Enabling Accessibility through Model-based User Interface Development

Daniel ZIEGLER[1] and Matthias PEISSNER
Fraunhofer Institute for Industrial Engineering IAO, Stuttgart, Germany

Abstract. Adaptive user interfaces (AUIs) can increase the accessibility of interactive systems. They provide personalized display and interaction modes to fit individual user needs. Most AUI approaches rely on model-based development, which is considered relatively demanding. This paper explores strategies to make model-based development more attractive for mainstream developers.

Keywords. Adaptive User Interfaces, Personalization, Model-based UI Development, Model Editor

1. Introduction and Background

Users with disabilities and other users with special needs require individualized solutions to access technology. In many cases, this can include very specific input and output devices. Solutions like AsTeRICS [1] help to build individual assistive technologies for interacting with computers based on a set of configurable hardware and software modules. Such approaches can enable universal interaction with a wide range of existing software user interfaces (UIs).

Personalization on the software side of the UI, however, can be much more difficult. In most cases, it will not be possible to create diverse UIs for existing applications or services to satisfy individual user needs. Quite often, the technical interfaces for custom implementations are not available. Additionally, continuous maintenance efforts may be required due to updates of the original application or service. This issue can be addressed by polymorphic UIs, which include multiple variants of the UI [2]. For example, SUPPLE++ automatically finds the best matching UI for users with motor and vision impairments based on a rating algorithm [3]. Following a different approach, MyUI generates individual UIs based on an extensible set of UI patterns created by accessibility experts [4].

In both cases, developers do not implement a UI by composing UI elements such as buttons and input fields like in traditional UI development. Instead, the developers work on an abstract model of the possible interactions between the system and a user. This model may be on different levels of abstraction according to the CAMELEON reference framework [5]. The final UI presented to users is then automatically generated from the model based on a set of predefined transformations.

In traditional UI development practice, the UI finally presented to users corresponds directly to the source code written by the developers. Today, many

[1] Corresponding Author: Fraunhofer Institute for Industrial Engineering IAO, Nobelstraße 12, 70569 Stuttgart, Germany, E-mail: daniel.ziegler@iao.fraunhofer.de

integrated development environments (IDEs) provide visual editors for UI creation with a preview of the final results. When creating abstract models instead of UI source code, developers may have difficulties to predict how the final UI will look [6]. Therefore, the transparency of the generation process and adaptation mechanisms (i.e. they are comprehensible for developers) is considered one of the requirements for market uptake of AUI systems [7].

The super-ordinate goal of this work is to mainstream model-based user interface development (MBUID) by making it more attractive for developers. This will help to automatically integrate accessibility features in industrial standard software. This paper reports the results of a qualitative study with front-end developers to find out more about existing reservations against MBUID and about industrial requirements to make MBUID an attractive approach for standard software development projects.

2. Methods

2.1. Objectives

The goal of the study conducted with front-end developers was to explore the following questions:

- Which areas of application are expected to benefit from the model-based development of polymorphic UIs?
- Which benefits of MBUID should be emphasized to motivate mainstream market uptake? Which challenges arising from this approach should be addressed?
- Is predictability of the resulting UIs one of the key challenges when using abstract models for UI development and may a concrete visualization help to mitigate it?
- Is there a potential to gain additional benefits from the visual representation of the final UI, for example by generating specific UI storyboards?

2.2. Procedure

Eight industrial developers from three countries across Europe (Germany, Poland, and Denmark) volunteered to participate in the study. In their daily work, building UIs has a share of 10% to 80% (Mean M = 41.25, Standard Deviation SD = 24.16) with only one of them having a dedicated User Experience design role. On a five point scale the rating of their own knowledge regarding the Unified Modeling Language covered the complete range (M = 2.63, SD = 1.41).

The group sessions started with an introduction to the topics of polymorphic UIs and MBUID. Then they were asked to individually fill out a questionnaire on the possible areas of application together with possible benefits and challenges of this approach.

