Proceedings of the

1st International Workshop on

Large-scale and Model-based Interactive Systems

Approaches and Challenges (LMIS 2015)

Co-located with the 7th ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS 2015)

June 23, 2015, Duisburg, Germany

UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR TELEMATIK

TECHNISCHE UNIVERSITÄT DRESDEN

Proceedings of LMIS 2015 Workshop
Workshop on Large-scale and model-based Interactive Systems: Approaches and Challenges, June 23 2015, Duisburg, Germany.

Publication Online-CEUR Proceedings (CEUR-WS.org)
CEUR-WS Vol-1380
Publication Year 2015
`http://ceur-ws.org/Vol-1380/`

Edited by:
Ronny Seiger[1], Bashar Altakrouri[2],
Andreas Schrader[2], Thomas Schlegel[1]

[1]Software Engineering of Ubiquitous Systems Group,
Technische Universität Dresden, Dresden, Germany
[2]Ambient Computing Group,
Universität zu Lübeck, Lübeck, Germany

The workshop was partially supported by the German Federal Ministry of Education and Research (BMBF) (Code - 16SV6369).

GEFÖRDERT VOM

Bundesministerium
für Bildung
und Forschung

CONTENTS

1 INTRODUCTION

Pervasive and ubiquitous computing introduce a new quality of interaction both into our lives and into software engineering. This has led into an unprecedented interest in implying the full potential of the human body's sensory and motor systems for multi-modal interactivity, manifested by new market initiatives for motion gestures, brain-computer interfaces, multi-touch devices, etc. Whilst this new Post-WIMP interaction paradigm provides rich interaction possibilities and fertile ground for innovation, its increasing popularity imposes new critical challenges for the adoption of interaction techniques in real-world ambient spaces. Software becomes increasingly dynamic, requiring frequent changes to system structures, distribution and behaviour. The aforementioned needs and challenges are mainly caused by increased user mobility, increased heterogeneity of available interaction resources, and increased diversity of physical abilities (i.e., diversity of user population).
This workshop discusses various approaches to handle these challenges to support flexible, context-aware and interactive spaces. We put special focus on promising approaches for coping with dynamics and uncertainties inherent to interactive ubiquitous systems, particularly model-based interaction at runtime and large-scale interaction ensembles (i.e., combining and adapting multiple interactions at runtime). The workshop was held as a full day workshop and aimed to provide a forum for discussing new ideas, issues and approaches. It included a keynote speech, presentation of participants' contributions and various forms of interactive discussions concerning the presented topics.

Workshop Topics

In this workshop, we were mainly interested in exposing those challenges and potential approaches for tackling them. The workshop aimed to stimulate a discussion on the aforementioned core research questions by inviting position papers between 4 and 6 pages in length on any of the the following topics:

- Model-driven architecture (MDA) in the context of interactive systems
- Advantages and potential problems of using MDA in the interactive systems domain
- Distributed user interfaces and UI migration at runtime
- Model-driven generation of (intelligent) interfaces
- Tools and frameworks for supporting the model-driven development
- Concepts for context-awareness and self-adaptation of interactive systems
- Requirements, insights and experiences from existing mobile and pervasive settings
- Architectural concepts for dynamic runtime deployment of interaction techniques

- Formal languages, notations, and concepts for describing interactions for NUIs
- Designing and implementing highly adaptive interaction techniques
- Studies on users' diversity in NUI, including age, physical limitations, etc.
- Studies on user challenges in highly adaptive interactive environments
- Analysis of limitations of existing NUI middleware frameworks and systems
- Analysis and evaluation of HCI community practices and norms for disseminating interaction techniques
- Adjustable, customizable, and modular interactive systems

2 WORKSHOP ORGANIZERS

2.1 Ronny Seiger

Software Engineering of Ubiquitous Systems Group

Technische Universität Dresden

01062 Dresden

Germany

ronny.seiger@tu-dresden.de

Ronny Seiger has been a research assistant within the research project VICCI, funded by ESF, at SEUS at Technische Universität Dresden. His research interests include distributed systems architectures, security and privacy, web technologies, business process, event processing, and software engineering. During his studies, he has been a student assistance whithin the projects Theseus/Texo and FlexCloud at the chair for computer networks. In addition, he has been a working student and thesis student within the new business development department at T-Systems Multimedia Solutions GmbH. In the VICCI project, he is responsible for the design and implementation of a dynamic, highly adaptive runtime environment for complex cyber-physical systems, applying means for central and decentral communications, complex event processing and process orchestration.

2.2 Bashar Altakrouri

Ambient Computing Group

University of Lübeck
23562 Lübeck
Germany

altakrouri@itm.uni-luebeck.de

Bashar Altakrouri is currently a senior researcher at the Ambient Computing Group at the Institute of Telematics at Lübeck University. He worked previously as a research associate at the Embedded Interactive Systems group at Lancaster University (Lancaster, The United Kingdom), research assistant at the International School of Digital Media (Lübeck, Germany), intern at the Open University of Netherlands (Netherlands), and Computer Lab Assistant at the Palestine Polytechnic University (Hebron, Palestine). He is mainly involved in designing, prototyping and implementing Context-aware Systems, Internet of Things (IoT), Natural User Interfaces, and Mobile Services and Applications. His work is currently focused on frameworks for deployable and adaptive interaction techniques for inclusive smart interactive environments for elderly and physically challenged users.

2.3 Andreas Schrader

Ambient Computing Group

University of Lübeck

23562 Lübeck

Germany

schrader@itm.uni-luebeck.de

Andreas Schrader is a professor for Ambient Computing and head of the Ambient Computing Working Group at the University of Lübeck realizing interactive and context-sensitive multimedia applications in ubiquitous and pervasive computing systems. Current focus area is the development of concepts for Ambient Assisted Living as a means for serving an ageing society. In a number of third-party funded projects (BMBF and others) the group develops frameworks for context-aware mobile services, dynamic composition of interaction channels in spontaneous device ensembles and ambient health solutions in co-operation with several clinical partners. Prof. Schrader has published more than 75 papers and achieved several awards for best paper (IEEE iThings 2013) and best demo (IEEE Percom 2013, IoT 2012). He has performed lectures at various universities in Germany, Sweden, Lithuania and Latvia. He is member of ACM and GI, committee member for many international scientific conferences and journals and acts as reviewer for German and Austrian national boards. He is also holding patents in Germany, Japan and the U.S.

2.4 Thomas Schlegel

Software Engineering of Ubiquitous Systems Group

Technische Universität Dresden

01062 Dresden

Germany

thomas.schlegel@tu-dresden.de

Thomas Schlegel is heading the Junior Professorship SEUS at the Technical University of Dresden since 2010. He contributed to more than 60 publications, numerous activities in program committees as well as reviewer and various academic courses and scientific cooperation, he engages in research and academics in the field of Software Engineering of Ubiquitous Systems, focusing on interaction, models, processes and software systems. He perviously worked for different companies like HP, Daimler, Agilent and ETAS/Bosch, and Fraunhofer IAO, where he initiated and coordinated a series of national and international research projects.

3 PROGRAM COMMITTEE

- Ulf Blanke, ETH Zuerich, Switzerland
- Daniel Burmeister, University of Lübeck, Germany
- Mirko Fetter University of Bamberg, Germany
- Mehmet Aydin Baytas, Koc University, Turkey
- Jo Vermeulen, University of Birmingham, UK
- Simo Hosio, University of Oulu, Finland
- Beat Signer, Vrije Universiteit Brussel Brussels, Belgium
- Peter Forbrig, University of Rostock, Germany
- Jan van den Bergh, Hasselt University, Belgium
- Heinrich Hussmann, Ludwig-Maximilian University Munich, Germany
- Anette Weisbecker, Fraunhofer IAO, Stuttgart, Germany
- Stefan Sauer, University of Paderborn, Germany
- Philippe Palanque, University of Toulouse, France
- Fabio Paterno, CNR-ISTI, Italy
- Gerhard Weber, Technische Universität Dresden, Germany
- Florian Daniel, University of Trento, Italy
- Gerrit Meixner, Heilbronn University, Germany
- Philippe Palanque, IRIT Toulouse, France
- Thomas Springer, Technical University of Dresden, Germany
- Jürgen Ziegler, University Duisburg-Essen, Germany
- Birgit Bomsdorf, Hochschule Fulda, Germany
- Romina Kühn, Technische Universität Dresden, Germany
- Christine Keller, Technische Universität Dresden, Germany
- Martin Christof Kindsmüller, Brandenburg University of Applied Sciences, Germany

4 PROGRAM

1st Workshop on Large-scale and Model-Based Interactive Systems: Approaches and Challenges
Duisburg, Germany – June 23, 2015, 9:00–17:30

9:00 ***Welcome and Introductions***

9:15 ***Keynote***

- Johannes Schöning

  Navigation in Ambient Spaces

10:00 ***Paper Presentations***

- Daniel Burmeister, Bashar Altakrouri and Andreas Schrader

  Ambient Reflection: Towards self-explaining devices

10:30 ***Coffee Break***

11:00 ***Paper Presentations***

- Ronny Seiger, Florian Niebling, Mandy Korzetz, Tobias Nicolai and Thomas Schlegel

  A Framework for Rapid Prototyping of Multimodal Interaction Concepts

- Bashar Altakrouri and Andreas Schrader

  Challenging Documentation Practices for Interactions in Natural User Interfaces

- Jürgen Engel, Christian Märtin and Peter Forbrig

  A Concerted Model-driven and Pattern-based Framework for Developing User Interfaces of Interactive Ubiquitous Applications

12:30	***Lunch***
14:00	***Paper Presentations*** • Enes Yigitbas, Bastian Mohrmann and Stefan Sauer *Model-driven UI Development integrating HCI Patterns*
14:30	***Discussions and Visual Roadmapping I***
15:30	***Coffee Break***
16:00	***Discussions and Visual Roadmapping II***
17:00	***Conclusions and Outlook***

5 ACCEPTED PAPERS

Invited Keynote:

- ***Navigation in Ambient Spaces***
  Johannes Schöning

Accepted Papers:

- ***Ambient Reflection: Towards Self-explaining Devices***
  Daniel Burmeister, Bashar Altakrouri and Andreas Schrader
- ***A Framework for Rapid Prototyping of Multimodal Interaction Concepts***
  Ronny Seiger, Florian Niebling, Mandy Korzetz, Tobias Nicolai and Thomas Schlegel
- ***Challenging Documentation Practices for Interactions in Natural User Interfaces***
  Bashar Altakrouri and Andreas Schrader
- ***A Concerted Model-driven and Pattern-based Framework for Developing User Interfaces of Interactive Ubiquitous Applications***
  Jürgen Engel, Christian Märtin and Peter Forbrig
- ***Model-driven UI Development integrating HCI Patterns***
  Enes Yigitbas, Bastian Mohrmann and Stefan Sauer

Keynote: Navigation in Ambient Spaces

Johannes Schöning
Expertise Centre for Digital Media
Hasselt University – tUL – iMinds
johannes.schoening@uhasselt.be

ABSTRACT
More and more spaces and environments are now sensitive and responsive to the presence of their users and not longer in a research prototype stage. In my talk I will highlight several interaction challenges that arise for the users in these novel and complex so-called “smart” environments. In particular, I will focus on how people can be guided through these spaces with the help of novel interface technologies. While these space may offer hundreds of different services and experiences to their users, still this fundamental problem will remain as the users need to actively navigate in these environments. In the talk I will present different of our approaches to help people "find their ways" in these environments with novel interfaces. Personal wearable devices can play an important role to assist the users. One example highlighted in the talk, besides others [1], will be the StripeMap prototype [3]. Small wearable devices present new challenges for cartography as well as HCI. In cartography large display sizes have significant advantages. The StripeMaps is a system that adapts the mobile web design technique of linearization for displaying maps on smartwatches' small screens. Just as web designers simplify multiple column desktop websites into a single column for easier navigation on mobile devices, StripeMaps transforms any two-dimensional route map into a one-dimensional "stripe". A conducted user study shows show that this simplification allows StripeMaps to outperform both traditional mobile map interfaces and turn-by-turn directions for pedestrian navigation using smartwatches.

