

Usability Evaluation of Model-Driven Cross-Device Web User Interfaces

Enes Yigitbas, Anthony Anjorin, Ivan Jovanovikj, Thomas Kern, Stefan Sauer, Gregor Engels

▶ **To cite this version:**

Enes Yigitbas, Anthony Anjorin, Ivan Jovanovikj, Thomas Kern, Stefan Sauer, et al.. Usability Evaluation of Model-Driven Cross-Device Web User Interfaces. 7th International Conference on Human-Centred Software Engineering (HCSE), Sep 2018, Sophia Antipolis, France. pp.231-247, 10.1007/978-3-030-05909-5_14 . hal-02270698

HAL Id: hal-02270698
https://inria.hal.science/hal-02270698

Submitted on 26 Aug 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Distributed under a Creative Commons Attribution| 4.0 International License

Usability Evaluation of Model-Driven Cross-Device Web User Interfaces

Enes Yigitbas[1], Anthony Anjorin[1], Ivan Jovanovikj[1], Thomas Kern[2], Stefan Sauer[1], and Gregor Engels[1]

[1] Paderborn University
Zukunftsmeile 1, 33102 Paderborn, Germany
firstname.lastname@upb.de,
[2] Diebold Nixdorf AG,
Heinz-Nixdorf-Ring 1, 33106 Paderborn, Germany
thomas.kern@dieboldnixdorf.com

Abstract. User Interface (UI) development is a challenging task as modern UIs are expected to be available across a wide range of diverse platforms while assuring high usability for heterogeneous users. Model-driven engineering principles have been applied in the context of multi-device and cross-device UI development to tackle complexity in development. While previous work related to usability evaluation of model-driven UIs primarily focused on single- and multi-device UIs, an investigation of the usability of model-driven cross-device UIs was not fully covered yet. In this paper, therefore, we present a model-driven UI development (MDUID) approach for cross-device UIs and analyze whether the applied MDUID approach has a positive impact on the usability of the generated UI. To accomplish this, we conduct a usability evaluation based on the generated UI for a cross-channel banking web application. The usability evaluation results provide detailed feedback regarding fulfillment of different usability criteria and enable improvement of involved models as well as model transformations.

Keywords: Model-driven UI Development, Usability, Cross-Device UIs

1 Introduction

Nowadays users are surrounded by a broad range of networked interaction devices (e.g. smartphones, smartwatches, tablets, terminals etc.) for carrying out their everyday activities. The number of such interaction devices is growing, new possible interaction techniques are emerging and modern UIs are expected to be available across a wide range of diverse platforms.

In the context of our research and development project with our industrial partner Diebold Nixdorf AG[3], we have focused on the development of cross-device user interfaces that can span across various platforms and support a cross-channel banking experience.

[3] https://www.dieboldnixdorf.com

Figure 1 shows the underlying example scenario where cross-device UIs are used in the context of a cross-channel banking web application. The idea is that banking services are accessible through different channels and depending on the situation, customers can access the service through any channel they wish to use. For example, if the customers pursue a cross-channel interaction for a payment cashout process, they can begin an interaction using one channel (*Prepare Cashout* at *Desktop-PC* at home), modify the transaction on their way on a mobile channel (*Edit Cashout* via *Smartphone*), and finalize it at the automatic teller machine (*ATM*). As the described scenario covers different devices with varying platform properties and interaction techniques, each target platform has specific needs regarding its UI. Especially the target platform *ATM* has different hardware capabilities. In the example scenario, the coupling between *Smartphone* and *ATM* can be established through *Authorization via NFC* technology or via the classical way using a banking card (*Authorization via Card*). For authentification, an eye-tracker is integrated into the ATM where the customer can enter a password by gazing at predefined password symbols (*Authentification via VisualPin*) .

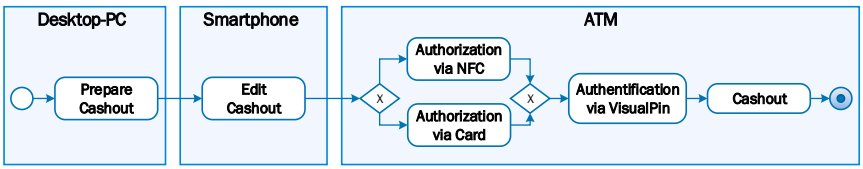


Fig. 1: Example Scenario

As the motivational example scenario indicates, UI development is a challenging task due to the trade-off between efficiency and usability: creating different UIs available across various devices while assuring high usability for heterogeneous users.

In the past, model-driven user interface development (MDUID) approaches were proposed to support efficient development of UIs. Widely studied approaches are UsiXML [9], MARIA [16], and IFML [3] that support the abstract modeling of user interfaces and their transformation to final user interfaces. Although those MDUID approaches increase the efficiency and consistency in the generation of multiple UI variants, existing usability studies show that completely automated approaches are not optimal in terms of the quality regarding the usability of the UI [2].

As usability is an important criterion for user acceptance and user experience of interactive systems, usability issues have to be taken into account when developing multiple variants of a UI, especially for a cross-device usage scenario as in our case. Therefore, a proper investigation of applying MDUID for cross-device

UIs and its impact on the usability of the generated UI is essential to identify usability problems and determine whether acceptance by the user is given.