Secondly, two animated prototypes of model editors were shown to the groups in random order. The first prototype depicted in Figure 1 was based on an abstract visualization of the abstract UI model using boxes and arrows similar to the notation of UML2 state machine diagrams [8]. The second prototype incorporated the visualization of specific wireframes following a Model-with-Example approach where developers create the abstract model while seeing one concrete example of the final UI (see Figure 2). This approach addresses the predictability issue by presenting specific wireframes instead of textual definitions. The wireframes are generated according to user and device profiles selected by the developer out of previously defined lists (e.g. implemented as drop-down list box). Figure 3 compares the visualization of interaction states in both prototypes. After the prototypes were presented, participants were asked about potential benefits and challenges of the outlined development procedures and tooling.

Finally, a third prototype introduced an additional feature. Based on an existing UI model, this feature allows developers to define stories that represent a specific course of interaction through the application. The development environment generates storyboards for each of these stories to show the final UI for a specific user and a specific device set up (see Figure 4).

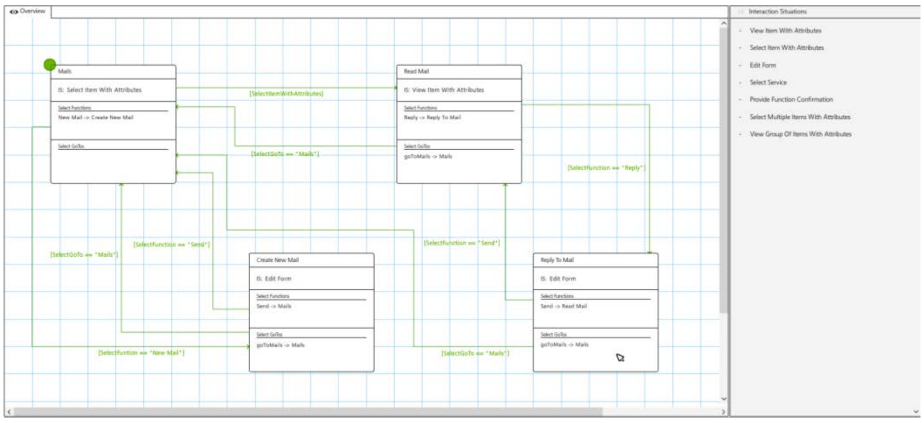

Figure 1. Model editor using an abstract visualization similar to UML state machine diagrams. Boxes represent the states of the interaction, while arrows stand for available navigation paths. The interaction possibilities of each state are defined by textual expressions.

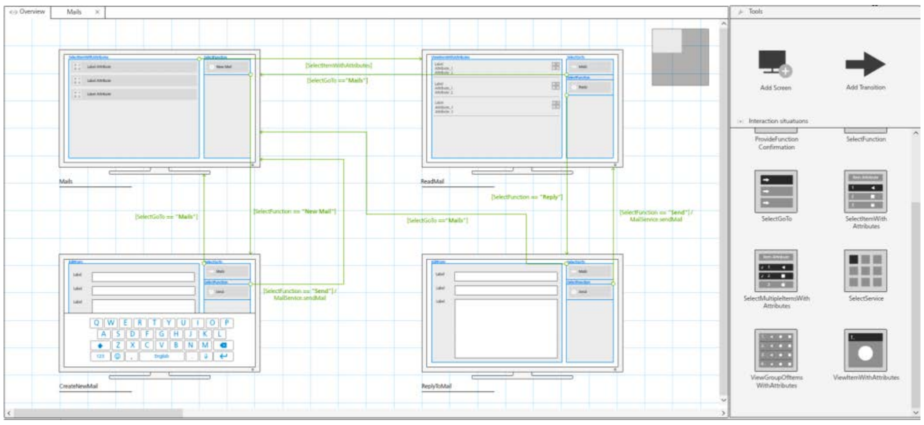
Overview
Mails
Mails
ReadMail
CreateNewMail
ReplyToMail
[SelectItemWithAttributes]
[SelectGoTo =="Mails"]
[SelectFunction == "Reply"]
[SelectGoTo == "Mails"]
[SelectFunction == "New Mail"]
[SelectGoTo =="Mails"]
[SelectFunction == "Send"] / MailService.sendMail
[SelectFunction == "Send"] / MailService.sendMail
Tools
Add Screen
Add Transition
Interaction situations
ProvideFunction Confirmation
SelectFunction
SelectGoTo
SelectItemWith Attributes
SelectMultipleItemsWith Attributes
SelectService
ViewGroupOfItems WithAttributes
ViewItemWithAttributes

Figure 2. Model editor using the Model-with-Example visualization for the case of smart TV as the selected device profile. Interaction states are represented by wireframes of the defined interaction possibilities according to the selected profiles. Navigation arrows start from the part of the wireframe that triggers the transition.