Author Keywords
Smartwatches; Cartography; Mobile Maps; Pedestrian Navigation

ACM Classification Keywords
H.5.2 [Information Interfaces and Presentation]: User Interfaces—*input devices and strategies, interaction styles*

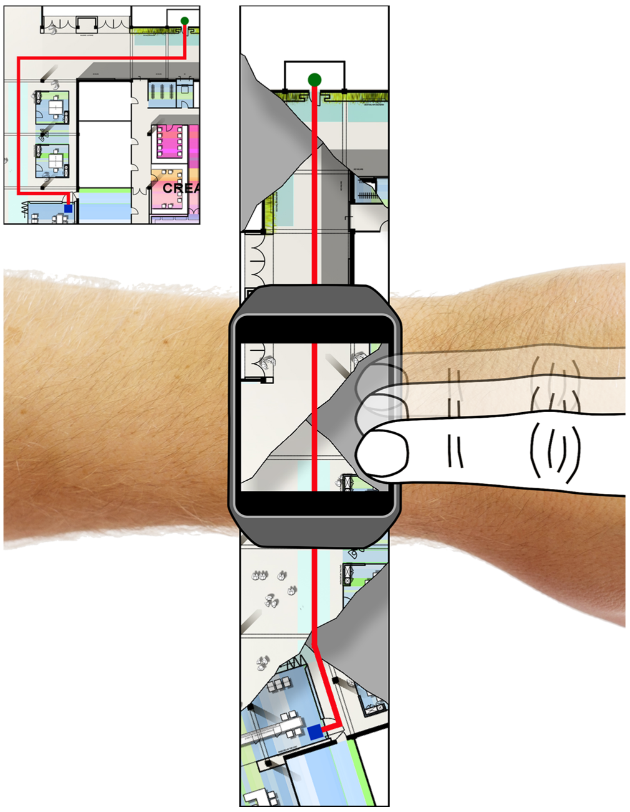


Figure 1: The StripeMaps concept. As screen space and interaction possibilities are limited on smartwatches, the StripeMaps application converts a 2D map to a 1D stripe. The original path on the 2D map is shown on the mini-map in the upper left corner. The cut (shown on the smartwatch) indicates the direction of the turn the user needs to make to navigate along the path.

ABOUT THE SPEAKER
Johannes Schöning a professor of computer science with a focus on HCI at Hasselt University. In addition, he is a visiting lecturer at UCL London within the Intel Collaborative Research Institute for Sustainable Cities. His main research interests are new methods and interfaces to navigate through spatial information. In general Johannes Schöning is developing new intelligent user interfaces that help people to solve daily tasks more effectively [2]. Before taking over the faculty position in Hasselt he worked in industry and at the German Research Centre for Artificial Intelligence DFKI in Saarbrücken from 2009 to 2011.

During that time Mr. Schöning received a PhD in computer science at the Saarland University in 2010 and a Master in Geoinformatics at the University of Münster at the Institute for Geoinformatics in 2007. His research and work was awarded with several prices and awards, such as two Google Research Awards, the ACM Eugene Lawler Award or the Nokia Innovator Award.

REFERENCES

1. Maaret Posti, Johannes Schöning & Jonna Häkkilä: Unexpected Journeys with the HOBBIT – The Design and Evaluation of an Asocial Hiking App. DIS 2014: Proceedings of the International Conference on Designing Interactive Systems, (2014)
2. Johannes Schöning: Interaction with geospatial data. it - Information Technology: Volume 57, Issue 1, (2015)
3. Dirk Wenig, Johannes Schöning, Brent Hecht, Rainer Malaka: StripeMaps: Improving Map-based Pedestrian Navigation for Smartwatches. MobileHCI 2015: Proceedings of the International Conference on Human-Computer Interaction with Mobile Devices and Service, 2015

Ambient Reflection: Towards self-explaining devices

Daniel Burmeister
University of Lübeck
Institute of Telematics
burmeister@itm.uni-luebeck.de

Bashar Altakrouri
University of Lübeck
Institute of Telematics
altakrouri@itm.uni-luebeck.de

Andreas Schrader
University of Lübeck
Institute of Telematics
schrader@itm.uni-luebeck.de

ABSTRACT
In the course of ubiquitous and pervasive computing a variety of smart devices are developed and entering our everyday life. These devices increasingly rely on novel interaction modalities from the field of Natural Interaction, such as gesture control. Common concepts to explain and illustrate devices' interaction possibilities can't be applied to these interaction techniques due to embedding of devices and as a consequence disappearing interfaces as well as distribution of functionalities among device ensembles in terms of IoT, AAL and Smart Home. These emerging and currently existing problems in accessing devices' interaction possibilities present users with new challenges. In addition, current possibilities for device documentation provide only a limited viable option to learn devices. Hence, a general documentation for interconnected devices and thus functionality can not be created manually. In order to counteract these problems we present an approach for in-situ generation of an ambient manual for interconnected smart devices.

Author Keywords
Ambient Computing; Human Computer Interaction Guidance

ACM Classification Keywords
H.5.2. User Interfaces: Training, help, and documentation

INTRODUCTION
As a result of the ongoing research and development in the areas of ubiquitous and pervasive computing the variety of heterogenous commercial devices with rich functionalities and novel interaction techniques arise. Particularly with regard to the fields of Ambient Assisted Living (AAL), Internet of Things (IoT) and Smart Home, users are increasingly faced with natural interaction. While bulk of HCI research strives to create interaction techniques that are easy to learn, natural, self-explaining, and novel, documentation of interaction techniques is generally an underestimated and ignored issue or simply considered luxury and unnecessary. Considering the progressive complexity in ambient scenarios containing heterogenous devices and interaction techniques, this results into an increasing gap between the users' ability to learn and remember these techniques and the provided functionality.

Currently, documentation of smart devices' interaction techniques in ambient space scenarios is usually spatial distributed and highly eco-centric, if accessible at all. Concerning the ongoing interconnection of devices and interaction behavior in terms of IoT, such documentation can't be realized manually. In total, the variety of current Smart Objects challenge users in accessing and operating. It is reasonably assumed, that these interaction challenges will increase significantly caused by complexity and unpredictable interconnections in ambient spaces.

SMART AMBIENT SPACES
Ambient spaces are manifested by an expanding world of interconnected Smart Objects full of rich interaction capabilities driven by ubiquitous and pervasive technologies. Research and industrial development in this area have resulted into vast increase in the number of smart commodity devices and objects (thereafter, called Smart Objects) seamlessly interweaving in a wide range of inhabited environments (e.g., households). A recent study conducted by BITKOM (Federal Association for Information Technology, Telecommunications and New Media) revealed that every household in Germany owns at least 50 electrical devices with an increasing tendency towards more devices [4] and half of all household devices are expected to be connected as part of a network by 2018 as reported by RWE Effizienz GmbH [25].

While users are currently familiar with handling normal physical objects and with interacting with simple and often limited number of electrical devices [25], the variety and diversity of functions and handlings of Smart Objects pose new challenges, especially to enable and familiarize users with interaction possibilities in ambient spaces [5, 28].

The increasing number of devices as well as the increasing diversity of offered functions imposes serious learning issues for the user according to Poppe et al. [20]. In one of his articles, Norman argued that this may easily lead to long-term usability obstacles and inflate problematic and irrational use of devices [16]. For instance, in case of a time change, different clocks in households offer inconsistent ways and interaction modalities for modifying the time. Hence, even this simple operation normally challenges the user [11]. Such challenges easily evolve with more emphasis on the required implicit knowledge of users and the lack of adequate documentation [30]. Sometimes devices can't even be controlled without the use of additional material [27].

Workshop on Large-scale and model-based Interactive Systems: Approaches and Challenges, June 23 2015, Duisburg, Germany.

The interaction challenges and difficulties with current and future smart devices and artifacts were also the subject of Norman's book, titled Living with Complexity [17]. In this book, Norman drew a clear distinction between *complexity* and *complication*. While complexity refers to the form of presentation of possible interaction states and transitions, complication donates the psychological state of a person who tries to learn an interaction with an object. Hence, complex objects and artifacts are not necessarily complicated to interact with. Complication barriers can appear due to different reasons including changing the environment and simultaneously changing artifacts. We believe that ambient spaces may result into various complication barriers due to three inherited characteristics reported by Pruvost et al. [23], namely, the heterogeneity and distributivity (containing a variety of devices with various capabilities); dynamic media mobility (interaction capabilities are highly dynamic as interaction devices may join and leave the ambient space at any time), and user mobility (challenging users to attend to interaction needs). This leads, very commonly, to missing the natural mapping of offered functionality and adequate interaction modalities [15, p. 12], as well as to hindering the user from building the correct mental model of the system.

OPPORTUNITIES AND CHALLENGES OF NATURAL USER INTERFACES

Recent advancements in HCI research have revealed new and novel interaction techniques to operate and control devices in ambient spaces by using Natural User Interfaces (NUIs) as in multi-touch gestures, motion-gestures, gaze-interactions, etc. [8]. In literature, different definitions of interactions with NUIs were elaborated and most of them refer to the user's natural abilities, practices, and activities to control interactive systems. Many of those interactions are mostly caused and characterized by motion and movement activities, ranging from pointing, clicking, grasping, walking, balancing, dancing, etc. as discussed in [1].

In the last 10 years, NUIs, using touch and motion enabled technologies, found their way commercially and became widely accessible to the end user. Moreover, users are becoming more acquainted with using different body parts to interact with applications such as gaming (e.g., motion-controlled active play by Microsoft Kinect or the Wii system), data browsing, navigation scenarios (e.g., tilting for scrolling photos as in iOS and Android devices), and many more. This has encouraged the HCI community to continuously expand towards the NUI paradigm and currently various new calls have been arisen to explore new potential in designing for the whole body in motion [7, 9]. Despite the efforts towards intuitive and simple interfaces, the NUI paradigm is challenged by an expanding user population and diversity with respect to age and physical abilities, as discussed in [1]. On the one hand, the naturalness of NUI does not imply the simplicity to recall and use interaction techniques [18]. On the other hand, utilizing the human body and its parts for interaction comes with its own set of complexities. Simple commands, like "raise your arm", may have very different interpretations. Different aspects are important to consider for correctly executing such a simple command, for instance movement direction, involved body parts, timing information, etc.

GUIDANCE IN AMBIENT SPACES

In order to correctly use simple or complex technologies, the availability and accessibility of relevant information are essential for the user. Therefore, Norman [14] coined the term *affordances* in respect of objects' self-revealing interaction possibilities to easily enable users interacting with them. The same applies for the interaction in ambient spaces, however the current concept of affordances does not apply to the ongoing embedding of devices and accordingly their interaction possibilities [26, 22]. The dynamic nature of ambient spaces imposes different learning and affordance challenges on users. In this regard, relaying solely on visual appearance and affordances of a smart object to explain its logic and function are not enough [26]. Hence, adequate documentation and presentation of interaction possibilities and the utility of an object are essential part for learning ambient spaces, which aim at correct usage of devices and optimized user mental models.

In ambient spaces, documentation is not only vital for the use of objects but also for the design process itself and for a successful share and exchange of components and knowledge. Although different device manufacturers pay attention to the consistency of interaction patterns and product descriptions, there are currently no consistent and unified standards for describing smart objects and their offered interaction possibilities in ambient spaces. Based on this, users have to repeatedly remember how to interact with such devices [24]. This recurring state of knowledge between beginner and expert in interacting with a device is called perpetual intermediate [6, p. 42].

Our previous work on reviewing existing documentation-related tools for NUIs revealed four general observations or shortcomings, namely the lack of widely adopted tools by NUI designers, the absence of dedicated NUI documentation tools, the lack of end-user support, and the lack of support and considerations of body movements and postures as part of the interaction descriptions (if at all found). Furthermore, the review revealed that there is a lack of formalized languages and notations of generic motion documentation [1, 3]. For the previously mentioned reasons and potentially more, people turn to rely on other learning approaches and methods. Trial-and-Error is a very common practice to unveil adequate system interactions used by users. However, it is not necessarily the most effective approach in many cases. This was the subject of many research studies in the area of safety and critical environments. A study has revealed that 70% of surgeons and 50% of nurses demonstrated problems dealing with medical devices in operating theaters [12], where 40% of the respondents indicated that the ignorance of adequate operation guidelines of medical devices have resulted into repeatedly occurring hazards. In a previous study [1], the majority of reported respondents of a questionnaire (more than 90%) rely on try and error to learn interactive techniques of personal interactive devices (e.g., smart phones, interactive TVs, handhelds, and game consoles). This can be due to the limited range and simplicity of interaction features currently available in the users' commodity devices (e.g., swipe, shake, and pan). Nonetheless, there is a strong evidence that learning and memorizing interaction techniques will become more complex due to the vast growth of multi-touch- and motion-based interactions in terms of, but not limited to, the number of interactions proposed, the increasing complexity of interaction techniques, expanding diversity of interaction types, involved body parts, involved actions, and runtime ensembles of interaction techniques [7, 9, 13, 2]. This clearly advocates the need for reference documentation of interaction techniques as a necessity and an aid tool for users [19]. In fact, interactivity in ambient spaces is becoming even increasingly dynamic (interaction environments are becoming increasingly heterogeneous and dynamic and no longer static and closed [23]), adaptive (required for sustainable utility and usability), and multi-modal. Hence, interactive ambient spaces are created in an ad-hoc manner, where multiple interaction techniques grouped together to adapt the available interaction resources and possibilities to the user's physical context and abilities. This shift towards an evolving world of interactivity (smart spaces, user mobility, anthropomorphic abilities and disabilities, preferences, etc.) requires new dissemination, deployment, and adaptation mechanisms for NUI. For these reasons, documentation for training, demonstration, and reference purposes plays a major role to set the limits and boundaries for NUI deployment and adoption in interactive ambient spaces.

