Model-with-Example: Model-Based Development of Polymorphic User Interfaces

Daniel Ziegler[1], Matthias Peissner[1], Doris Janssen[1]

Fraunhofer IAO, Institute for Industrial Engineering[1]

{daniel.ziegler, matthias.peissner, doris.janssen}@iao.fraunhofer.de

Abstract

Polymorphic user interfaces are getting more and more interesting as they offer different modes of display and interaction for different devices or situations and especially for diverse user needs and personal preferences. Model-based user interface design and development is a widely known approach to reduce development efforts while maintaining the flexibility to produce diverse UIs. However, it is not much used in practice because existing approaches often do not offer an attractive balance of abstraction as required by generators and a concrete and vivid representation for developers. In this paper, we present the Model-with-Example (MwX) approach. It allows developers to work on a wireflow-like visualisation while they edit an abstract model of the UI and supports the communication within the project team by rendering specific storyboards from the abstract model. A first user study shows that the MwX approach can positively affect development efficiency and overall acceptance of model-based development.

1 Introduction

The concept of polymorphic user interfaces refers to the design of multiple alternative user interface (UI) instances to address diverse users and usage contexts (Savidis and Stephanidis, 2004). The complexity emerging from this increasing variety in UI design solutions results in the need for new development approaches. Over years, Model-Based User Interface Design and Development (MBUID) has become a common approach to reduce efforts in the creation of polymorphic UIs. It addresses the rising need for variety in UI design that is caused by an increasingly heterogeneous context of use, including users' personal needs and preferences, device capabilities and implementation platforms, as well as environmental conditions under which the system is used (Meixner et al., 2011). Researchers have developed a number of MBUID systems that focus on different aspects of the context of use. However, until today, there is no significant market uptake in the mainstream software industry.

Veröffentlicht durch die Gesellschaft für Informatik e. V. 2018 in
R. Dachselt, G. Weber (Hrsg.):
Mensch und Computer 2018 – Tagungsband, 02.–05. September 2018, Dresden.
 https://doi.org/10.18420/muc2018-mci-0229

One important obstacle might be the modelling tools that serve as the interface between developers and the respective MBUID system. Graphical modelling languages do not seem to be enough to lower the threshold for UI creators (Vanderdonckt, 2008) and shorten the learning curve (Peissner, Schuller, et al., 2014). Resulting UIs automatically generated from the created models are often unpredictable for them (Myers et al., 2000) because of their complexity. To ensure efficient development processes, the models must be concise and clear. Moreover, the mechanisms to generate the resulting UIs must be easy to comprehend and control (Peissner, Schuller, et al., 2014).

This paper makes the following contributions to the field of modelling tools for polymorphic UIs: (1) It introduces the Model-with-Example approach and explains how it emerges from an analysis of existing MBUID systems for polymorphic UIs. (2) It presents the concept of a modelling environment for the new approach. (3) It provides the results of a concept evaluation with potential users and discusses future improvements and planned extensions.

2 Related Work

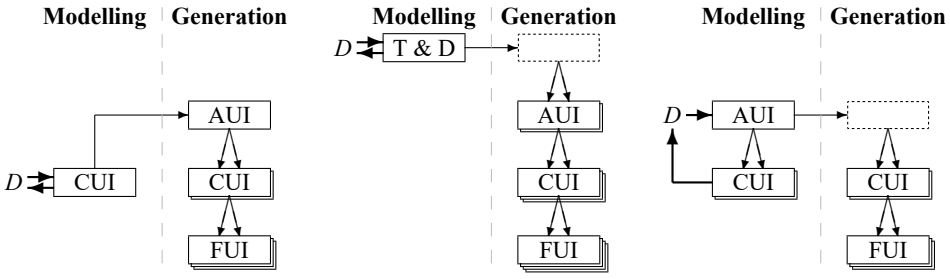
The *CAMELEON* reference framework (Calvary et al., 2003) has become the de facto standard for structuring MBUID processes and defines four levels of abstraction. The Final User Interface (FUI) is bound to a specific implementation platform while the Concrete User Interface (CUI) abstracts from the implementation. The CUI still refers to certain ways of interaction whereas the Abstract User Interface (AUI) is independent of any interaction modality. Finally, the task & domain level describes processes and concepts relevant for the application. Akiki et al. (2014) "consider tool support to be crucial for the adoption of adaptive model-driven UI development by the software industry". While models enable more flexibility in adaptations by defining the UI on more abstract levels they require higher developer skills and mental effort compared to widget toolkits. Therefore, appropriate development tools will be needed to make MBUID easier and more efficient.