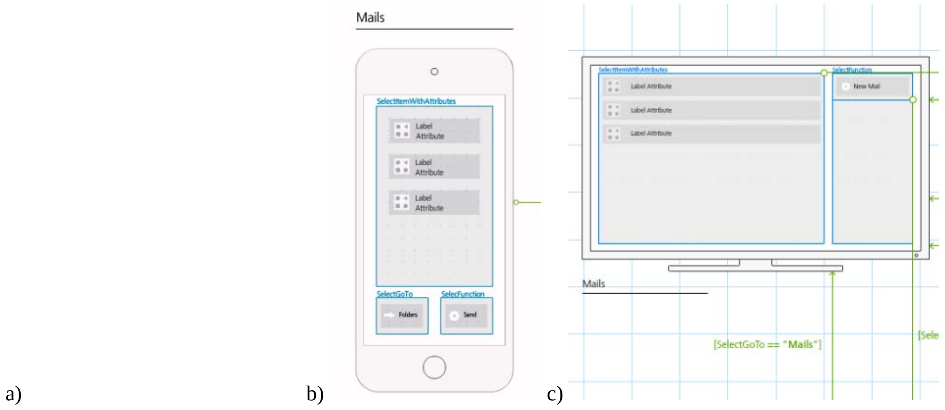

Figure 3. Comparison of different interaction state visualizations: a) abstract visualization in the first prototype; b) concrete visualization as wireframe for a smartphone in the Model-with-Example prototype; c) concrete visualization as wireframe for a smart TV in the Model-with-Example prototype.

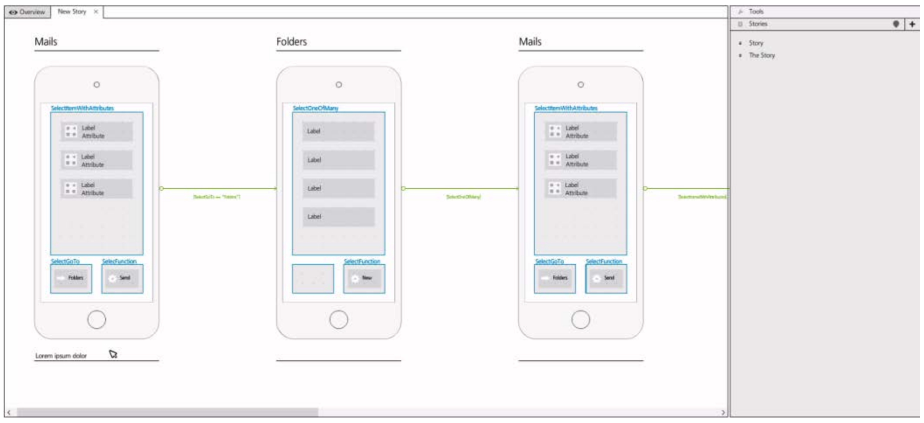

Figure 4. Visualization of a specific interaction sequence as a storyboard with wireframes according to a smartphone profile. Interaction states that occur more than once in the sequence are displayed multiple times accordingly (e.g. the “Mails” state at position one and three in this example).

3. Results

3.1. Application Areas for Polymorphic User Interfaces

The participants employed different dimensions and criteria to describe the applications of polymorphic UIs they envisaged. They mentioned mobile apps and web applications as the main deployment platforms. Both platforms address a great diversity of users which makes it difficult to gain detailed knowledge about them during development. An essential characteristic of mobile apps seems to be that situations and conditions often change over time and even during their usage, e.g. when users leave a building into bright sunlight. For web applications, the participants mainly thought of aspects related to persons which the UI should adapt to. The mentioned goal of addressing accessibility issues might be motivated by the fact that the public accessibility awareness focuses mainly on the web. Moreover, the participants thought that polymorphic UIs could help to provide individual user experiences especially in web applications.