A FRAMEWORK FOR AMBIENT REFLECTION

In order to offer a possibility to compensate the previously mentioned emerging problems in interaction and documentation, we strive for developing a three-divided framework for Ambient Reflection as an integral component of reflective systems self-x properties [21, p. 322 et seq.]. By providing this framework as a feasible solution, we foster the multimodal self-description of (interconnected) devices regarding interaction possibilities. In total, our envisioned framework consists of three building blocks, namely an *Ambient Reflective Documentation Language*, *Documentation Fusion* and *Presentation Oriented Publishing*. In the following paragraphs, these components are described in detail.

Ambient Reflective Documentation Language

Current possibilities for technical documentation are limited to unstandardized media entities, i.e., each device is described in different modalities using different types of media in various formats. Hence, caused by this diversity an automated processing is not possible. In order to achieve this property a unified extensible documentation language for ambient spaces should be provided, covering a structured description about devices specification and interaction possibilities on a high granularity (further referred as micro-level). Moreover, a documentations content should be decoupled from its presentation in order to achieve more flexibility for further processing, which already has successfully been done (e.g. by [29]). This approach of presentation-neutral describing of Smart Objects may guarantee distributivity, extensibility and further presentation oriented processing.

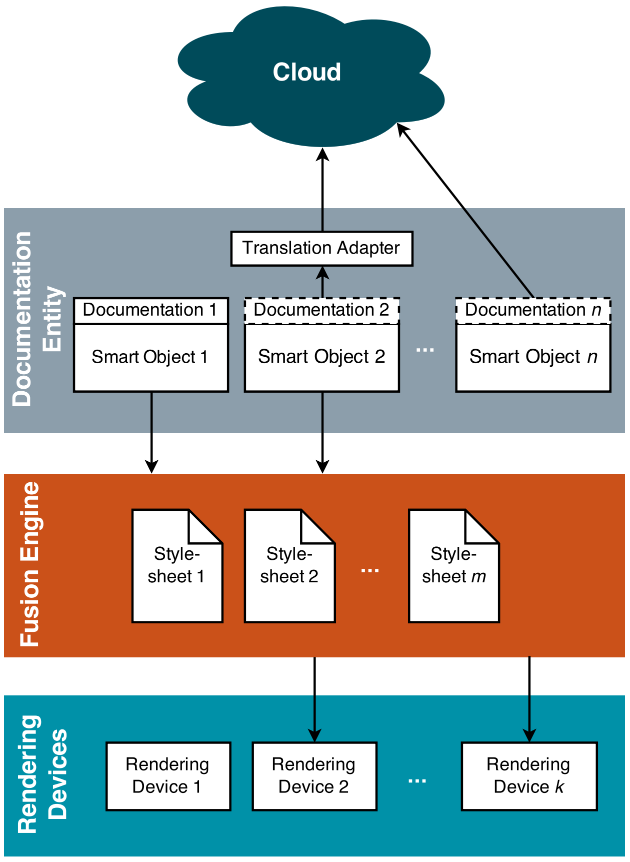


Figure 1. Ambient Reflection Framework

Documentation Fusion

Given the assumption, that relevant devices in an ambient space scenario were described by an ambient reflective documentation, the documentation fusion will take place. In the following, a device with access to its remote located or attached documentation is called documentation entity. Including the current context and environmental state, the fusion step performs in-situ processing and merging of distributed documentation entities. As result it generates a presentation-neutral adaptive ambient space manual for interconnected ensembles of Smart Objects in ubiquitous and pervasive environments, considering just involved devices and interaction possibilities.

Distributing the generated material to a dedicated coordinating engine will enable a guidance system to offer further instructions regarding interaction of device ensembles at the time the user needs or asks for support. Nonetheless, the fusion step might be skipped to provide even single device interaction guidance as well.

Presentation Oriented Publishing

Using concepts of presentation oriented publishing for markup languages [10] adds an additional abstraction layer between the generated manual and the final presentation of instructions to the user. The formerly mentioned decoupling properties of presentation-neutral documentation facilitates the documentation language translation into other renderable languages. Furthermore, the inclusion of a standardized style description to the translated language concerning presentational aspects will offer a possible solution for adaptivity in presentation. Finally, the fused documentation should be deployed to appropriate rendering devices. E.g. a fused documentation might be translated into the Scalable Vector Graphics format using different color sets for color-blind users and visualized by an internet browser's rendering engine.

EXEMPLARY SCENARIO

In order to further illustrate the frameworks' working-process the following scenario is assumed (see figure 1): A user resides in an environment containing of *n* Smart Objects. Since a device ensemble, consisting of Smart Object 1 and 2, was built, the user needs guidance in usage. Object 1 is already described by using the Ambient Reflective Documentation Language, where Object 2 still needs to be described. Therefore, an adapter is used to translate existing documentation, written by the manufacturer, into the documentation language. It is likewise conceivable, that an ambient reflective documentation is remote located, whereas the device just provides the destination (as done by Smart Object *n*). The Fusion Engine fetches and merges the documentation of the involved Objects as well as applying a stylesheet based on the users preferences. Finally, the generated manual is deployed to rendering devices 2 and *k* and by association delivered to the user. It should be noted, that a rendering device might be equal to a Smart Object in the environment and thus might also be documented. Hence, a set of documented rendering devices might form a device ensemble with other Smart Objects and thus are by definition documentation entities.

CONCLUSION

Considering the current development in the areas of IoT, AAL and Smart Home face users with new challenges in terms of Human Computer Interaction. Devices' and their functionality are progressively interconnected, embedded and new interaction techniques within the scope of Natural Interaction arise. As a result of the ongoing disappearing of user interfaces as well as the emerging usage of gesture control, the current concept of affordances may not apply to current developments. Existing difficulties in HCI will increase in the areas of ubiquitous and pervasive computing, caused by the environments high complexity, heterogeneity and dynamic. Beyond this, present technical documentation does not follow a common pattern or is adapted to the user's needs. As a possible solution to tackle these problems, we presented the approach of a conceptual ambient reflection framework, consisting of three major components: An Ambient Reflection Documentation Language for describing interaction possibilities of Smart Objects on a micro-level, documentation fusion using the description language to merge documentation entities of interconnected devices for generating an ambient manual tailored to the users' context and needs as well as the presentation oriented publishing for multi-modal rendering the manual in-situ. In total, we strongly believe to counteract emerging interaction problems in ambient space scenarios by further investigating this framework.

FUTURE WORK

Next, we take to carry out a study to identify different contexts and therefore needs of users with respect of documentation and guidance in interaction. Including these findings and further research regarding description languages, we will develop a unified Ambient Reflective Description Language for Smart Objects and apply it to a representative set of Smart devices, composed of different device categories. In addition, we try to determine a set of generic rules and processes in order to achieve a consistent manual generated by the fusion engine. Upon this, the development of interweaving style description and documentation and the delivery to rendering devices should enable Smart Objects to describe themselves. Finally, we plan to evaluate our framework by carrying out a scenario-based evaluation to determine the precision of the fusion itself as well as the usefulness of our provided guidance for the user.

REFERENCES

1. Altakrouri, B. *Ambient Assisted Living with Dynamic Interaction Ensembles*. PhD thesis, University of Lübeck, the Department of Computer Sciences/Engineering, published by Zentrale Hochschulbibliothek Lübeck, August 2014.
2. Altakrouri, B., and Schrader, A. Towards dynamic natural interaction ensembles. In *Fourth International Workshop on Physicality (Physicality 2012) co-located with British HCI 2012 conference*, A. D. Devina Ramduny-Ellis and S. Gill, Eds. (Birmingham, UK, 09 2012).
3. Altakrouri, B., and Schrader, A. Describing movements for motion gestures. In *1st International Workshop on Engineering Gestures for Multimodal Interfaces (EGMI 2014) at the sixth ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS14)* (Rome, Italy, June 2014).
4. BITKOM Bundesverband Informationswirtschaft, Telekommunikation und neue Medien e. V. Leitfaden zur Heimvernetzung Band 2, 2011.
5. Bongers, B. Interacting with the disappeared computer. In *Mobile HCI, Physical Interaction Workshop on Real World User Interfaces.* (Udine, Italy, 2003).
6. Cooper, A., Reimann, R., and Cronin, D. *About Face 3: The Essentials of Interaction Design*. Wiley, 2007.
7. England, D. Whole body interaction: An introduction. In *Whole Body Interaction*. Springer, 2011, 1–5.
8. Estrada-Martinez, P. E., and Garcia-Macias, J. A. Semantic interactions in the internet of things. *International Journal of Ad Hoc and Ubiquitous Computing 13*, 3 (2013), 167–175.
9. Fogtmann, M. H., Fritsch, J., and Kortbek, K. J. Kinesthetic interaction: revealing the bodily potential in interaction design. In *Proceedings of the 20th Australasian Conference on Computer-Human Interaction: Designing for Habitus and Habitat*, ACM (2008), 89–96.

10. Goldfarb, C. F., and Prescod, P. *The XML Handbook*. Prentice-Hall, Upper Saddle River, New Jersey, 1998.

11. Leitner, G., Hitz, M., Fercher, A. J., and Brown, J. N. A. Aspekte der Human Computer Interaction im Smart Home. *HMD - Praxis Wirtschaftsinform. 294* (2013).

12. Matern, U., Koneczny, S., Scherrer, M., and Gerlings, T. Arbeitsbedingungen und Sicherheit am Arbeitsplatz OP. *Deutsches Ärzteblatt 103*, 47 (November 2006), A 3187 – 3192.

13. Navarre, D., Palanque, P., Ladry, J.-F., and Barboni, E. Icos: A model-based user interface description technique dedicated to interactive systems addressing usability, reliability and scalability. *ACM Transactions on Computer-Human Interaction (TOCHI) 16*, 4 (2009), 18.

14. Norman, D. A. Affordance, conventions, and design. *interactions 6*, 3 (1999), 38–43.

15. Norman, D. A. *The Design of Everyday Things*. Basic books, 2002.

16. Norman, D. A. Simplicity is not the answer. *interactions 15*, 5 (2008), 45 – 46.

17. Norman, D. A. *Living with complexity*. MIT Press, 2010.

18. Norman, D. A., and Nielsen, J. Gestural interfaces: A step backward in usability. *interactions 17*, 5 (Sept. 2010), 46–49.

19. Pham, D. T., Dimov, S., and Setchi, R. Intelligent product manuals. *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering 213*, 1 (1999), 65–76.

20. Poppe, R., Rienks, R., and van Dijk, B. Evaluating the future of hci: challenges for the evaluation of emerging applications. In *Artifical Intelligence for Human Computing*. Springer, 2007, 234–250.

21. Poslad, S. *Ubiquitous Computing: Smart Devices, Environments and Interactions*. Wiley, 2009.

22. Preim, B., and Dachselt, R. Die Interaktion mit Alltagsgeräten. In *Interaktive Systeme*. Springer, 2010, 135–161.

23. Pruvost, G., Heinroth, T., Bellik, Y., and Minker, W. User interaction adaptation within ambient environments. In *Next Generation Intelligent Environments*. Springer, 2011, 153–194.

24. Quesenbery, W. Balancing the 5Es of Usability. *Cutter IT Journal 17*, 2 (2004), 4–11.

25. RWE Effizienz GmbH. Wendepunkte der Energiewirtschaft. Online, 2014. `http://www.rwe.com/app/Pressecenter/Download.aspx?pmid=4012118&datei=1`.