While previous work in the context of usability evaluation of model-driven UIs covered single- [1] and multi-device [2] UIs, our goal is to investigate the usability of generated cross-device UIs. In this regard, our analysis focuses on the usability evaluation of model-driven cross-device UIs and the user's perception of the cross-device interaction based on our MDUID approach. By conducting a usability evaluation experiment with different users, we aim to understand current limitations of our MDUID approach for cross-device UIs and identify improvement potential.

The remaining sections of this paper are organized as follows: Section 2 presents our model-driven cross-device UI development approach. In Section 3, we describe the setup of our usability evaluation study and present the main results. Related work is presented in Section 4 and finally Section 5 concludes the paper and gives an outlook on future work.

2 Model-Driven Cross-Device UI Development Approach

In this section, we recapture our approach for model-driven cross-device UI development from previous work [18] and show its application for the motivated cross-channel banking scenario.

2.1 MDUID Approach for Cross-Device UIs

An overview of our model-driven cross-device UI development approach is depicted in Figure 2. The model-driven development process for cross-device UIs is divided into three phases. In the first phase, *Modeling*, a *Domain Model* described as a UML class diagram and an *Abstract UI Model* based on the Interaction Flow Modeling language (IFML)[4], serve as specification of the data entities as well as structure, content and navigation needed to characterize the UI in an abstract manner.

The second phase, *Transformation*, deals with the automated transformation of the *Abstract UI Model* to different final UI (*FUI*) representations for the varying target platforms PC, ATM and Smartphone. For this purpose, several model-to-text transformation *(M2T)* templates were defined that transfer the *Abstract UI Model* to the *final UIs*.

The last phase, *Execution*, is dedicated to executing the heterogeneous UI views on different target platforms. For this purpose, a common UI framework can be used were data is exchanged between different platforms/channels. In order to support a seamless handover between channels and allowing task-continuity for the user, our approach includes an *Application and Synchronization Server*, which is responsible for storing and sharing of data (e.g. UI state or user preferences). The UI state, including entered input data by the users,

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

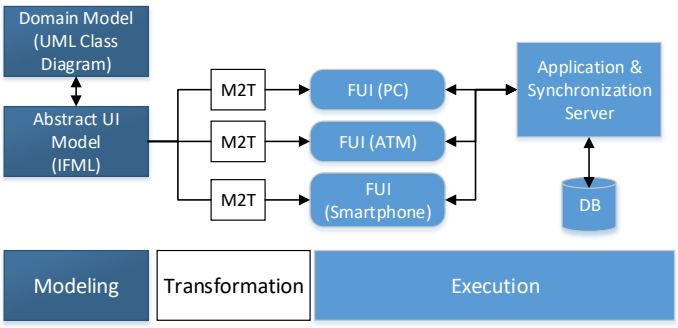


Fig. 2: Model-Driven Cross-Device UI Development Approach

is stored and restored, allowing users to move across channels while seamlessly continuing their tasks.

2.2 Application of the Approach

In the following, we describe how the above described model-driven approach was applied to the motivated example scenario (see Figure 1) for generating cross-device UIs for the cross-channel web application.

Based on the existing IFML Editor Eclipse plugin [5], developers are able to specify the domain and abstract UI model.

In Figure 3 an excerpt of the specified domain model is shown in form of a UML class diagram. The depicted domain model covers main concepts and relations to represent a banking application. In our example case, each *User* of the cross-channel banking web application has at least one account. To each account any number of a *BankCard* and *Transaction* are assigned. A transaction includes the respective attributes like date, amount of money, balance etc. Each transaction can in turn be detailed in form of a *Denomination* describing the specific banknotes that are wished to be withdrawn. The enumeration on the right side of the domain model show possible expressions for certain data types, such as bank card type or user profile.

After specifying the domain model, we specify the abstract UI model based on IFML. Figure 4 shows a small excerpt from the IFML model that is characterizing the UI for our cross-channel banking web application. Essentially, one can see the abstract UI modeling of the masks for the registration, login, and main windows. The three main windows themselves contain more elements to enable a detailed specification of the interaction objects nested therein. For example in the login window, there is a form "LoginForm", which in turn has two input fields (SimpleField) for entering username and password. The *LoginWindow* contains also events - (shown in a circle) - such as *"Register"* or *"Log in"* for

[5] http://ifml.github.io

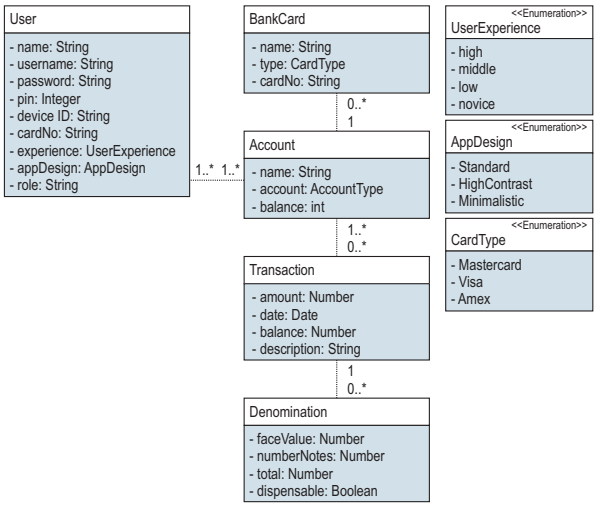


Fig. 3: Excerpt of the Domain Model for the Cross-Channel Banking Application

triggering actions from the business logic (see for example Action *"Register"*). If you follow the event *"Register"* in the login window, the corresponding navigation edge follows to the registration window. In a similar way, the other masks *"RegistrationWindow"* and *"MainWindow"* are modeled to specify the abstract UI model.