Based on their predominant modelling approach, existing systems can be grouped in two categories: GUI-Builder-like and task-based approaches. These categories have also been detected by Nguyen et al. (2016) when reviewing the support for generative UI patterns.

2.1 GUI-Builder-like approaches

One major area of MBUID is the development of UIs that can be used across different technical platforms. Such multi-platform development tools target at multiple UI implementations (i.e. FUIs) that are designed as similar as possible. They mostly follow a GUI-Builder-like approach starting from CUI models. The graphical editors for these models typically work quite similar to modern Graphical User Interface (GUI) builders available in many Integrated Development Environments (IDEs) by positioning UI elements like labels, buttons and text fields.

For example, the *Damask* system presented by Lin and Landay (2008) allows designers to prototype cross-device UIs with sketches. It supports developers with its collection of cross-device UI patterns, but does not ensure consistency between their device-specific instances.



(a) GUI-Builder-like *(b)* Task-based *(c)* Model-with-Example

Figure 1. **Structural flow of models through the CAMELEON levels of abstraction during modelling by developers and automatic UI generation by the system in different approaches. Bold arrows on the left of each figure represent the interaction with developers (*D*) while modelling. Dashed boxes stand for models that are copied without modification.**

Since each device is represented by a new layer, the complexity of creating prototypes increases rapidly with a growing number of UI variants. In contrast, Meskens et al. (2008) explicitly present *Gummy* as a multi-platform GUI builder. Its rule-based transformation engine derives an initial CUI from the existing platform-specific CUI for each new target platform. Nebeling et al. (2014) propose *XDStudio* as an interactive development system for UIs that are distributed across devices. Authors create UI variants from an existing web-based UI implementation, either in simulated target environments or directly on target devices. In both modes the author always gets a very direct impression of the resulting UIs.

This also is the major advantage of modelling at the concrete level in the GUI-Builder-like approach: Developers are always aware of how the UI finally will be presented to users at runtime. Many commercial GUI prototyping tools work quite similar. In addition, most systems perform the generation of UI variants at design-time allowing developers to control the generation process itself and to check the generated results, but also leading to rapidly growing efforts in the case of an increasing number of target contexts. UIs variants for different contexts of use often vary in aspects that result from transformations on higher levels than from CUI to FUI. Therefore, in GUI-Builder-like approaches an interpretation of the CUI into higher levels of abstraction is required. This interpretation may either be an implicit part of adaptation rules between CUIs or explicitly performed as a transformation from a CUI model to an intermediate AUI model (see Figure 1a). Because adaptation rules can be very complex, developers may easily create constellations in the CUI without being aware of their meanings. Especially when combined with a dynamic generation of UI variants at runtime, this problem results in the risk of incorrect adaptations.

2.2 Task-based approaches

In contrast to multi-platform UIs, personalised and context-sensitive interactive systems provide each user specific UI to fit the individual needs (Edmonds, 1981) and current contextual

circumstances (Oppermann, 2005). To maximise possible adaptations, these systems incorporate models on all CAMELEON abstractions including the task & domain level.

Quill, for example, is an IDE aiming to efficiently support diverse development processes potentially starting from models at any level of abstraction (Motti et al., 2013). It requires all model transformations to be applicable in both directions. These transformations are still to be supervised and controlled by the developer. The issue of deriving abstract models from more concrete ones is also mentioned by García Frey et al. (2012) in the description of *UsiComp*. While it is generally capable of performing transformations in both directions, similar to Quill only forward engineering to more concrete models has already been tested.

The difficulties concerning reverse transformations in practice require a top-down and thus task-based modelling approach. To generate FUIs, three transformations starting from the task model are applied that follow all levels of abstraction (see Figure 1b). Each of these transformation steps may depend on aspects of the context of use and may therefore generate multiple models at its target level. Due to transformations depending on the context of use and the huge number of potential variants, the generation process typically has to be performed at runtime. It is not possible for developers to individually check the generated FUIs before deployment.

3 Model-with-Example

In all previously described modelling approaches a model editor presents a visualization (concrete syntax) on the same level of abstraction as the model that is created (abstract syntax). In contrast, with Model-with-Example (MwX) developers create an AUI model while the editor provides them a visualization on the more concrete CUI level. However, the AUI is used for the generation of FUIs resulting in a structural flow of models as depicted in Figure 1c. This approach addresses three basic requirements for market acceptance to improve the acceptance of MBUID for polymorphic UIs: Efficiency of the development process, short learning curve for developers, and transparency of UI generation (Peissner, Schuller, et al., 2014).