The purpose of an application emerged as a second interesting dimension. Life-assisting applications could incorporate polymorphic UIs to adapt to users' requirements, which are changing with their age or support different goals of usage. In professional environments, management applications might present different levels of detail depending on different situations. For applications used on industrial shop floors, participants mentioned the support of workers with individual accessibility needs as well as impediments by required protective equipment and variable working positions.

Regarding applications in general, participants on the one hand referred to devices' screen size as well as available or preferred input mechanisms as potential criteria. On the other hand, they mentioned task-oriented UIs that are able to adapt to different interaction procedures preferred by users according to different goals.

3.2. Benefits and Challenges of the Model-driven Approach

Although asked for benefits of the model-driven approach, participants also mentioned personalization as a positive aspect of polymorphic UIs in general. They mentioned their possibility of personal adjustments for each user and their ability to address true user needs. As personalization should lead to higher levels of usability, they expect a higher user satisfaction.

Interestingly, four of the eight the participants referred to benefits regarding the software architecture of applications. Most notably they mentioned the separation of UI and business logic functionality as a key benefit. Positive effects expected to result from this clear structure were code that is easy to change or extend and a reduced need for code refactoring. It is remarkable that one of those participants considered this requirement of separation also as a challenge. One participant expected the implementation and testing phases of the development process to be easier to plan and estimate.

As expected the difficulty of predicting how the UI will finally be presented to users was one of the challenges of the model-driven development approach mentioned in the questionnaires. Another aspect closely connected to this was testing. Participants expected testing for usability and accessibility to get harder together with an increased complexity especially for user tests.

The participants also anticipated an increased skill level to be necessary for developers. They thought that developers were required to learn new development approaches and needed to gain a reasonable understanding of the rules and patterns that generate the final UI. According to the participants, this applied also to requirement engineers, as user requirements need to be analyzed in a broader and more detailed manner.

Since an application will incorporate a number of different UI variants, the participants expected the size of the software to grow. This might lead to increased effort, especially in maintenance.

3.3. Comparison of Abstract and Model-with-Example Editors

The prototypes presented for both model editor concepts have been aligned to the typical structure of modern IDEs. Participants recognized this structure in both prototypes and rated this familiarity as a benefit. But embedding the editors into an IDE frame also led to the objection that some parts of the UI have not been used in both use cases. Another commonality of both editors seems to be a lack of guidance through the modelling process. In this context, participants mentioned hints regarding next steps for novice users as well as information on the completeness of the model compared to the requirements.

The abstract visualization has been recognized by the participants as the key difference between the first prototype and the Mode-with-Example editor. It led to the central benefits of the abstract notation: it does not suggest a concrete UI, focuses on transitions and functions and thus provides an overview over possible states and interactions of the modelled application. According to the participants' feedback, the challenge that comes along with this abstract visualization is that the UI variants are not visible and it remains unclear how the UI will finally be presented to the users. It has also been mentioned that the UML-like notation may be too abstract for unexperienced developers.

In contrast, they considered the wireframe-like visualization as the key benefit of the Model-with-Example editor. It allows developers to see both the interaction structure and the final view of the UI by displaying the content of the states together with the transitions. The profile drop-down list boxes enable developers to switch between variants fast and easy to gather an overview of the polymorphic UI. One participant even noticed that in the prototype there was no drop-down list box for profiles explicitly covering environmental conditions such as the luminosity of ambient light. Noteworthily, another participant expressed the expectation that this visualization would result in higher development efforts because developers would check all UI variants.

The participants indicated that on one hand the Model-with-Example visualization improves the understanding of the model under creation. On the other hand, it increases the visual complexity of the representation which may become confusing or cumbersome especially in cases of larger models.