The specified domain model and the IFML model serve as input for our code generator to transform them to the specific final UIs for the varying platform. For transforming these models into final web UI views, we implemented an Xtend[6] plugin that maps the IFML model elements to specific HTML5 elements. The Xtend plugin includes different Xtend templates to transfer the IFML source model into web UIs supporting manifold devices. During the transformation process, the application's view is built upon basic components with a custom look & feel, like buttons, text input fields, dropdown lists, tables, etc. As a basis for these components, we implemented components based on the *HTML5 Web Components*[7] specification promoted by Google as W3C standard. Our custom components are sensitive to the application environment they are being used in (desktop, mobile, ATM) and adapt themselves accordingly. On mobile devices, for example, buttons are larger and more suitable for touch operation than on desktop devices. During the execution phase, the generated web views build up a HTML5/JavaScript single-page application running in a web browser. It exchanges JSON messages with the back-end server through HTTP/REST. The back-end server is implemented in JavaScript and uses Node.js[8] as its runtime environment. It is built upon Google's V8 JavaScript engine also used by Google

[6] http://www.eclipse.org/xtend
[7] https://www.w3.org/TR/components-intro
[8] https://nodejs.org

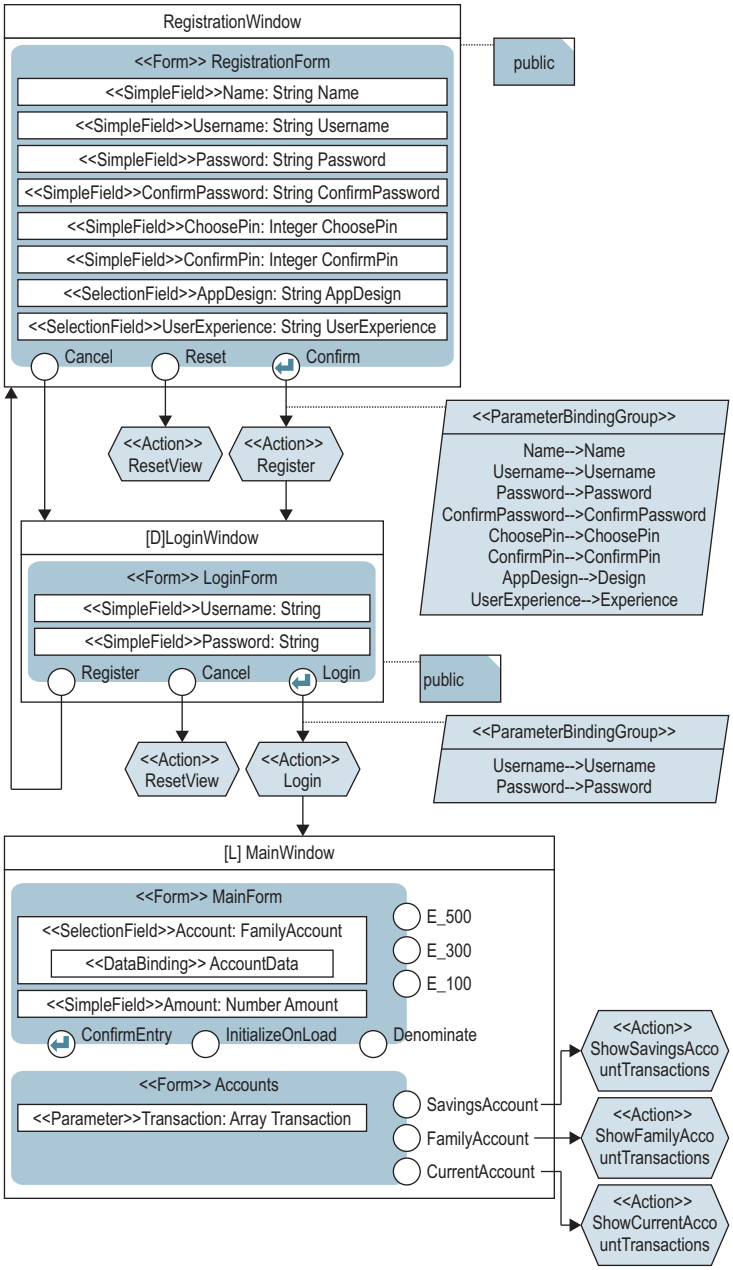


Fig. 4: Excerpt of the IFML Model for Cross-Channel Banking Application

Chrome and provides a high-performance runtime environment for non-blocking and event-driven programming. In order to support task continuity and transfer application state between devices, the current state name and its associated context are saved to the Application & Synchronization Server. Inside a view controller and prior to saving a state, all context information necessary for recovery is added to a state-context object. This includes the UI's view-model, as well as any other necessary information associated with the current state. To invoke a previously saved state, the application just needs to retrieve the current state name and invoke it.