26. Streitz, N., Prante, T., Röcker, C., Van Alphen, D., Stenzel, R., Magerkurth, C., Lahlou, S., Nosulenko, V., Jegou, F., Sonder, F., et al. Smart artefacts as affordances for awareness in distributed teams. In *The Disappearing Computer, Interaction Design, System Infrastructures and Applications for Smart Environments*, Springer (2007), 3 – 29.

27. Thimbleby, H., and Addison, M. Intelligent adaptive assistance and its automatic generation. *Interacting with Computers 8*, 1 (1996), 51–68.

28. van der Vlist, B. J., Niezen, G., Hu, J., and Feijs, L. M. Semantic connections: Exploring and manipulating connections in smart spaces. In *Proceedings of the 15th IEEE Symposium on Computers and Communications, ISCC 2010*, IEEE (June 2010), 1–4.

29. Walsh, N. Docbook 5: The definitive guide. Online, 2010. `http://docbook.org/tdg5/`.

30. Zandanel, A. Users and households appliances: Design suggestions for a better, sustainable interaction. In *Proceedings of the 9th ACM SIGCHI Italian Chapter International Conference on Computer-Human Interaction: Facing Complexity*, CHItaly, ACM (New York, NY, USA, 2011), 96–100.

A Framework for Rapid Prototyping of Multimodal Interaction Concepts

Ronny Seiger
Technische Universität Dresden
Dresden, Germany
ronny.seiger@tu-dresden.de

Florian Niebling
Technische Universität Dresden
Dresden, Germany
florian.niebling@tu-dresden.de

Mandy Korzetz
Technische Universität Dresden
Dresden, Germany
mandy.korzetz@tu-dresden.de

Tobias Nicolai
Technische Universität Dresden
Dresden, Germany
tobias.nicolai@mailbox.tu-dresden.de

Thomas Schlegel
Technische Universität Dresden
Dresden, Germany
thomas.schlegel@tu-dresden.de

ABSTRACT
Ubiquitous systems provide users with various possibilities of interacting with applications and components using different modalities and devices. To offer the most appropriate mode of interaction in a given context, various types of sensors are combined to create interactive applications. Thus, the need for integrated development and evaluation of suitable interaction concepts for ubiquitous systems increases. Creation of prototypes for interactions is a complex and time consuming part of iterative software engineering processes, currently not well supported by tools as prototypes are considered to be short-living software artifacts. In this paper, we introduce the framework *Connect* that enables rapid prototyping of interaction concepts with a focus on software engineering aspects. The framework allows the definition and modification of event-to-action mappings for arbitrary interaction devices and applications. By applying *Connect*, model-based prototypes of multimodal interaction concepts involving multiple devices can be created, evaluated and refined during the entire engineering process.

ACM Classification Keywords
H.5.2 User Interfaces: User-centered design

Author Keywords
interaction framework; interaction concepts; multimodal interaction; rapid prototyping; software engineering

INTRODUCTION
Ubiquitous Systems as coined by Weiser [22] describe user-centered systems at the intersection of mobile and pervasive computing combined with ambient intelligence. In addition to the high degree of embeddedness of pervasive systems, ubiquitous systems are characterized by a high level of mobility, often consisting of a large number of heterogeneous and possibly resource-limited devices, which are loosely-coupled into dynamic network infrastructures. The emergence of smart spaces (smart homes, smart factories, smart offices) shows the increasing importance and spreading of ubiquitous systems throughout different areas of everyday life.

As user interaction with devices that disappear into the background often cannot be realized using traditional metaphors, new ways of interaction have to be explored during the software engineering process for creating ubiquitous system components. The development and improvement of multimodal interaction concepts (i. e., interactions using one or more input modalities [21]) is thereby not limited to initial prototyping, but equally important during implementation, testing and evaluation stages.

Existing tools for interaction design and rapid prototyping of ubiquitous user interaction can be successfully employed during the initial prototyping phases. In later development phases as well as in iterative engineering processes such as user-centered design (UCD), their applicability is often restricted due to their limited ability to automatically propagate changes in prototypes to subsequent stages of development. The mostly informal descriptions and implementations of interaction concepts and interactive applications limit their extensibility with respect to new interaction devices and modalities. The lack of models and formalism also prevents prototypes for interactions from being used and matured in later stages of the development process, which is why prototypes are usually considered to be short-living software artifacts.

In this paper, we propose a model-driven framework for prototyping of interaction concepts that can be applied throughout the different phases of iterative software engineering processes. The focus of the introduced *Connect* framework for interaction design is placed on software engineering aspects

Workshop on Large-scale and model-based Interactive Systems: Approaches and Challenges, June 23 2015, Duisburg, Germany.

and models. It enables the rapid development of prototypes and enhancement at runtime during expert evaluation and user studies. Extensibility concerning new types of interaction devices as well as interactive components is easily achieved by inheritance in the object-oriented framework architecture. As a result of a high-level, model-based design of interaction concepts, modifications to the interactions–even of very early prototypes–can be reused and advanced from beginning to end of the development cycle. The framework supports individualizations concerning different groups of users as well as distinct scenarios by customizing interaction concepts using specialization and configuration. We introduce a tool based on the *Connect* framework that facilitates the creation and customization of interaction concepts at runtime even by non-programmers. The framework is demonstrated developing an interaction prototype within a Smart Home–a ubiquitous environment consisting of various devices for multimodal interactions with physical and virtual entities.

RELATED WORK

Prototypes are useful tools for assessing design decisions and concepts in early but also in advanced stages of the software engineering process. Especially in iterative design processes for creating usable systems, future users have to be involved continuously to provide feedback and to improve concepts and software artifacts [9]. According to Weiser, one of the essential research and development methods for interactive ubiquitous systems is the creation of prototypes [23]. The rapid prototyping technique aims at creating prototypes in a time and resource efficient way to mature artifacts in agile software engineering processes [15]. The focus of our work is on providing a framework for the rapid development and evaluation of multimodal interactions for ubiquitous systems. Especially within the UCD process, prototypes are needed to evaluate design ideas and improve the usability of interactive systems [17]. Complex interaction scenarios involving multimodal interactions require the use of technically mature prototypes to improve the usability of the system or application [14].

The basis for our prototyping framework is the *OpenInterface* platform developed by Lawson et al. [13]. The platform allows the integration of arbitrary heterogeneous components as interaction devices and applications. OpenInterface provides a lightweight framework and runtime environment leveraging the prototyping of multimodal interaction concepts using high-level data fusion and pipeline concepts to connect interaction devices with interactive applications. In contrast to other frameworks for prototyping of interactions [7, 4], OpenInterface is based on a platform and technology independent description of a component's functionality and interface. Our framework adapts these concepts with a stronger focus on the underlying models and component descriptions in order to facilitate the extension, runtime adaptation and reuse of components and interactions during the iterative stages of UCD. Other existing interaction and prototyping frameworks (e. g., *CrossWeaver* [20] and *STARS* [16]) are realized in a more informal way for specific scenarios and therefore lack extensibility of software components, interaction devices and modalities as well as reusability. The OpenInterface workbench [13] provides developers with a comprehensive toolset for configuring components and interactions. Our aim is to provide an easy to use tool also enabling non-programmers to rapidly create interaction prototypes.

In [10] Hoste et al. describe the *Mudra* framework for fusing events from multiple interaction devices in order to leverage multimodal interactions. Mudra provides means for processing low-level sensor events and for inferring high-level semantic interaction events based on a formal declarative language. The framework is primarily focused on combining multiple interaction devices to enable advanced forms of interaction input. It can be used complementary to the *Connect* framework as part of defining and integrating low-level and high-level sensor components. However, the application of the formalism and semantic models that Mudra is based on increases the effort and complexity for rapidly prototyping interaction concepts and introduces a number of additional components to the lightweight *Connect* framework.

In addition to the work of Dey et al. [6], one of the first conceptual frameworks for the rapid prototyping of context-aware applications predominant in ubiquitous systems, the *iStuff* toolkit [3] and its mobile extension by Ballagas et al. [2] represent further related research our *Connect* framework is based on. These toolkits offer a system for connecting interactive physical devices and interaction sensors with digital applications and software components. The iStuff toolkit suite supports multiple users, devices and applications as well as interaction modalities. However, due to the limited software models applied within these tools, the set of supported interaction devices is rather static. A model-based approach for dynamically creating multimodal user interfaces composed of several components is described by Feuerstack and Pizzolato [8] as part of their *MINT* framework. Mappings between user interface elements, user actions and actions to be executed can be formally defined with help of this framework and used for dynamically generating interactive applications. Both the iStuff and MINT framework are intended to be used in the design and development process of user interfaces whereas our focus lies on the prototyping and evaluation of interaction concepts (i. e., event-to-action mappings) in different stages of the UCD process. However, in order to prototype and develop interactive applications including interaction concepts and user interfaces, the iStuff and *Connect* framework can be used complementary.

The *ProtUbique* framework by Keller et al. facilitates the rapid prototyping of interactive ubiquitous systems [11]. It supports an extensible set of interaction channels transmitting low-level interaction events from heterogeneous devices and sensors. These interaction events are unified by the framework and accessible programmatically in order to prototype interactive applications. As ProtUbique offers interfaces to access its interaction channels, it is possible to directly combine both the ProtUbique and *Connect* framework. Interaction channels are integrated into *Connect* in the form of sensors supplying interaction events. Connect can then be used to map these events to actions that will be executed by actuators or applications.

With the emergence of ubiquitous systems, users play a central role in the software development process. Prototypes are well suited for involving users in the design process and for improving concepts and software artifacts based on user feedback. Various frameworks for the prototyping of interactive applications including user interfaces and interaction concepts exist. These frameworks are often focused on the use of prototyping techniques in early development stages and limited in the set of supported software components and interaction modalities. However, agile and iterative software engineering processes are required for developing interactive ubiquitous systems. Therefore, we propose a model-driven framework for the rapid prototyping of multimodal interaction concepts. By applying models for the definition of interactive components and their interrelations, extensibility and reusability of interaction concepts and interactive prototypes in multiple design stages is facilitated. In that way, the usability and user experience of applications and systems for ubiquitous environments can be increased.

INTERACTION FRAMEWORK

Structure

We designed the *Connect* framework from a software engineering point of view using abstract models and their building blocks as a starting point. The framework adheres to a basic class structure consisting of multiple types of components, which are interconnected with each other. Fig. 1 shows the class diagram using UML notation. A *Component* is a software entity having a defined interface and known behavior [13]. In analogy to control systems linking physical sensors with actuators, we distinguish between *SensorComponents* representing entities that are able to produce interaction events and *ActuatorComponents* able to consume interaction events and trigger subsequent actions. *ComplexComponents* combine these capabilities. In addition, specializations of complex components are used to enable the logical and temporal combination of sensor and actuator components. Ports describe the components' interfaces in order to define interactions and connections between multiple components. *EventPorts* define the types of events a sensor component is able to produce and *ActionPorts* represent the types of actions an actuator component is able to perform. The activation of an event port leads to the activation of the action ports the event port is connected to. A central Manager class handles the instantiation of components and maintains a list of all active component instances, which are accessible from within the scope of the framework.

Sensor Components

Sensor components represent devices and applications that are able to detect interactions and produce corresponding interaction events. The *SensorComponent* is an interface sensors have to implement, e. g., by an adapter connecting the sensor device via its API to the *Connect* framework. An *EventPort* is a wrapper for every type of event the sensor component is able to trigger. The sensor component maintains a list of all its events and creates corresponding event ports. By implementing the sensor component interface, new types of

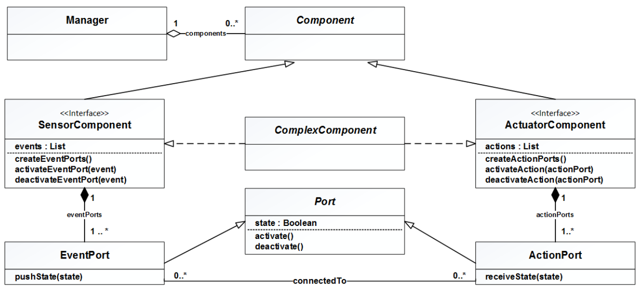


Figure 1. Class diagram of the *Connect* framework

interaction devices and arbitrary event sources can be integrated into the framework. In order to integrate new types and corresponding instances of sensor components into the framework, an adapter for receiving the sensors' events has to be implemented. On receiving an event from the sensor, the state of the corresponding event port is updated, i. e., the port is activated or deactivated. An event port can be connected to one or more action ports of one or more actuator components. The event port's activation leads to the activation of the connected action ports. As arbitrary event sources are supported, interactive devices independent of modality and number can be combined to be used for multimodal interactions, i. e., using one or more input modalities. Currently, only binary states for events (active/inactive) without additional data payload are supported by the *Connect* framework.