The interaction with the model editor itself has been rated as faster and generally better in the Model-with-Example prototype. Compared to that, the participants expected the abstract editor to require more interaction effort.

3.4. Generation of Storyboards

The storyboard feature predominantly earned positive feedback. Most of the participants provided comments related to testing and quality assurance in general. For example, the generated storyboards may serve as test cases defining the course of interaction. They may also be used to compare the current UI against the defined target or real user behavior, identifying unnecessary or missing parts of the model. In addition, participants expected that by defining the stories developers would encounter long interaction sequences that might be shortened. From the participants' point of view, all of these will improve the quality of the resulting UI.

According to the participants, the generated storyboards with wireframes will also be helpful in project internal documentation and manuals handed over to clients and users. They also requested a special presentation mode and export functions for the generated storyboards. Considering other comments, these functions would help in discussions with other stakeholders by providing a clean and simple way of creating and presenting specific scenarios.

Both applications, documentation and project communication, are based on another benefit mentioned in the questionnaires. Storyboards could help to handle and clarify complex models by focusing on a single course of interaction. On the downside, participants also stated this might also reduce the awareness for the structure of the whole system. Especially, it was not obvious to all how cycles and self-references in the model are represented in the storyboard.

Overall, the participants expected the generation of storyboards to speed up the development process although it would require additional efforts for the initial story definition.

4. Discussion and Outlook

When analyzing the results of the user study it seems reasonable to build upon current approaches like responsive web design to mainstream model-based development of polymorphic UIs. There has been a strong emphasis on web-based and mobile applications together with device-oriented aspects.

As expected transparency of the generation process and prediction of the resulting UIs seem to be relevant issues for developers. The participants mentioned these both when referring to the model-based approach itself as well as when they were comparing the editor prototypes. According to the comments, Model-with-Example seems to be a reasonable approach to mitigate these issues.

Despite the potential issue of current software architectures that do not clearly separate UI and business logic, the participants reported no essential development practices that would be in conflict with the presented approach. On the contrary, it might even improve software quality in terms of testing, documentation and project communication.

The incorporation of the MBUID in mainstream software development would certainly change how developers' work. It separates the creation of modular UI solutions from the modeling of the interaction for a specific application or service. While the Model-with-Example editor presented here contributes towards the acceptance of this development approach, the value the generated UIs finally will provide to users heavily depends on the quality of the available modular UI solutions together with the generation rules and mechanisms.

Thus, while this paper focuses on the development of specific applications, future work will be required to cover the procedures for the creation and development of the modular UI solutions. One central question of this research will be how to manage the quality of those modular UI solutions to ensure the generated UIs are of value for a widest range of users.

Acknowledgement

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

References

[1] Nussbaum, G., Veigl, C., Acedo, J., Barton, Z., Diaz, U., Drajsajtl, T., ... & Paspallis, N. (2011). AsTeRICS – Towards a rapid integration construction set for assistive technologies. In *AAATE Conference 2011* (pp. 766-773). IOS Press.

[2] Savidis, A., & Stephanidis, C. (2004). Unified user interface design: designing universally accessible interactions. *Interacting with computers*, *16*(2), 243-270.

[3] Gajos, K. Z., Wobbrock, J. O., & Weld, D. S. (2007). Automatically generating user interfaces adapted to users' motor and vision capabilities. In *Proceedings of the 20th annual ACM symposium on User interface software and technology* (pp. 231-240). ACM.

[4] Peissner, M., Häbe, D., Janssen, D., & Sellner, T. (2012). MyUI: generating accessible user interfaces from multimodal design patterns. In *Proceedings of the 4th ACM SIGCHI symposium on Engineering interactive computing systems* (pp. 81-90). ACM.

[5] 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.

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

[7] Peissner, M., Schuller, A., Ziegler, D., Knecht, C., & Zimmermann, G. (2014). Requirements for the successful market adoption of adaptive user interfaces for accessibility. In *International Conference on Universal Access in Human-Computer Interaction* (pp. 431-442). Springer International Publishing.

[8] Object Management Group (2015). *OMG Unified Modeling Language™ (OMG UML), Version 2.5*. http://www.omg.org/spec/UML/2.5