2004)
26. Schlungbaum, E., Elwert, T.: Automatic User Interface Generation from Declarative Models. In: Vanderdonckt, J. (ed.) Proc. of CADUI 1996, pp. 3–17. Presses Universitaires de Namur (1996)
27. Seissler, M., Breiner, K., Meixner, G.: Towards Pattern-Driven Engineering of Run-Time Adaptive User Interfaces for Smart Production Environments. In: Jacko, J.A. (ed.) Human-Computer Interaction, Part I, HCII 2011. LNCS, vol. 6761, pp. 299–308. Springer, Heidelberg (2011)
28. Vanderdonckt, J., Bodart, F.: Encapsulating Knowledge for Intelligent Automatic Interaction Objects Selection. In: Proc. INTERCHI 1993, pp. 424–429. IOS Press, ACM, Amsterdam (1993)
29. Vanderdonckt, J., et al.: UsiXML: a User Interface Description Language for Specifying multimodal User Interfaces. In: Proc. W3C Workshop on Multimodal Interaction (WMI 2004), July 19-20 (2004)
30. Wendler, S., et al.: Development of Graphical User Interfaces based on User Interface Patterns. In: Proc. Patterns 2012, Nice, France, pp. 57–66. IARIA (2012)