Figure 5 shows the different generated final UIs for our cross-channel banking web application. To be more specific, the *MainWindow* for the cash withdrawal process is shown for each involved device Desktop, Smartphone and ATM.

More technical details regarding the transformation and execution phases can be found in our previous work [18] where we present used technologies, languages and frameworks for the generation and execution phases in more detail.

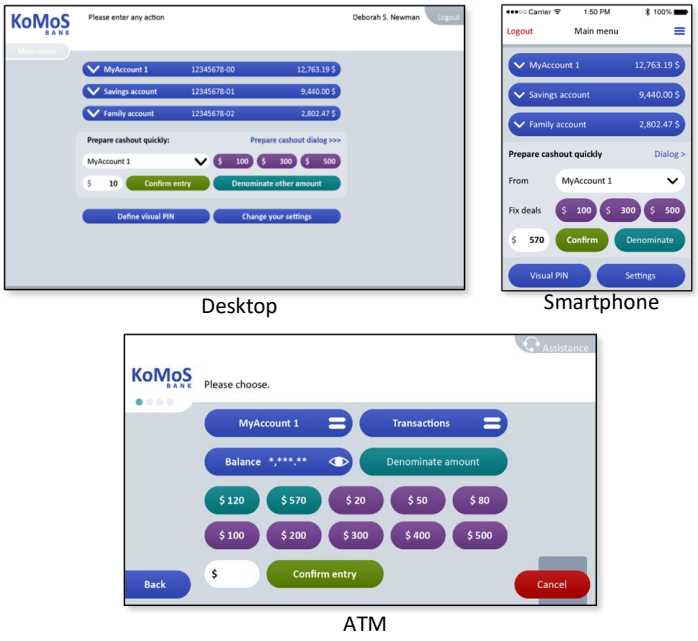


Fig. 5: Generated UIs for different device types

3 Usability Evaluation

While the previous section recapitulated our existing solution for model-driven cross-device UI development and showed its application in the context of a cross-channel banking application, the main goal of the paper is to analyze the usability of such a generated cross-device user interface.

Therefore, in the course of our industrial research project we conducted a usability test to assess the quality of model-driven cross-device web UIs. The goal was to evaluate how well or poorly the generated cross-channel banking web application called "KoMoS" and its user interface performed, and how satisfied the users were in completing two withdrawal tasks, one based on the classical way using a single access point on the ATM and one based on a cross-device usage scenario. In this section, we first describe the setting and procedure of our usability study. After that, we present the results of our usability study measured in terms of effectiveness, efficiency and satisfaction as suggested in ISO 9241-11 [8].

3.1 Setup Usability Study

We set up a usability laboratory at Diebold Nixdorf AG to conduct the usability test. Figure 6 shows the general setting of our usability lab. The usability test area contains the involved devices for the cross-device interaction: *Desktop-PC*, *Smartphone*, and *ATM*. Additionally, there are two cameras that serve to record the user interaction with the overall system. *Camera A* focuses the user, to track user behavior (facial expressions, gestures, etc.) and *Camera B* serves to observe the input of the user. Beside the test area, we have a question answering area, where participants answer an online questionnaire using a tablet after completing their task.

The test involved 15 persons where we had five participants from Paderborn Universities (mainly CS students), five employees from Diebold Nixdorf AG (from different departments like marketing, R&D, etc.) and five elderly persons where two of them lived in a nursing home. The interaction between the user and the generated KoMoS banking application was video-recorded and logged. An instructor and one independent observer participated in the study. The instructor interacted with the user while the observer took a record of observations and assisted during the usability test. Users were informed that the usability of the generated KoMoS banking application was going to be tested to check whether the system met their needs. No pretask training was scheduled and all of the participants were going to see the generated application for the first time. The design of the usability test followed a logical sequence of events for each user and across tests. The general usability testing procedure was as follows:

- Users were given information about the goals of the test and the setup of the usability laboratory. They were also informed about the video recording for analysis purposes.