An example of a locally integrated sensor component is the computer's keyboard. The event ports correspond to the set of the individual keys. A smartphone device sending touch events via a dedicated app to an instance of the *Connect* runtime is an example of a remotely integrated sensor component. The set of touch events provided by the smartphone and supported by the app represent the event ports.

Actuator Components

Actuator components represent devices and applications that are able to actively perform and execute actions. Analogous to a sensor component, the *ActuatorComponent* is an interface actuators have to implement in order to connect the actuator to the *Connect* framework. An *ActionPort* is a wrapper for an action or method the actuator component is able to execute. New types of actuator components can be integrated into the framework as implementations of the interface *ActuatorComponent*. Adapters for calling the actuator components' particular operations from inside the framework have to be implemented for every type of actuator. An action port can be connected to one or more event ports of one or more sensor components. Upon receiving an activation from an event port connected to an action port, the actuator component activates the action port and executes the corresponding method. Arbitrary local and remote devices and applications can be integrated into the framework as actuator components. Currently, we support the activation of methods without the processing of input or output parameters.

An example of an actuator component is a service robot whose movement functionality is provided in the form of directed movement actions (e. g., forward, backward, left, right). For each direction there is a corresponding action port.

Complex Components

Complex components represent devices and applications that combine sensor and actuator functionalities. These entities can contain multiple event and action ports, i. e., they are able to produce events for actuator components and receive events from sensor components. The *ComplexComponent* class is viewed as an abstract class that has to implement both the *SensorComponent* and the *ActuatorComponent* interfaces in order to be integrated into the *Connect* framework.

An example of a complex component is a smartphone sending touch interaction events and providing executable operations (e. g., for taking pictures or switching on the its light).

Logical and Temporal Components

Logical Components are viewed as specializations of complex components. They are used for creating logical connections (AND, OR, NOT, XOR, etc.) between multiple event ports of one or more sensor components. The logical component's action ports are used as input ports for events from the sensor components and its event ports are used as output ports producing events for the activation of subsequent actuator components. By cascading these logical components, complex logical circuits for sensor events triggering actions of actuator components can be created. In addition, we integrate flip-flop and trigger components for saving of and switching between states. That way, it is possible to define more complex interaction concepts involving multiple interaction devices in advanced stages of the engineering process and also to introduce modes of interaction (i. e., state-dependent behavior of the interactive prototypes).

Besides logical components, *Temporal Components* for describing temporal dependencies between sensor events are supported as extensions of complex components. That way we are able to define the activation of higher level events, e. g., after a defined number of repeating sensor events or after the appearance of an event within a defined time frame. The functions and algorithms–including additional attributes–that are executed when the logical or temporal component is activated have to be provided for each new type of complex component. As new types of components are introduced into the framework's underlying class model by inheritance, only the base classes' methods have to be overwritten to use instances of these new components.

Dynamic Components

Thus far we are able to extend the set of sensor, actuator and complex components by introducing implementations and specializations of the appropriate classes into the model at design time. In order to add new types of components at runtime, we extend the framework by the concept of *Dynamic Components*. These components are created by *Connect*'s runtime based on a formal model of a component's functionality and ports. Currently, we support the use of a WSDL (Web Services Description Language [5]) document describing the available operations of a service-based actuator component. The WSDL format provides a suitable formalization of an actuator's callable methods and their parameters, which can be parsed in order to automatically create an actuator component implementing the *ActuatorComponent* interface and the corresponding action ports.

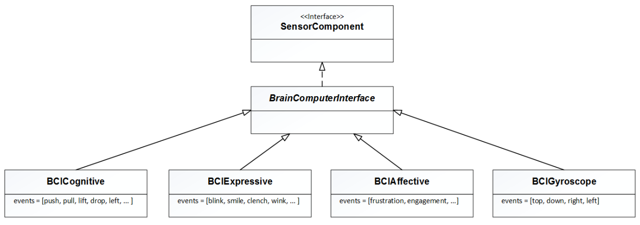

<<Interface>>
SensorComponent
BrainComputerInterface
BCICognitive
events = [push, pull, lift, drop, left, ...]
BCIExpressive
events = [blink, smile, clench, wink, ...]
BCIAffective
events = [frustration, engagement, ...]
BCIGyroscope
events = [top, down, right, left]

Figure 2. Extensions of the *SensorComponent* to support BCI input modes

PROTOTYPING MULTIMODAL INTERACTION CONCEPTS

Exemplary Sensors

Brain Computer Interface

In order to show the framework's capability of supporting multimodal interactions and its applicability within the Smart Home scenario, we extended the core sensor component by the Emotiv EPOC[1] EEG brain computer interface (BCI) acting as a source of interaction events [19]. The BCI used in our setting provides interaction modes enabling the recognition of thoughts (*Cognitive*), facial expressions (*Expressive*), emotions (*Affective*) and head movement (*Gyroscope*). Each of these modes is introduced as a subclass of the abstract *BrainComputerInterface* class, which implements the *SensorComponent* interface (see Fig. 2). Event ports are created for every possible type of sensor event produced by the BCI in each mode (e. g., for blink, wink left, look right, smile, and laugh in the Expressive mode). Upon instantiation of an object of one of the interaction mode classes, a listener for event ports corresponding to the sensor events is initialized.

Tablet

The second exemplary sensor component from the Smart Home domain that we integrated into our test setting is an Android-based tablet device. A dedicated app sends interaction events regarding the pressing of specific buttons and events detected by the tablet's gyroscope sensor to an instance of the *Connect* framework. In order to support this event source, we introduce the abstract *Tablet* class implementing the *SensorComponent* interface. From that class, the *Button* and *Gyroscope* modes are derived as subclasses (see Fig. 3). Event ports representing the particular buttons and gyroscope movement directions (i. .e., forward, backward, left, right) enable the detection of the corresponding interaction events and connection to other components.

Exemplary Actuators

Service Robot

A TurtleBot 2[2] service robot plays the role of an actuator in the context of our Smart Home scenario [19]. We abstracted its movement functionality into two operational modes extending the abstract *ServiceRobot* class: *Manual Movement*

[1] https://emotiv.com/epoc.php
[2] http://www.turtlebot.com/

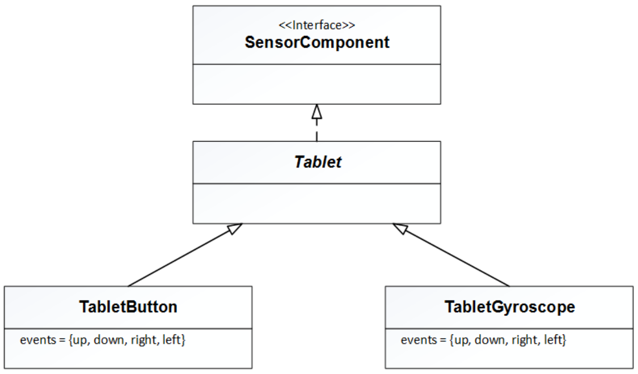

<<Interface>>
SensorComponent
Tablet
TabletButton
events = {up, down, right, left}
TabletGyroscope
events = {up, down, right, left}

Figure 3. Extensions of the *SensorComponent* to support tablet input modes

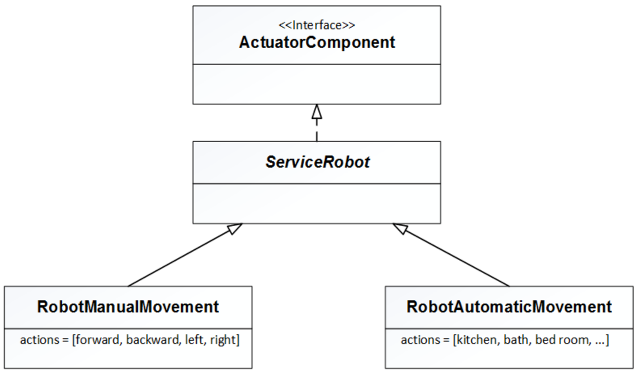


Figure 4. Extension of the *ActuatorComponent* to support a service robot actuator

and *Automatic Movement* (see Fig. 4). The manual movement mode supports the fine-grained control of the robot platform by direct movement commands (i. e., forwards, backwards, to the left, to the right). Using the automatic movement mode, the robot can be send to specific locations in a room or building. In automatic mode, driving, path planning and obstacle avoidance are handled by the robot itself. The action ports for these two actuator component modes correspond to the available movement directions (manual mode) and to the specific target locations (automatic mode).

Service-based Light Switch

The capability of dynamically adding new components at runtime is an important feature of the *Connect* framework. As it supports the automated generation of an actuator component based on a WSDL document, we implemented a web service for the remote control of a light switch providing a *switch on* and a *switch off* operation. Upon parsing of the WSDL file and creation of action ports for both operations, the *Connect* runtime acts as a client sending requests to the web service.

Security Component

In order to prevent incorrect behavior and actions caused by imprecise interaction devices and unintended user interactions at runtime, a *Security Component* is introduced into the framework as an implementation of the actuator component. This component provides an operation for deactivating all event and action ports and thereby disabling the current

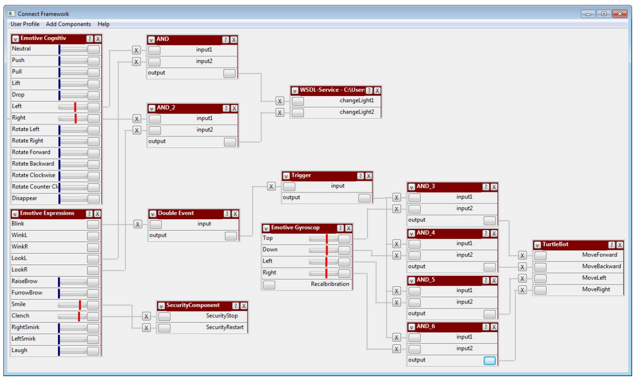


Figure 5. Complex network of input and output components forming an interactive prototype

interactions and listeners for new events. The security component's second operation resets all ports to the inactive state and re-enables the event listeners to continue with the interaction. Both operations can be connected to the event ports of an arbitrary–preferably reliable–sensor component.

Prototyping Tool

We implemented a Java application based on the *Connect* framework. The tool allows the graphical instantiation of known types of sensor, actuator and complex components. The lists of available ports, individual attributes as well as the component's graphical representation are coded into the class structure and component's data model. Instantiated components can be configured using the tool. In addition, it is possible to generate service-based actuator components from a WSDL file. Connections between component ports are created and modified graphically at runtime using drag and drop gestures (cf. Pipeline metaphor [13]). That way, circuits for interactions consisting of sensors, actuators, logical components and temporal components can bes designed. For certain types of sensor events there are sliders that are used for setting activation thresholds. As many interaction devices provide sensor data in the form of numerical values–not just Boolean values for the active/inactive states–the definition of activation thresholds increases the accuracy of event detection/activation and supports individual user configurations.

Fig. 5 shows a screenshot of the configuration tool's user interface containing three instances of sensor components (BCI modes), logical components and an actuator component (service robot). The tool's user interface provides visual feedback regarding currently active sensor events, connections, and actions as well as numerical values for sensor input.

Prototype Configurations and User Profiles

The composition of components as well as their interconnections, attributes and port thresholds can be persisted in individual prototype configurations and user profiles based on the class model presented in the previous section. These settings are saved in and loaded from XML-based files. In this way, individual interaction concepts can be created for specific prototypes, component configurations, scenarios and users according to their capabilities. These model-based configurations can then be used as templates for creating new interaction concepts or for refinement at a later stage of the development process [18]. Listing 1 shows an extract of a prototype configuration describing an event port of a sensor component connected to an action port.