Efficiency of the development process: One major aspect of an efficient model-based development process is the effort for creating the necessary models. The need for developers to work with or even manually create comprehensive models on multiple abstraction levels results in high modelling efforts. Therefore, MwX is based on the creation of one single concise AUI model. All transformations are performed during the runtime generation process. Developers are not required to manipulate or even look at any generated intermediate model. New UI variants can easily be added to the runtime system without the need for developers to change any model or perform any transformation in the development environment.

Short learning curve: While abstraction is necessary to enable a wide variety of UIs, in today's development practice "designers are accustomed to starting with concrete interfaces" (Lin and Landay, 2008) and "prefer to have a concrete representation during their design activities" (Meskens et al., 2008). The creation of more abstract models (i.e. task models) results in a big conceptual distance between those models and the developers' mental model of a UI. The mental effort for model creation rises and initially learning to create such models gets harder.

Similar to GUI-Builder-like approaches, in MwX developers work with a CUI representation. This means there is a small distance between the level of abstraction the developer is working on and the FUI presented to users at runtime. In consequence, we expect MwX to reduce the effort of learning to create AUI models.

Transparency of UI generation: A development environment for polymorphic UIs provides transparency of the generation process if developers are able to figure out a clear connection between the specified model and the final generated output (see Myers et al., 2000). However, reverse transformations in middle-out processes as mentioned for Quill and implicit interpretations in GUI-Builder-like approaches make it hard for them to gain a proper understanding of the underlying mechanisms due to potentially ambiguous results. Voice menus, for example, might use the same mechanisms for navigation options and possible transactions, for which most designers would use different controls on a graphical UI. MwX uses only forward transformations from more abstract CAMELEON levels to more concrete models. Thus, it supports the transparency of the generation process by preventing ambiguous transformation steps. Additionally, MwX increases transparency by using a concrete visualization directly in the model editor, providing developers an immediate overview of possible UIs resulting from the model they currently create.

3.1 Concrete representation as wireframes

The MwX model editor is based on the concepts of the Abstract Application Interaction Model (AAIM), an User Interface Description Language (UIDL) on the AUI level that is based on Unified Modeling Language (UML 2) state machine diagrams. It has been explicitly designed to serve as the only concise artefact to be created by application developers (Peissner, Häbe, et al., 2012). The concept of interaction situations is used to represent the purpose of each state in the modelled interaction flow. For example, an interaction situation *SelectOneOfMany* in the AAIM could result in a drop-down list widget or a grid of buttons at runtime.

In the MwX model editor, an AAIM is not displayed in the original boxes-and-arrows notation, but as wireframes that represent one of its specific CUIs. This combination results in a representation similar to wireflows (Laubheimer, 2016), which provide an impression of an application's interaction structure as well as of its page layout. The editor selects the displayed CUI variant based on user, platform and environment profiles. These profiles may be predefined similar to XDStudio's distribution profiles (Nebeling et al., 2014) or individually customised on-the-fly like in the What-You-See-Is-What-Others-Get (WYSIWOG) preview of MyUI (Peissner, Häbe, et al., 2012).

The model editor contains two views: an overview showing a complete AAIM and a detail view of a single state. In the model overview the level of detail depends on the zoom level of the editor. When zooming out, interaction situations are represented by their name for a more compact textual visualisation (see Figure 2). Zooming into more detail results in the visual representation of the interaction situations by their wireframes. The same wireframe representation is used in the detail view that can be opened with a double click on a specific state. In both views, interaction situations can be edited by drag-and-drop them from the palette on the right or by entering their name directly.