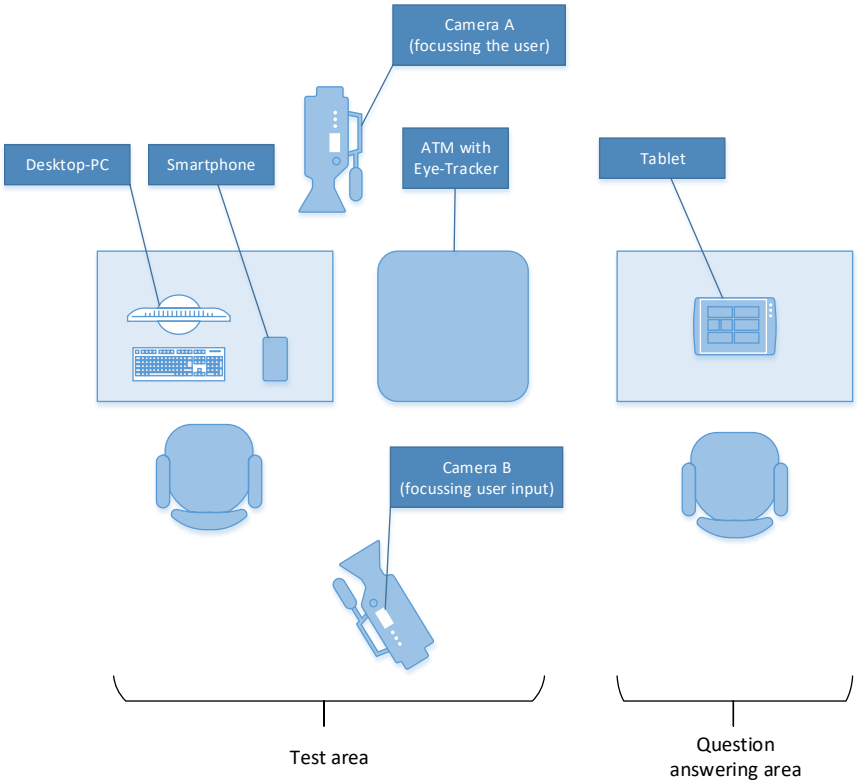


Fig. 6: Usability Setting

– Users were given a series of clear instructions that were specific for the test. They were advised to try to accomplish the tasks without any assistance, and that they should only ask for help if they felt unable to complete the task on their own. Users should also use the think-aloud technique while accomplishing the task, so that possible usability problems could be analyzed.
– Users were asked to complete two different types of tasks. The first task was to withdraw a predetermined amount of money from the ATM - a classical cashout at the ATM using a bank card and PIN. For this task, the participants received a bank card with the corresponding PIN. The second task was to withdraw money based on the cross-device interaction scenario, so the users had to start preparing a cashout at the Desktop-PC, edit the cashout at the smartphone and finalize it at the ATM. For avoiding a possible ceiling effect, there was no time limit to complete the tasks.
– Users were then asked to fill in an online questionnaire after completing the last task. This questionnaire is arrranged in a hierarchical format and con-

tains: 1) a demographic questionnaire which is used to confirm their age, job description and gender. They also scored their attitude towards technology usage and especially their experience with the involved device types. 2) several measures of specific interface factors based on the IsoMetrics questionnaire [5] for the evaluation of graphical user interfaces (ISO 9241/10). Each item of this questionnaire is rated on a scale from one to seven with positive statements on the right side and negative statements on the left side. Additional space that allows the users to make comments is also included in the questionnaire. 3) Additional specialized questions regarding cross-device interaction and usage of eye-tracking were incorporated in the questionnaire to assess the specific details of the KoMoS banking scenario.
- After finishing the test, users and the instructor had the possibility to talk about their experience and exchange ideas about lessons learned. Users were given a small reward for their participation

3.2 Results of the Usability Study

In this section, we discuss the results and findings of the usability study regarding the criteria effectiveness, efficiency and satisfaction.

Effectiveness Effectiveness relates the goals of using the product to the accuracy and completeness with which these goals are achieved [8]. It was measured using the following measures proposed in the Common Indutry Format (CIF) [17] for usability tests: the completion rate and the frequency of assists. The completion rate is the percentage of participants who completed each task correctly. The frequency of assists is the number of times that the instructor assisted the participant. Regarding effectiveness our usability study showed that each participant except one older woman (with the age of 86 and big vision problems) was able to complete both tasks. In some few cases the instructor had to give some smaller hints to complete the task: for example that the cashout preparation step was successfully accomplished and that the participant could move to the next step at the ATM or if the participant was unsure about the input possibilities a small hint was provided that input via eye-tracking or pin pad was possible. Nevertheless the assisted completion rate for both tasks was 14 out of 15 participants.

Some observed usability problems regarding effectiveness:

- Layout / Design : The denomination dialogue which supports the selection of specific banknotes during the withdrawal process was not intuitive for all users. Some of the elderly participants for example clicked directly on the banknote symbols (which were inactive by construction) instead of the "'+"' Symbol for selecting specific banknotes. Furthermore the structure of the menu was not optimal for some users. Although the most important functions were directly accessible, some participants needed more time to get used to the menu and the different button types which were used in the KoMoS banking application.

- System Feedback: The participants missed at some points in the system dialog feedback from the system which helps them to know about the next steps. For example, some of the users were not sure after preparing the cashout on the smartphone if they have finished this task. Furthermore, the users wished feedback on possible input techniques. As we have different input methods (Pin-Pad, Touchscreen and Eye-Tracker on the ATM) they were not sure when to use which input method.
- Difficulty with operating the application via eye-tracking: The users were not familiar with controlling an application via eye-tracking. So different questions arose: Where can one look without causing a wrong entry? How long I have to gaze at a specific UI element (symbol) to select it?