Listing 1. Extract from a prototype configuration

```
<conf>
 <component>
  <class>IO.TabletButtonComponent</class>
  <id>TabletButtonComponent</id>
   <ports>
    <port>
          <class>Core.EventPort</class>
          <id>Up</id>
          <state>false</state>
          <connectedPorts>
           <connectedPort>
            <componentId>TurtleBot</componentId>
            <portId>MoveForward</portId>
           </connectedPort>
          </connectedPorts>
         </port>
   </ports>
  </component>
 <component>
  ...
</conf>
```

Prototyping and Evaluation

With the help of the *Connect* framework and the configuration tool, interaction concepts describing mappings between interaction devices and active controllable components can be created. Once integrated into the framework, multiple instances of multimodal sensor, actuator and complex components are ready to be used and loosely-coupled at runtime for a particular setting. Connections between sensors and actuators and their respective ports are modifiable at runtime (create, update, delete). Compared to hard-wired event-to-action mappings, model-based prototypes of interaction concepts can be created and modified quickly with the help of *Connect* in order to test and evaluate their usability and suitability for concrete use cases as part of the software engineering process.

Connect can be part of various user-centered prototyping and evaluation methods and stages reusing models that describe components and their interrelations. As the prototyping tool follows known metaphors from the WIMP paradigm and integrates easy to understand graphical metaphors (e. g., pipelines and circuits), it is also possible for designers, non-programmers and end-users to understand and define interaction concepts for a given scenario. That way, future users can be involved in early stages of the design process leading to more intuitive interactions and usable applications for ubiquitous systems.

Due to the model-driven approach for describing components and their interrelations applied in *Connect*, it is possible to persist and reload component configurations and their connections. Prototypes of interaction concepts can be repeatedly tested, reused and refined in order to increase the usability of the interactive application that is under development. By creating user profiles for specific users, user groups and scenarios, the corresponding prototypes can be used for user-centered evaluation methods (e. g., user studies and expert evaluation) during various stages of the software development process. In addition, user profiles facilitate the creation of interactive applications according to a user's individual cognitive capabilities and preferences.

The extensibility of the framework's underlying models for components and interactions allows for the integration of new interactive devices and applications at later engineering stages. That way, interaction concepts can be developed starting from simple event-to-action mappings and evolved to more complex models and scenarios for interactive ubiquitous systems. Changes within these models can be propagated to the affected software artifacts. By adding a mechanism for model versioning, traceability for the evolution of these interaction models in iterative stages of the design process is achieved and templates of interaction concepts can be created for reuse in new projects.

DISCUSSION

The introduced *Connect* framework for the creation and iterative advancement of interaction concepts is built on model-based components and configurations. Our chosen design allows for convenient extension through the inclusion of additional arbitrary interaction components (i. e., sensors and actuators) via concepts of object-orientation. Basic design patterns reduce the effort for developers to extend the set available interaction devices and applications. At runtime, interaction components can be rapidly combined into pipelines and circuits to form complex interaction concepts. Modifications of interaction concepts as well as persisting and loading of interaction and component configurations are supported by the framework. Compared to related research, this approach allows for a high level of reusability and refinement of interactive prototype applications and configuration in succeeding development stages. With the help of the graphical configuration tool it is possible to rapidly create working prototypes for multimodal interactive applications and use these prototypes for test and evaluation purposes.

Due to the use of known metaphors, e. g., pipelines and circuits, the creation and configuration of complex interaction concepts can be achieved even by non-programmers such as interaction designers or end users, supporting different phases of iterative software engineering processes like UCD. By providing model-based abstractions for component design and data flow, interaction design may be advanced from simple prototypes during requirement analysis to complex multimodal interaction concepts containing numerous different sensors and actuators in subsequent phases of development. This iterative improvement of existing concepts proved to be especially helpful during usability testing using expert evaluation and end user studies.

On the one hand, the introduction of logical and temporal components into more complex interaction concepts allows for the use of multiple interactive input devices. On the other hand, these components enable interaction modes and the definition of interaction sequences to prevent unintended interaction events. These mechanisms become necessary when using imprecise and "always on" interaction devices (e. g., brain-computer interfaces and eye trackers) to prevent Midas touch. With respect to the BCI, a double-blink within a certain timeframe could, for example, be used to trigger an action instead of a simple "natural" blink. As shown in the prototyping tool, thresholds defining the activation of sensor ports can be defined for sensor events containing Integer or Double values. This mechanism also helps interaction designers with the integration of imprecise devices. Lastly, a security component can be added to the interaction concept and test environment to stop all ongoing interactions in case of any malfunctions. This is especially helpful when interacting with real world physical devices in ubiquitous systems.

The *Connect* framework is a prototyping tool for engineering interactive applications. It can be employed during the development process for designing and testing of interaction concepts. In combination with other frameworks for the prototyping of user interfaces (e. g., MINT [8], iStuff [2]), interactions (e. g., OpenInterface [13]) and interaction devices (e. g., ProtUbique [11]) as well as for the fusion of high-level sensor events (e. g., Mudra [10]), *Connect* represents an addition to the engineering toolchain for interactive ubiquitous systems. Due to its extensibility and model-driven development approach, interaction and configuration models created with *Connect* could be used as input for subsequent tools and phases in the software engineering process.

The current development state of the *Connect* framework still contains some shortcomings that will be improved in consecutive versions. Until now, user defined data types beyond simple Boolean values at event and action ports are not supported. To be able to accommodate analogue sensors, at least some form of floating point data and complex data types will have to be included into the extensible data model. In addition, aggregation of components attached to distributed instances of the framework is not yet possible. With the availability of active components that contain significant processing power themselves such as smartphones, integration of preprocessed sensor values can be simplified by combining multiple networked *Connect* systems.

CONCLUSION & FUTURE WORK

Engineering software for interactive ubiquitous systems requires flexible and iterative development processes. The continuous involvement of future users throughout the entire process is a key aspect of the user-centered design and development methodology for ubiquitous systems. Prototypes are a well suited tool for the development, test and evaluation of theoretical concepts in almost all stages of the software engineering process. We developed an extensible and easy to use framework that supports rapid prototyping, evolution and evaluation of interaction concepts for ubiquitous systems. The *Connect* framework is based on a modular object-oriented software model, which views interaction devices as sensor components and interactive applications as actuator components. Interaction concepts can be defined, modified and tested at runtime by connecting these components. The framework follows a model-driven software engineering approach enabling the extension and integration of new types of components into interactive prototypes as well as the reuse of component and prototype configurations during development. Related frameworks and concepts generally lack extensibility, flexibility and reusability due to the limited use of models and other formalisms. Therefore, interaction frameworks often support only a static set of interaction devices and applications that can be used for the development of short-living interaction prototypes at early design stages. With *Connect*, the set of software components can be easily extended to support new types of devices and applications, which can be combined to create multimodal interactions. *Connect*'s runtime and user-friendly prototyping tool facilitate the use of multiple input and output devices as entities involved in interactions as well as dynamic reconfiguration of interactions at runtime. The model-based descriptions of components and interactions leverage the reuse and iterative refinement of components, concepts and prototypes for the user-centered software engineering process of ubiquitous systems.

Regarding future work, we will extend the component models in order to be able to process non-Boolean input and output values and states for event and action ports. The use of a formal definition language for describing the interfaces of a sensor component will add the ability to also introduce new types of sensor components to *Connect* at runtime. By combining the prototyping framework *ProtUbique* [11] for defining interaction sources and high-level interaction events with service-based communication for sending interaction events to *Connect*, we will create a flexible toolchain for developing prototypes of interactive multimodal applications. In addition, dynamic component platforms (e. g., OSGi [1]) can be employed to introduce additional runtime flexibility concerning the support of new types of software components. We will also look into the distributed communication among several instances of the *Connect* runtime. An instance of *Connect* running on one computer could be used as a sensor component within another *Connect* instance, which allows the preprocessing and derivation of higher order events on local computers in order to save resources and simplify the modeling of complex interactions. To evaluate the framework's applicability, we will use *Connect* for the prototyping of interactive software components as part of the engineering process for Smart Home applications [19, 12].

REFERENCES

1. OSGi Alliance. 2003. *Osgi service platform, release 3*. IOS Press, Inc.

2. Rafael Ballagas, Faraz Memon, Rene Reiners, and Jan Borchers. 2007. iStuff mobile: rapidly prototyping new mobile phone interfaces for ubiquitous computing. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM, 1107–1116.

3. Rafael Ballagas, Meredith Ringel, Maureen Stone, and Jan Borchers. 2003. iStuff: a physical user interface toolkit for ubiquitous computing environments. In

Proceedings of the SIGCHI conference on Human factors in computing systems. ACM, 537–544.

4. Jullien Bouchet and Laurence Nigay. 2004. ICARE: a component-based approach for the design and development of multimodal interfaces. In *CHI'04 extended abstracts on Human factors in computing systems*. ACM, 1325–1328.
5. Erik Christensen, Francisco Curbera, Greg Meredith, Sanjiva Weerawarana, and others. 2001. Web services description language (WSDL) 1.1. (2001).
6. Anind K Dey, Gregory D Abowd, and Daniel Salber. 2001. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human-computer interaction* 16, 2 (2001), 97–166.
7. Pierre Dragicevic and Jean-Daniel Fekete. 2001. Input device selection and interaction configuration with ICON. In *People and Computers XVInteraction without Frontiers*. Springer, 543–558.
8. Sebastian Feuerstack and Ednaldo Pizzolato. 2011. Building multimodal interfaces out of executable, model-based interactors and mappings. In *Human-Computer Interaction. Design and Development Approaches*. Springer, 221–228.
9. John D Gould. 2000. How to design usable systems. *Readings in Human Computer Interaction: Towards the Year* (2000), 93–121.
10. Lode Hoste, Bruno Dumas, and Beat Signer. 2011. Mudra: a unified multimodal interaction framework. In *Proceedings of the 13th international conference on multimodal interfaces*. ACM, 97–104.
11. Christine Keller, Romina Kühn, Anton Engelbrecht, Mandy Korzetz, and Thomas Schlegel. 2013. A Prototyping and Evaluation Framework for Interactive Ubiquitous Systems. In *Distributed, Ambient, and Pervasive Interactions*. Springer, 215–224.
12. Suzanne Kieffer, J-YL Lawson, and Benoit Macq. 2009. User-centered design and fast prototyping of an ambient assisted living system for elderly people. In *Information Technology: New Generations, 2009. ITNG'09. Sixth International Conference on*. IEEE, 1220–1225.
13. Jean-Yves Lionel Lawson, Ahmad-Amr Al-Akkad, Jean Vanderdonckt, and Benoit Macq. 2009. An open source workbench for prototyping multimodal interactions based on off-the-shelf heterogeneous components. In *Proceedings of the 1st ACM SIGCHI symposium on Engineering interactive computing systems*. 245–254.
14. Linchuan Liu and Peter Khooshabeh. 2003. Paper or interactive?: a study of prototyping techniques for ubiquitous computing environments. In *CHI'03 extended abstracts on Human factors in computing systems*. ACM, 1030–1031.
15. Luqi. 1989. Software evolution through rapid prototyping. *Computer* 22, 5 (1989), 13–25.
16. Carsten Magerkurth, Richard Stenzel, Norbert Streitz, and Erich Neuhold. 2003. A multimodal interaction framework for pervasive game applications. In *Workshop at Artificial Intelligence in Mobile System (AIMS), Fraunhofer IPSI*.
17. Martin Maguire. 2001. Methods to support human-centred design. *International journal of human-computer studies* 55, 4 (2001), 587–634.
18. Nicolai Marquardt, Robert Diaz-Marino, Sebastian Boring, and Saul Greenberg. 2011. The proximity toolkit: prototyping proxemic interactions in ubiquitous computing ecologies. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*. ACM, 315–326.
19. Ronny Seiger, Tobias Nicolai, and Thomas Schlegel. 2014. A Framework for Controlling Robots via Brain-Computer Interfaces. In *Mensch & Computer 2014–Workshopband: 14. Fachübergreifende Konferenz für Interaktive und Kooperative Medien–Interaktiv unterwegs-Freiräume gestalten*. Walter de Gruyter GmbH & Co KG, 3.
20. Anoop K Sinha and James A Landay. 2003. Capturing user tests in a multimodal, multidevice informal prototyping tool. In *Proceedings of the 5th international conference on Multimodal interfaces*. ACM, 117–124.
21. Wolfgang Wahlster. 2006. *SmartKom: foundations of multimodal dialogue systems*. Vol. 12. Springer.
22. Mark Weiser. 1991. The computer for the 21st century. *Scientific american* 265, 3 (1991), 94–104.
23. Mark Weiser. 1993. Some computer science issues in ubiquitous computing. *Commun. ACM* 36, 7 (1993), 75–84.