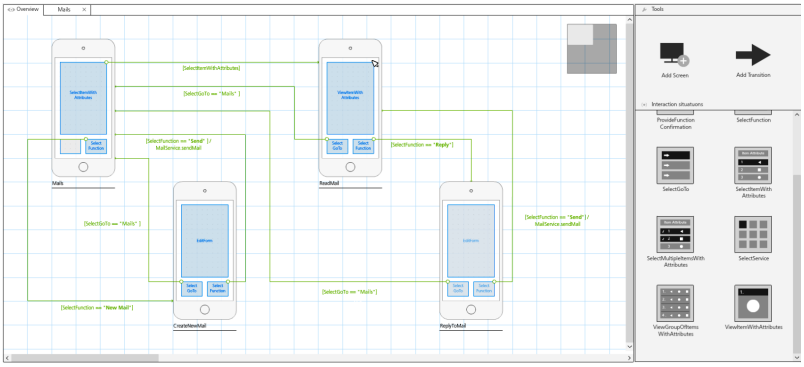
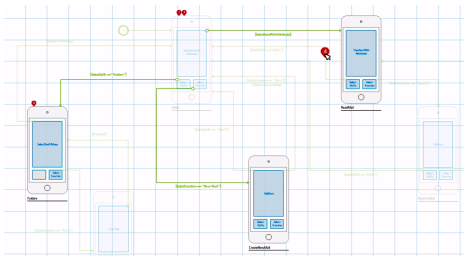
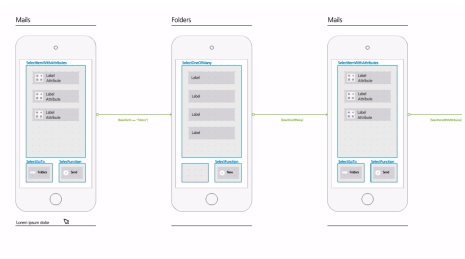


Figure 2. **General concept of the AAIM editor following the Model-with-Example approach. Interaction states are represented by wireframes according to the selected context of use profiles. Navigation paths are shown as green transition arrows.**



(a) Step-by-step definition of a story in the AAIM editor. Only states directly reachable from the current one are highlighted and selectable.

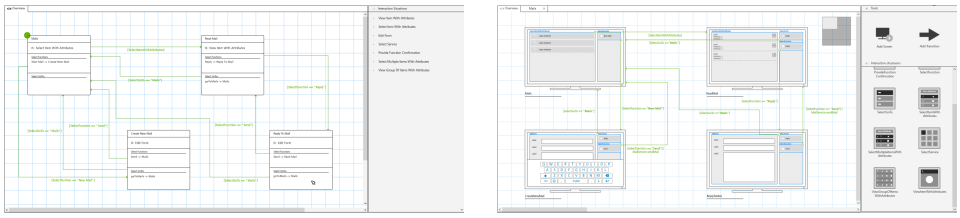


(b) Example of a generated storyboard as a linear representation of a defined story. Each step is shown as the corresponding wireframe visualisation of the AAIM detail view.

Figure 3. **Concept of the storyboard feature of the MwX modelling environment.**

3.2 Generation of specific storyboards

UI storyboards have proven to be a valuable tool in the creation of interactive systems over decades (Vertelney, 1989). They clearly visualise the interaction sequence of only one defined usage story and thus reduce complexity for communication with stakeholders. The availability of concrete wireframe representations enables the MwX environment to generate specific storyboards for each set of user characteristics, platform specifications and environmental parameters. The modelling environment allows developers to interactively define a set of stories based on an existing AAIM as shown in Figure 3a. By clicking on a specific state with the story tool, it is selected as the starting point of a new story. Subsequent steps of the story can be defined accordingly. The editor supports developers by highlighting states that are directly connected to the state selected before by a transition. Only these states may be used as next step of the sequence to ensure consistency.



(a) Prototype A, created according to the original MyUI AAIM editor in order to match the visual appearance of the MwX prototype.

(b) Prototype B: Model-with-Example AAIM editor showing wireframes for the platform profile of a smart TV.

Figure 4. **Screenshots of the prototypes used for comparison in the evaluation study.**

Defined stories can be opened in the storyboard view. This view presents a side-by-side representation of the respective interaction sequence as depicted in Figure 3b. States are displayed as duplicated wireframes if they occur multiple times in the definition of a story.

4 Evaluation with Developers

We carried out a user study with UI developers to evaluate if the MwX approach is suitable for the model-driven development of polymorphic UIs, addressing these research questions:

RQ1 Does the MwX approach *reduce the effort* of creating an AUI model?
RQ2 Is the MwX approach able to *gain additional benefits* in the creation of an AUI model?
RQ3 Does the MwX approach *improve the overall acceptance* of the AUI modelling?