Efficiency Efficiency relates to the resources expended in relation to the accuracy and completeness with which users achieve goals [8]. For the evaluation of the efficiency, the required time for the different two tasks was measured. Table 1 shows a comparison between the needed time for accomplishing both tasks. For accomplishing the classical cashout process using the generated KoMoS banking application only on the ATM, the participants needed approximately 64 seconds in average. Compared to this reference scenario the completion of the task in the cross-device scenario takes approximately 135 seconds in average. In this regard, we should notice that the clasical cahout process is a common known activity for all participants (although the used app is new). In contrast, most of the users were not very familiar with the cross-device scenario where different devices and interaction techniques were involved. Nevertheless, in the cross-device scenario we can observe that the needed time for finalizing the withdrawal process at the ATM takes only approximately 46 seconds in average since the transaction has been already prepared on the smartphone. The majority of the overall time with 135 seconds in average was needed for preparation at the smartphone (89 seconds in average). Thus, the stay at the ATM is reduced by about 25 percent by a cross-device interaction compared to the classical ATM cashout process. If we imagine that the preparation of the transaction on the smartphone is done in an idle time, this can be seen as a time saving.

Satisfaction Satisfaction is defined as freedom from discomfort, and positive attitudes towards the use of the product [8]. For the evaluation of the satisfaction criteria, we measured and analyzed the questionnaire answers of the participants. As depicted in Figure 7 the evaluation of the satisfaction criteria shows a positive

Table 1: Efficiency: Comparison between ATM reference scenario and Cross-device scenario

Scenario	ATM	Smartphone	Total
Reference (ATM-only)	64sec	-	64sec
Cross-Device	46sec	89sec	135sec

feedback of the participants regarding the generated UI of the KoMoS banking application. In total average, the interaction with the UI received the score 5,2 out of 7 which is quite acceptable. While, participants especially honor the criteria like ease-of task handling, learnability or liquid usage across different device types, they also point out that some aspects like system feedback, personalization and adaptability are not optimally solved through the generated UI of the KoMoS banking application.

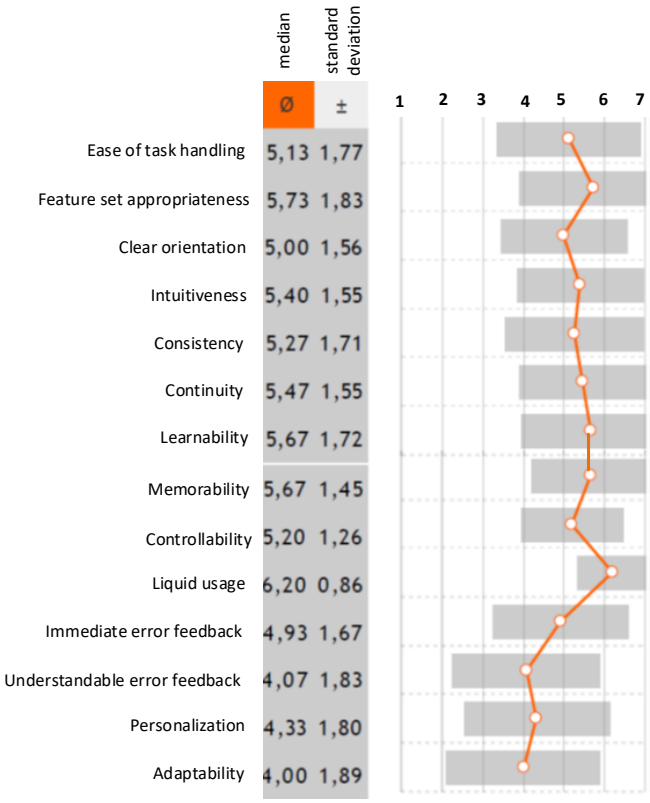


Fig. 7: General usability results regarding satisfaction criteria

The second part of the questionnaire covers specific aspects about the cross-device baking scenario. In this regard, user's satisfaction of the generated UI regarding authorization via card vs. smartphone (NFC), authentification via PIN vs. Eye-Tracking and the device switches were assessed based on questions. Figure 8 shows that there are not big differences in the authorization via card or smartphone regarding comfortability and quickness. However, the participants

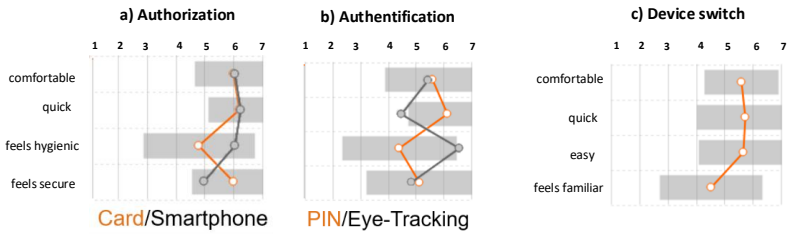


Fig. 8: Usability results regarding specific aspects

honor authorization via smartphone (NFC) as more hygienic. In contrast to that, classical authorization via bank card is perceived as more secure by the users. Regarding the both authentification types, the classical solution based on PIN entry is seen as more quick but rather unhygienic, while the Eye-tracking authentification takes longer but increases the satisfaction regarding hygiene as the users do not have to touch the Pin-Pad.