Challenging Documentation Practices for Interactions in Natural User Interfaces

Bashar Altakrouri
Ambient Computing Group
Institute of Telematics
University ofLübeck
Lübeck, Germany
altakrouri@itm.uni-luebeck.de

Andreas Schrader
Ambient Computing Group
Institute of Telematics
University of Lübeck
Lübeck, Germany
schrader@itm.uni-luebeck.de

ABSTRACT
Dozens of novel natural interaction techniques are proposed every year to enrich interactive eco-systems with multitouch gestures, motion gestures, full body in motion, etc. We present a novel investigation of the community's applied documentation practices for Natural User Interfaces (NUI). Our investigation includes analyzing a survey targeted at NUI designers and a large sample of recently published multitouch and motion-based interaction papers. To the best of our knowledge, this paper is the first to offer a close investigation of this kind. The results reveal that good NUI documentation practices are rare and largely compromised. Thus, we argue that engineering interactive systems for large-scale dynamic runtime deployment of existing interaction techniques is greatly challenged.

Author Keywords
Natural User Interfaces (NUI); Gesture Interfaces; Motion Interfaces; HCI modeling; HCI documentation; HCI sharing.

ACM Classification Keywords
H.5.m. Information Interfaces and Presentation (e.g., HCI): Miscellaneous; H.5.2 Information interfaces and presentation (e.g., HCI): User Interfaces

INTRODUCTION
Calls arise to explore new potential in designing for the whole body in motion as part of the NUI paradigm [3][4], to facilitate users' interactions with real-world pervasive ecosystems (ambient spaces). In the literature, different definitions of NUI [6] were elaborated, which mostly refer to the user's natural abilities, practices, and activities to control interactive systems. Devised from Wachs et al. [10], interactions with NUI can be shortly defined as voice-based and kinetic-based interactions. Kinetic-based interactions are mostly caused and characterized by motion and movement activities, ranging from pointing, clicking, grasping, walking, etc. [2]

Workshop on Large-scale and model-based Interactive Systems: Approaches and Challenges, June 23 2015, Duisburg, Germany.

Herein, we focus on a subset of Kinetic-based interactions, namely multitouch- and motion-based interactions. In the last decade, touch and motion enabled technologies found their way commercially and became widely accessible to the end user, in various application domains such as gaming (e.g., motion-controlled active play by Microsoft Kinect[1] or the Wii system[2]), data browsing, navigation scenarios (e.g., tilting for scrolling photos as in iOS[3] and Android[4] devices) and many more.

Despite the immense progress and success in different application domains, interactive environments will pose additional significant challenges to the design, engineering and deployment of NUI technologies. Considering user heterogeneity, e.g. due to aging and demographic change ("come-as-you-are" paradigm), user mobility to unknown environmental settings at design time (interaction context) and spontaneous construction of interactive environments in-situ at runtime, the isolated design of natural interface devices will not be sufficient any more, regardless of the quality and naturalness of the proposed interaction scheme per se. In their work, Altakrouri and Schrader [2] proposed a shift towards completely dynamic on-the-fly ensembles of interaction techniques at runtime. The Interaction Ensembles approach is defined as "Multiple interaction modalities (i.e., interaction plugins) from different devices are tailored at runtime to adapt the available interaction resources and possibilities to the user's physical abilities, needs, and context" [2]. A shift of this kind imposes new dissemination, deployment, and adaptation requirements for engineering interaction techniques and interactive systems for NUI. Precisely for those reasons, better understanding and analysis of the practiced documentation habits of interaction techniques for NUI plays a major role to bridge the possible gaps between designing interaction techniques and engineering interactive systems.

In this paper, we present a novel investigation of the community's applied documentation practices for interactions in NUI. We believe that an investigation of this kind is essential to understand some of the challenges for engineering interactive systems in ambient spaces and setting proper interac-

[1]http://www.microsoft.com/en-us/kinectforwindows/, latest access on 25.03.2015.
[2]http://www.nintendo.com/wii, latest access on 25.03.2015.
[3]http://www.apple.com/ios/, latest access on 25.03.2015.
[4]http://www.android.com/, latest access on 25.03.2015.

tion dissemination guidelines, where interactions are becoming increasingly dynamic, adaptive and multi-modal.

Our novel investigation is concluded by analyzing a survey targeted at NUI designers and a large sample of recently published multitouch and motion-based interaction papers. Although limited in scale, we believe that this investigation opens the door for important open research issues for the CHI and EICS community around this problem domain.

In this paper, the term documentation is used to capture the way an interaction technique is defined and described by the interaction designer (i.e. developer). Principally, documentation refers to any written material, visual clues, animated clues, formal description models and languages, etc, used to describe or disseminate the developed interaction. The literature covers various approaches to describe touch-based interactions. An extensive review on those approaches is out of the scope of this paper. In their work about formal descriptions for multitouch interactions, Hamon et al. [5] analyzed the expressiveness of various user interface description language (an extension to [8]) and suggested the ICO formalism for modeling multitouch interactions. Principally, modeling includes data description, state representation, event representation, timing, concurrent behavior, dynamic instantiation, etc. Recently, Altakrouri et al. [1] targeted their effort to describe the movement aspects of motion-based gestures and the physical context (i.e., abilities and disabilities) of the user.

Documenting interaction technique is relevant for the correct execution of interactions by end users, the preservation of technique by designers, the accumulation of knowledge for the community, and the engineering of interactive systems. We argue that documenting interactions should be treated as an important resource of context information about the interaction technique, which can be also utilized by interactive systems for various reasons. For instance, filtering relevant interaction techniques at runtime in response to the user's physical context (e.g., disabilities) as in the Interaction Ensemble approach mentioned above.

Better understanding of the currently applied documentation practices does not only reveal the current dissemination strategies but also triggers possible needs for new tools, guidelines, and systems that improve those practices and ultimately bridge the gap between the design of single interaction techniques and the development of interactive systems.

In this paper we will substantiate the following main contributions and findings:

- We present a number of observations regarding the NUI designers' most commonly applied documentation choices, most importantly, documentation frequency and media type of choice.
- We unveil that NUI documentation is largely underestimated and compromised by NUI designers due to the lack of adequate documentation tools, absence of documentation standards, and irregularity of documentation habits.

METHODOLOGY

Our study included two investigation areas: (1) analyzing a tailored survey targeted at NUI designers and (2) coding and analyzing a large sample of recently published multitouch and motion-based interaction papers. In this section, we first outline our approach before we present the results in the following section.

Survey on NUI documentation

The first step in our review was to capture a snapshot on the current most employed practices for NUI documentation by carrying out an online survey. The survey aimed to partially characterize a number of designers' documentation practices, including: (1) The adoption level and frequency of documentation practices and standards in design and development of NUI; (2) The designers' satisfaction with their practiced NUI documentation habits; (3) The needs for new documentation tools and methods; (4) The commonly used documentation methods, tools, and media types; and (5) The perceived importance of documentation for sharing, acceptance, user experience, and correctness.

The survey was targeted at both NUI designers (i.e. NUI developers) and users, it was split into two sections accordingly. In this paper, we only focus and report about the designer section, which contained a total sum of 11 different multiple choice and likert scale questions. The survey was bound to a maximum completion time of 3 minutes to maximize the number of voluntary participations. The survey included an introductory section where the notion of NUI, specially for multitouch- and motion-based interfaces, as well as the purpose of the survey were introduced.

The survey was distributed online through specialized HCI mailing lists (including BCS-HCI run by the British Computer Society Human-Computer Interaction Group[5]), ubiquitous computing mailing lists (including Ukubinet-announce run by the Imperial College London[6] and announcements@ubicomp.org[7]), Lübeck university mailing lists, and social networks (i.e. Facebook, Twitter, and ResearchGate). The survey was open for participation for about 3 weeks.

Analyzing the interaction publications landscape

The second step in our review intended to capture a closer look at the published work in the area of interaction techniques. In order to find out how the community expresses, documents, and shares interaction techniques, we have decided to base our investigation on a collection of the most recent ACM published work under the ACM classification (H.5.2 Information Interfaces and Presentation: User Interfaces - Input devices and strategies) for the years 2012 and

[5]https://www.jiscmail.ac.uk/cgi-bin/webadmin?A0=bcs-hci, latest access on 25.03.2015.

[6]https://mailman.ic.ac.uk/mailman/listinfo/ukubinet-announce, latest access on 25.03.2015.

[7]http://mail.ubicomp.org/mailman/listinfo/announcements_ubicomp.org, latest access on 25.03.2015.

2013 (until 22.08.2013). Out of 518 total papers in this category, we manually coded and analyzed a total sum of 93 papers that matched one of two categories: (1) papers presenting novel interaction techniques; (2) papers applying or analyzing existing interaction techniques in various scenarios. Our filtering criteria excluded all none touch or none motion gesture papers (as considered out of the focus of this investigation), video papers (as those papers don't have enough space to cover the interaction technique and only convey very limited aspects of the work), and duplicated paper entries (if the same work was presented in multiple venues but with different contribution size, e.g., work-in-progress papers, short papers, full papers). In the case of duplication, the latest and longest contribution was considered. Our aim was not to conduct a complete and detailed review of all published papers. Instead, we aimed at providing a snapshot at the most recent published work as a living example of the current practiced documentation habits.

Our analysis and classification are based on the published paper and any corresponding material directly mentioned, linked, or attached with the published work (e.g., many published papers have also videos attached within the ACM library, or links to external resources). Other materials out of the aforementioned criteria were considered hidden and were not included in the study, such as in application help menus or offline accessible manuals.

The papers were coded based on four main aspects: *Type* - gesture types discussed in the paper including multitouch and motion gestures; *Still* - used still media types to document and describe the gesture including text, images, and sketches; *Animated* - used animated media types to document and describe the gesture including videos, animations, personal walkthrough, and onscreen walkthrough; and finally *Authoring* - reported or used authoring and documentation tools and formal languages. Our main goal of this analysis was to highlight general practices and habits rather than focusing on a particular paper title or the authors. Hence, we reference the reviewed papers by the unique identification key (ACM ID) instead of the papers' full title or author names.

RESULTS AND OBSERVATIONS

In this section, we present the results for each of our investigation areas. We have supported the data with a number of general observations to enhance the readability of the results. The observations are numbered and marked with an abbreviation to the corresponding section (D: Designer survey section and P: Papers analysis section).

Survey

A total of 332 anonymous individual responses were recorded, split into 267 NUI users (80% of the total respondents) and 65 NUI designers (20% of the total respondents).

The designer respondents are split to 11 expert designers, 14 professional designers, 28 competent designers, 10 advanced beginners, and finally 2 novice designers. This categorization is based on an explicit survey question about expertise self-assessment.

We have applied Kruskal Wallis test to identify any statistically significant differences among expertise groups. In most cases, no statistical differences amongst groups were found unless explicitly mentioned in the text.

*Observation - **D1**: Small majority of NUI designers are satisfied with their current documentation practices*: 57% of the designers responded positively to a question on the satisfaction with their current documentation habits.

*Observation - **D2**: Only a small minority of NUI designers practice NUI documentation continuously*: Figure 1 shows how often the designer respondents document designed NUI, independent of form or documentation type. The figure reveals that the majority of the respondents practice documentation either sometimes (42%) or frequently (38%). Merely small minority of designers (14%) practice documentation regularly. Statistically significant difference was identified among expertise (H(4) = 13.466, p = 0.009) with a mean rank of 43.93 for proficient, 33.75 for competent, 32.91 for expert, 18.70 for advanced beginner, and 18.0 for novice designers. Higher mean ranks indicate a more frequent documentation practice.

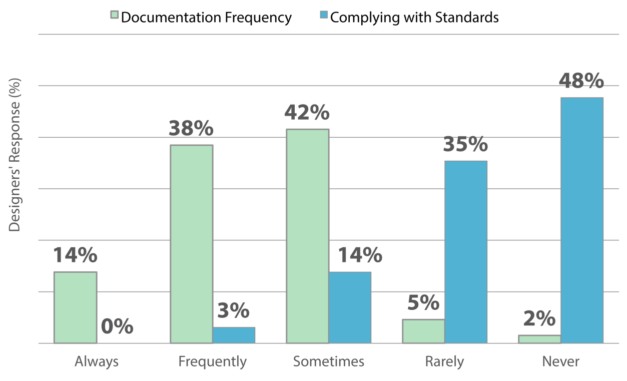


Figure 1. Practicing NUI Documentation and Complying with Standards

*Observation - **D3**: The vast majority of NUI designers never or rarely apply documentation standards*: One interesting aspect in this survey is to highlight the designers' habits to apply standard documentation approaches, as shown in Figure 1. The survey unveils that about half of the respondents never apply any documentation standards and merely a third did on rare occasions. Small number of respondents apply documentation standards either sometimes (14%) or frequently (3%).