4.1 Procedure

The study was conducted with eight industrial developers from three companies located in different European countries. Their overall occupation with the design and development of user interfaces ranges from 10% to 80% ($M = 41.25$, $SD = 24.16$) of their work. The self-assessment of their overall knowledge of the Unified Modeling Language covers the complete 5-point range ($M = 2.63$, $SD = 1.41$). In total, three group sessions of 90-120 minutes each have been carried out with the following structure: After the welcome and consent forms the moderator introduced the participants to the topic of polymorphic UIs and the basics of the MyUI approach to MBUID. After this introduction the participants completed a questionnaire to assess how relevant the topic is for them. Next, two animated prototypes of AAIM model editors were presented. Prototype A had been designed according to the original AAIM editor of MyUI (see Figure 4a). Prototype B followed the concept of a MwX model editor as described above (see Figure 4b). Both prototypes demonstrated the creation of the same AAIM for a simple email application. The order of the presentation of both prototypes has been randomized for each session. An additional prototype had been created to demonstrate the definition of a

	A (MyUI)		**B** (MwX)		**Diff.** ($B - A$)		**Storyboard**	
Scale	Mean	SD	Mean	SD	Mean	SD	Mean	SD
ABS	3.22	0.86	4.16	0.33	+0.94	0.95		
COM							4.67	0.40
EFF	3.66	0.80	4.53	0.39	+0.88	0.61	4.88	0.19
BEN	3.54	0.96	4.33	0.67	+0.79	0.87	4.29	0.60
ACC	3.25	1.28	4.38	0.74	+1.13	1.46	4.63	0.74

Note. M = mean, SD = standard deviation.

Table 1. **Descriptive statistics for the comparison of both AAIM editors including the difference of the scales calculated per participant and for the storyboard feature.**

story and generation of the respective storyboard as depicted in Figure 3. The session was closed after discussion about open issues of the participants.

After the presentation of each prototype the participants rate that prototype using a questionnaire that had been specifically designed for this study. All contained items were based on 5-point Likert scales labelled from 1 ‘Do not agree at all’ to 5 ‘Completely agree’. To address the research questions defined above, five scales have been used in the questionnaires: Appropriate level of abstraction (ABS) contained four items related to RQ1 for the model editor comparison. Efficiency of use (EFF) was related to RQ1 in general with four items. Support of communication (COM) addressed RQ2 with regards to the storyboard feature in three items. Expected benefits (BEN) covered general aspects of RQ2 with three items. Overall acceptance (ACC) directly related to RQ3 with a single item.

4.2 Results

The overall ratings for the editor comparison calculated as the mean of all participants’ values are shown in Table 1. Although the ratings for the MyUI prototype are on a medium level, the value of each of the scales is higher for the MwX prototype. When comparing standard deviations for both prototypes, values of all scales are noticeable smaller for the MwX prototype.

Calculating the difference as the mean of the difference per user results in remarkably high standard deviations. Looking into details, only one participant rated the level of abstraction of the MwX prototype slightly lower than the original AAIM editor ($ABS_B - ABS_A = -0.25$), while the other participants rated it to be more appropriate. For the efficiency of use, despite one participant with an equal rating for both prototypes there are only positive differences. Similar to the first scale, the results for the expected benefits show one participant with a negative difference ($BEN_B - BEN_A = -0.33$). In addition one participant gave the same ratings for both prototypes. In the case of the overall acceptance, one participant rated the MwX prototype one point lower than prototype A and two participants rated both prototypes to be equal.

The quantitative results for the storyboard feature are also shown in Table 1. Mean ratings of all four scales are very positive with values considerably over the neutral value ‘3’. The efficiency

of use even nearly reaches the maximum, meaning that almost all participants completely agree. Particular attention should be paid to the COM scale as it relates to the central intention of this feature. All participants rated the support of communication with a value of '4' or higher with four ratings at the maximum value of '5'. Standard deviation values are quite low for all contained scales, comparable to those of the MwX model editor prototype described above.

5 Conclusion and Outlook

Since our work aims to improve the acceptance of MBUID for polymorphic UIs only developers that are currently not used to MBUID approaches participated in the presented study. Results might differ for developers already familiar with existing modelling approaches. The participants have not used the modelling tool on their own, so results regarding the efficiency of use should be interpreted as an anticipation rather than a perception.

However, the results of the user study show that the MwX approach has potential to improve the anticipated efficiency of MBUID for polymorphic UIs. It can reduce development efforts by providing a modelling tool that works on appropriate abstraction levels according to developers' mental models. At the same time, the proposed concept decreases the direct efforts of using the model editor. Supporting project communication seems to be an interesting benefit of MwX as demonstrated by means of the automatic rendering of storyboards. However, according to their qualitative comments, participants were concerned about AAIMs getting too complex for real-world applications. Strong support for modularization of AUI models might help to reduce cluttered visualisation in case of large models. This would also support mainstream development practices for teams working concurrently at different parts of the same model.