In summary, the usability evaluation results show that cross-device UIs automatically generated based on our model-driven approach are accepted by the majority of the users. The high rate of effectiveness for both tasks indicates that it is possible to automatically generate the UI for a cross-channel application scenario with the needed functionality. However, detailed feedback regarding satisfaction and user comments also show that there is improvement potential. Especially the aspects layout design, system feedback, personalization and adaptability show potential for further improvements. In this connection, existing model-driven UI development approaches have to be incorporated with explicit modeling and transformation techniques for better supporting user assistance and adaptation.

4 Related Work

In recent years, a number of approaches have addressed the problem of UI development for multi-device and cross-device user interfaces. While there are several existing approaches in this direction, the number of usability evaluation studies focusing on UIs generated based on a model-driven approach is limited to a few examples. In this section, we especially review prior work that explores the development of multi-/cross-device user interfaces and usability evaluation of model-driven user interfaces.

4.1 Multi-/Cross-device UI Development approaches

The development of multi-device UIs has been subject of extensive research [14] where different approaches were proposed to support efficient development of UIs for different target platforms. Model-based and model-driven UI development approaches were proposed to create multi-device UIs based on the transformation

of abstract user interface models to final user interfaces. Widely studied approaches are UsiXML [9], MARIA [16], and IFML [3] that support the abstract modeling of user interfaces and their transformation to final user interfaces.

Previous work by the research community has also covered concepts and techniques for supporting the development of cross-device user interfaces. One of the concepts here is called UI migration, which follows the idea of transferring a UI or parts of it from a source to a target device while enabling task-continuity through carrying the UI's state across devices. In [15] for example, the authors present an agent-based solution to support migration of interactive applications among various devices, including digital TVs and mobile devices, allowing users to freely move around at home and outdoor. The aim is to provide users with a seamless and supportive environment for ubiquitous access in multi-device contexts of use. A more recent model-based approach which allows designers and developers to specify how to distribute interfaces at various granularity levels, ranging from entire user interfaces to parts of single interactive elements is presented in [11]. This solution includes run-time support for keeping the resulting user interfaces synchronized and customization tools that allow end users to dynamically change how the user interface elements are distributed across multiple interactive devices in order to address unforeseen situations.

In the case of web applications, most solutions rely on HTML proxy-based techniques to dynamically push and pull UIs [6]. An extension of this concept is presented in [13], where the authors propose XDStudio to support interactive development of cross-device UIs. In addition, there is also existing work on the specification support for cross-device applications. In [21] for example, the authors present their framework Panelrama which is a web-based framework for the construction of applications using distributed UIs. In a similar work [7], the authors present Conductor, which is a prototype framework serving as an example for the construction of cross-device applications.

4.2 Usability evaluation approaches

Due to the low number of usability evaluation studies existing for model-driven UIs and different scope of analysis criteria, it is very difficult to make comparisons among the existing usability evaluation approaches. In earlier work, Chesta et al. evaluated a multi-platform user interface generated by TERESA [12] according to several criteria: tool interface (intuitiveness, learnability), tool functionalities (completeness, developer satisfaction), final product obtained by employing the tool (user satisfaction, maintainability and portability), and approach cost/effectiveness (development efficiency, integrability). Their results suggest that the usage of the MDE approach improved some of these metrics compared to a manual approach where the user interface is manually produced.

Abrahao et al. [1] conducted an experimental study testing the usability of user interfaces that were automatically produced by MDE techniques. Based on their usability evaluation study the authors were able to identify main usability problems for the generated user interface. They argue that valuable feedback based on such experimental studies can be used to improve the used models and

their transformation to final UIs. In this context, they also underline the concept of *usability proven by construction* as the insights from the usability study can be used for defining design guidelines or anti-patterns for developing highly usable UIs.

A further usability study for multi-device UIs generated by MDE techniques was presented by Aquino et al. [2]. The authors describe an MDE approach that generates multi-platform graphical user interfaces (e.g., desktop, web) that are subject to an exploratory controlled experiment. The usability of user interfaces generated for the two mentioned platforms and used on multiple display devices (i.e., standard size, large, and small screens) were examined in terms of satisfaction, effectiveness and efficiency. The results of the paper suggest that the tested MDE approach should incorporate enhancements in its multi-device/platform user interface generation process in order to improve its generated usability.

While above mentioned approaches focus on the usability evaluation of generated UIs for single or multi-device platforms, our goal is to analyze usability of generated UIs across different devices. Therefore we especially take into account the perception of the users regarding the cross-device interaction. A similar usability evaluation approach to our study is presented in [10] where the authors introduce a cross-platform usability analysis model. While this approach provides a conceptual framework for analyzing usability of cross-device UIs, it does not focus on generated UIs using MDE techniques and also do not provide an experimental study in this regard.

5 Conclusion and Outlook

In this paper, we presented a usability study for cross-device user interfaces that were automatically generated based on a model-driven approach. We conducted a usability test with 15 different users based on a cross-channel banking web application to analyze strength and weaknesses in the usability of the generated cross-device UIs regarding effectiveness, efficiency and satisfaction. The usability evaluation results serve as an indicator to identify usability problems and can be used to further improve the existing model-driven UI development approach that was applied.

In ongoing work we are focusing on the usability evaluation of adaptive UIs that have been promoted as a solution for context variability due to their ability to automatically adapt to the context-of-use at runtime. Therefore, we have established a model-driven engineering approach [19] for adaptive UIs including an authoring environment [20] and design a suitable usability evaluation method for adaptive UIs.