Working with specific usage stories has proven to be an important opportunity to investigate further. In addition, a notation for stories that is independent of an existing AAIM would allow to define use cases and stories upfront in analysis and design. Predefined stories can serve as a specification for the AUI to be modelled, comparable to test specifications in test-driven development (Beck, 2002). The modelling environment would be able to automatically check consistency between the defined stories and the modelled AUI. This would enable a flexible usage of the MwX modelling environment according to individual development processes.

Acknowledgements

The research leading to these results has received funding from the European Union's Seventh Framework Program (FP7) under grant agreement No. 610510 (Prosperity4All). The opinions herein are those of the authors and not necessarily those of the funding agency.

References

Akiki, P. A., Bandara, A. K., & Yu, Y. (2014). Adaptive model-driven user interface development systems. *ACM Computing Surveys*, *47*(1), 1–33.

Beck, K. (2002). *Test driven development: By example*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., & Vanderdonckt, J. (2003). A unifying reference framework for multi-target user interfaces. *Interacting with Computers*, *15*(3), 289–308.

Edmonds, E. A. (1981). Adaptive man-computer interfaces. *Computing skills and the user interface*, *122*, 389–426.

García Frey, A., Céret, E., Dupuy-Chessa, S., Calvary, G., & Gabillon, Y. (2012). Usicomp: An extensible model-driven composer. In *Proc. EICS '12* (pp. 263–268). New York, NY, USA: ACM.

Laubheimer, P. (2016). Wireflows: A ux deliverable for workflows and apps. Retrieved September 20, 2017, from https://www.nngroup.com/articles/wireflows/

Lin, J. & Landay, J. A. (2008). Employing patterns and layers for early-stage design and prototyping of cross-device user interfaces. In *Proc. CHI '08* (pp. 1313–1322). New York, NY, USA: ACM.

Meixner, G., Paternò, F., & Vanderdonckt, J. (2011). Past, present, and future of model-based user interface development. *i-com*, *10*(3), 2–11.

Meskens, J., Vermeulen, J., Luyten, K., & Coninx, K. (2008). Gummy for multi-platform user interface designs: Shape me, multiply me, fix me, use me. In *Proc. AVI '08* (pp. 233–240). New York, NY, USA: ACM.

Motti, V. G., Raggett, D., Cauwelaert, S. V., & Vanderdonckt, J. (2013). Simplifying the development of cross-platform web user interfaces by collaborative model-based design. In *Proc. SIGDOC '13* (pp. 55–64). New York, NY, USA: ACM.

Myers, B., Hudson, S. E., & Pausch, R. (2000). Past, present, and future of user interface software tools. *ACM Transactions on Computer-Human Interaction*, *7*(1), 3–28.

Nebeling, M., Mintsi, T., Husmann, M., & Norrie, M. (2014). Interactive development of cross-device user interfaces. In *Proc. CHI '14* (pp. 2793–2802). New York, NY, USA: ACM.

Nguyen, T.-D., Vanderdonckt, J., & Seffah, A. (2016). Generative patterns for designing multiple user interfaces. In *Proc. MOBILESoft '16* (pp. 151–159). New York, NY, USA: ACM.

Oppermann, R. (2005). From user-adaptive to context-adaptive information systems. *i-com*, *4*(3/2005), 4–14.

Peissner, M., Häbe, D., Janssen, D., & Sellner, T. (2012). MyUI: Generating accessible user interfaces from multimodal design patterns. In *Proc. EICS '12* (pp. 81–90). New York, NY, USA: ACM.

Peissner, M., Schuller, A., Ziegler, D., Knecht, C., & Zimmermann, G. (2014). Requirements for the successful market adoption of adaptive user interfaces for accessibility. In *Proc. HCII 2014* (Vol. 8516, pp. 431–442). LNCS. Springer International Publishing.

Savidis, A. & Stephanidis, C. (2004). Unified user interface design: Designing universally accessible interactions. *Interacting with Computers*, *16*(2), 243–270.

Vanderdonckt, J. (2008). Model-driven engineering of user interfaces: Promises, successes, failures, and challenges. In *Proc. ROCHI '08* (pp. 1–10). Bucharest, Romania: Matrix ROM.

Vertelney, L. (1989). Using video to prototype user interfaces. *SIGCHI Bulletin*, *21*(2), 57–61.