Acknowledgement

This paper is based on some of the work within “KoMoS”, a project of the “it’s OWL” Leading-Edge Cluster, partially funded by the German Federal Ministry of Education and Research (BMBF). We also acknowledge the support of the

project "Mittelstand 4.0" funded by the German Federal Ministry for Economic Affairs and Energy.

References

1. S. Abrahao, E. Iborra, J. Vanderdonckt. 2008. Usability Evaluation of User Interfaces Generated with a Model-Driven Architecture Tool. In: Law E.LC., Hvannberg E.T., Cockton G. (eds) Maturing Usability. Human-Computer Interaction Series.
2. N. Aquino, J. Vanderdonckt, N. Condori-Fernandez, O. Dieste, and O. Pastor. 2010. Usability evaluation of multi-device/platform user interfaces generated by model-driven engineering. In Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM '10).
3. M. Brambilla, and P. Fraternali. 2014. Interaction Flow Modeling Language - Model-Driven UI Engineering of Web and Mobile Apps with IFML. The MK/OMG Press.
4. C. Chesta, F. Paterno', and C. Santoro. Methods and Tools for Designing and Developing Usable Multi-Platform Interactive Applications. PsychNology Journal 2, 1:123139, 2004.
5. G. Gediga, K. Hamborg, I. Duentsch. 1999. The IsoMetrics usability inventory: an operationalisation of ISO 9241-10. In: Behaviour and Information Technology, 18 (1999), pp. 151-164.
6. G. Ghiani, F. Paterno', and C. Santoro. 2012. Push and pull of web user interfaces in multi-device environments. In Proceedings of the International Working Conference on Advanced Visual Interfaces (AVI '12). ACM, New York, NY, USA, 10-17.
7. P. Hamilton and D. Wigdor. 2014. Conductor: enabling and understanding cross-device interaction. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '14). ACM, New York, NY, USA, 2773-2782.
8. International Organization for Standardization. ISO 9241-11: Ergonomic requirements for office work with visual display terminals (VDTs) Part 9: Guidance on usability, 1998.
9. Q. Limbourg and J. Vanderdonckt. 2004. USIXML: A User Interface Description Language Supporting Multiple Levels of Independence. In Engineering Advanced Web Applications: Proceedings of Workshops in connection with the 4th International Conference on Web Engineering. Rinton Press, 325-338.
10. K. Majrashi, M. Hamilton, and A. Uitdenbogerd. 2014. Cross-platform usability and eye-tracking measurement and analysis model. In Proceedings of the 26th Australian Computer-Human Interaction Conference on Designing Futures: the Future of Design (OzCHI '14). ACM, New York, NY, USA, 418-421.
11. M. Manca and F. Paterno'. 2016. Customizable dynamic user interface distribution. In Proceedings of the 8th ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS '16). ACM, New York, NY, USA, 27-37.
12. G. Mori, F. Paterno', and C. Santoro. Design and Development of Multidevice User Interfaces through Multiple Logical Descriptions. IEEE Trans. Software Eng., 30(8):507520, 2004.
13. M. Nebeling, T. Mintsi, M. Husmann, and M. Norrie. 2014. Interactive development of cross-device user interfaces. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '14).
14. F. Paterno' and C. Santoro. 2012. A logical framework for multi-device user interfaces. In Proceedings of the 4th ACM SIGCHI symposium on Engineering interactive computing systems (EICS '12). ACM, New York, NY, USA, 45-50.

15. F. Paterno', C. Santoro, and A. Scorcia. 2010. Ambient Intelligence for Supporting Task Continuity across Multiple Devices and Implementation Languages. Comput. J. 53, 8 (October 2010), 1210-1228.
16. F. Paterno' and C. Santoro, and L. D. Spano. 2009. MARIA: A Universal, Declarative, Multiple Abstraction-Level Language for Service-Oriented Applications in Ubiquitous Environments. ACM Transactions on Computer-Human Interaction 16(4),19:1-19:30.
17. J. Scholtz. 2000. Common industry format for usability test reports. CHI Extended Abstracts.
18. E. Yigitbas, T. Kern, P. Urban, S. Sauer. 2016. Multi-device UI Development for Task-Continuous Cross-Channel Web Applications. In: Casteleyn S., Dolog P., Pautasso C. (eds) Current Trends in Web Engineering. ICWE 2016. Lecture Notes in Computer Science, vol 9881.
19. E. Yigitbas, H. Stahl, S. Sauer, G. Engels. 2017. Self-adaptive UIs: Integrated Model-Driven Development of UIs and Their Adaptations. In: Anjorin A., Espinoza H. (eds) Modelling Foundations and Applications. ECMFA 2017. LNCS, vol 10376. Springer.
20. E. Yigitbas, S. Sauer, and G. Engels. 2017. Adapt-UI: an IDE supporting model-driven development of self-adaptive UIs. In Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS '17). ACM, New York, NY, USA, 99-104.
21. J. Yang and D. Wigdor. 2014. Panelrama: enabling easy specification of cross-device web applications. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '14). ACM, New York, NY, USA, 2783-2792.