*Observation - **D4**: The majority of NUI designers indicated a lack of NUI documentation tools and methods*: The respondents answered positively (66%) when asked whether there is a lack of NUI documentation methods and tools available for them to use.

*Observation - **D5**: NUI are mostly documented using text, pictures, sketches, and videos respectively*: Another goal of the survey was to identify the dominant media types used by designers to document interaction techniques. Figure 2 illustrates the distribution of used NUI documentation media types by designers. Text is the most used medium to describe and document NUI. Still visual documentation records (i.e. pictures and sketches) follow next. Moreover, animated visual records come fourth. Additionally, audio and formal languages come last with very low percentages.

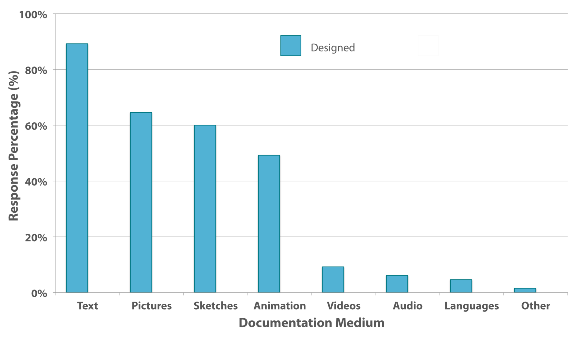


Figure 2. Medium for Documentation

*Observation - **D6**: NUI are rarely documented using formalized languages*: Figure 2 also shows clearly designers don't follow formalizations as a documentation media type.

*Observation - **D7**: The most ranked importance of NUI documentation is acknowledged for sharing NUI, followed by user experience*: Figure 3 illustrates the designers' perceived importance of NUI documentation for sharing, experience, acceptance, and correctness. The vast majority of responders scored documentation as a very important (45%) or an important (37%) factor for a successful sharing of NUI. Regarding user experience, the majority of respondents scored the documentation as an important (48%) or a very important (11%) factor respectively. Moreover, designers scored NUI documentation for user acceptance as very important (14%), important (40%), moderate (25%), and of little importance (18%). Merely 3% negatively scored documentation as unimportant for the user acceptance. Finally, the majority of respondents scored documentation as either an important (40%) or very important (26%) factor for the correctness of NUI execution. Approximately one third of the respondents scored documentation as moderate or of little importance for correctness.

Scientific publications

Figure 4 illustrates the complete classification of the analyzed papers based on the previously presented methodology. Papers that satisfy the conditions are distinguished with a coding mark as shown in the figure. The analyzed papers were motion (51%) and touch based (68%) interaction papers (note that a paper may fall into more than one category).

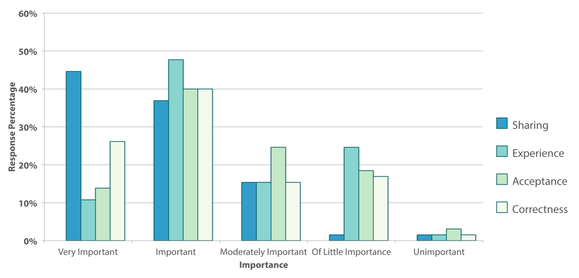


Figure 3. The Designers' Perceived Importance of Documentation for Acceptance, Correctness, Experience, and Sharing of NUI

*Observation - **P1**: NUI in publications are mostly documented using text, sketches, and pictures respectively*: As expected, figure 4 shows that text descriptions as a medium for documenting interaction techniques is used in all of the papers that we have reviewed. Sketches (59%) come second with a very close match with the designer survey in Figure 2. Pictures (53%) come third, slightly lower than in the designer survey.

Moreover, personal walkthrough is reported by 16% (the developer introduces the interaction technique to other developers or users by demonstration). Videos are reported by 11%. This percentage matches the survey's results (Figure 2). In research papers, mentioning and linking to video content is usually neither required nor critical for the acceptance of the research paper. Hence, videos related material to the technique are often hidden. The use of animations is reported only once. This matches to a large extend the designer survey results in Figure 2. On the other hand, other media types such as onscreen walkthrough are hardly used.

*Observation - **P2**: NUI in publications are never documented using formalized languages or interaction authoring tools*: To our expectations, none of the papers reported or used languages (including notations and formalisms) or interaction authoring tools (including gesture authoring tools). Finally, we found no statistical difference between the two main aforementioned analyzed groups of papers.

DISCUSSION

In this section, we present a number of interesting aspects regarding NUI documentation practices and possible impact on designing and engineering interactive systems.

Documentation habits: Ignorance or underestimation?

Our results show that the majority of the designer respondents are satisfied with their current NUI documentation practices (D1). Nonetheless, this satisfaction is not necessarily reflected on the quality and extend of applied documentation practices (D2, D3). Those observations unveil that NUI documentation is generally an underestimated or ignored problem by interaction designers and developers.

Clearly, the NUI paradigm vastly grows in terms of the number of interaction proposed, the diversity of interaction types, involved body parts, involved actions, etc. [3][4][8]. Great advancements, in terms of innovation and usability evaluation, of this type of interaction are usually proposed and presented at various venues and conferences such as ACM CHI (Human Factors in Computing Systems) and UIST (User Interfaces Software and Technology). Despite this effort, some researchers believe that very little effort is actually targeted to improve the reliability of systems offering and adoption these kinds of novel interaction techniques [8].

Soon the lack of adequate interaction documentation and dissemination will lead to challenge the design and engineering of interactive systems. Documentation can be used to

Authoring	Languages																		
	Tools																		
Animated	Walkthrough (S)																		
	Walkthrough (P)					⊙			⊙				⊙		⊙				
	Animation												⊙						
	Video											⊙							
Still	Sketches	⊙		⊙	⊙	⊙		⊙			⊙	⊙	⊙	⊙				⊙	⊙
	Pictures	⊙	⊙	⊙					⊙	⊙					⊙			⊙	⊙
	Text	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙
Type	Motion		⊙	⊙					⊙	⊙		⊙	⊙			⊙	⊙	⊙	⊙
	Touch	⊙		⊙	⊙	⊙	⊙	⊙	⊙		⊙			⊙	⊙				
Use	ACM ID	2481423	2481424	2468582	2468590	2468630	2468400	2468487	2491369	2465966	2465961	2465972	2491573	2449452	2459248	2447609	2460694	2460702	2388706
		40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57

Authoring	Languages																		
	Tools																		
Animated	Walkthrough (S)																		
	Walkthrough (P)		⊙			⊙	⊙							⊙	⊙				
	Animation																		
	Video					⊙								⊙					⊙
Still	Sketches	⊙		⊙									⊙			⊙	⊙		⊙
	Pictures		⊙					⊙	⊙	⊙	⊙	⊙		⊙	⊙			⊙	
	Text	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙
Type	Motion			⊙	⊙	⊙	⊙		⊙	⊙		⊙		⊙	⊙	⊙	⊙	⊙	⊙
	Touch	⊙	⊙	⊙		⊙				⊙	⊙	⊙	⊙			⊙			⊙
Use	ACM ID	2399091	2384945	2305305	2399103	2168636	2393165	2148191	2379264	2370265	2305297	2318032	2414572	2148190	2254669	2371599	2325644	2414614	2148142
		58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75

Authoring	Languages																		
	Tools																		
Animated	Walkthrough (S)																		
	Walkthrough (P)																		
	Animation																		
	Video	⊙																	
Still	Sketches	⊙					⊙	⊙	⊙			⊙	⊙	⊙	⊙	⊙	⊙	⊙	
	Pictures		⊙	⊙	⊙	⊙	⊙			⊙	⊙	⊙		⊙					⊙
	Text	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙
Type	Motion	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙		⊙		⊙				⊙	⊙	⊙
	Touch	⊙	⊙						⊙	⊙		⊙		⊙	⊙	⊙			
Use	ACM ID	2399104	2379644	2254582	2350070	2371601	2223823	2208330	2223689	2396669	2223782	2208391	2396650	2384993	2208390	2350073	2208585	2371589	2208575
		76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93

Authoring	Languages																				
	Tools																				
Animated	Walkthrough (S)							⊙													
	Walkthrough (P)			⊙								⊙			⊙				⊙		
	Animation																				
	Video													⊙							
Still	Sketches	⊙	⊙	⊙		⊙	⊙	⊙		⊙	⊙		⊙				⊙		⊙		⊙
	Pictures	⊙	⊙		⊙		⊙		⊙	⊙	⊙	⊙		⊙	⊙	⊙		⊙	⊙	⊙	
	Text	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙
Type	Motion	⊙	⊙	⊙		⊙		⊙				⊙	⊙		⊙	⊙	⊙	⊙		⊙	⊙
	Touch	⊙	⊙	⊙	⊙		⊙	⊙			⊙		⊙			⊙			⊙		
Novel	ACM ID	2466192	2481330	2466457	2470704	2468587	2468592	2468602	2468636	2468645	2468525	2491368	2491593	2451207	2449435	2459238	2459240	2447571	2460654	2460668	2460661
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

Authoring	Languages																			
	Tools																			
Animated	Walkthrough (S)																			
	Walkthrough (P)		⊙													⊙				
	Animation																			
	Video	⊙										⊙			⊙				⊙	
Still	Sketches	⊙	⊙	⊙		⊙		⊙	⊙	⊙			⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙
	Pictures	⊙	⊙				⊙		⊙		⊙	⊙			⊙	⊙				
	Text	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙	⊙
Type	Motion	⊙	⊙	⊙	⊙		⊙	⊙	⊙	⊙	⊙	⊙		⊙		⊙		⊙		⊙
	Touch	⊙		⊙		⊙								⊙	⊙		⊙	⊙	⊙	⊙
Novel	ACM ID	2318005	2168578	2305306	2254585	2414585	2318071	2399149	2254590	2148195	2384969	2148148	2148172	2223673	2371712	2407360	2396697	2208297	2371711	2350098
		21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39

* Under ACM classification H.5.2 Information Interfaces and Presentation: User Interfaces—Input devices and strategies
* Years 2012 and 2013 (Until 22.08.2013)
* Novel: New and Novel Interaction Technique Paper
* Use: Gesture based Interaction Paper (using interactions, analysing interaction techniques, application scenarios, etc)

Figure 4. ACM Multitouch and Motion based Interaction Papers Review - Interaction Documentation Practices and Habits Analysis for The Years 2012 and 2013 (until 22.08.2013)

extract information about the type of movements involved in the interaction, involved body parts, adequate interaction execution, etc. The absence of such information will necessarily lead to burden the deployment of interaction techniques in automated interactive systems, especially processes such as context acquisition, reasoning, interaction filtering, etc. are greatly hindered. The absence of documentation inevitably challenges analyzing the proposed interaction techniques. For instance, requirement analysis can be greatly compromised, analyzing physical requirements for NUI users is not possible, or correctly reproduce or extend a particular interaction techniques becomes an extremely challenging task.

Documentation types and methods

The results show that various media types are used by designers for documentation. From one hand, designers reported that the most used media types for documenting NUI are text, pictures, sketches, and videos respectively (D5). The academic paper investigation on the other hand shows text, sketches, and pictures respectively used in the analyzed papers (P1). Despite the aforementioned difference, it is clearly visible that designers rarely use formalized languages or approaches as a documentation medium (D3, D6, P2). Formalized description of NUI are an important mean to document various aspects on the interaction to insure integrity and correctness of execution, and the possibility of replicability.

Designers are not in favor of formal languages and audio. This can be due to the complexity of language learning, and the complexity of describing movements respectively. Whilst formalized languages can be hard to learn and apply, they are often used in different fields for documenting movements. Formalized languages have a clear benefit to preserve and transfer the technique to other designers without endangering the originality and vital aspects of the technique. Using the currently applied media such as text, pictures, sketches, and videos may lead easily to losing parts of the movements, overly complicated descriptions, losing timing information, etc.

In fact, according to Navarre et al. [8], formal interfaces description languages support interaction at the development (e.g., prototyping) as well as the operation phase, while conventional empirical or semiformal techniques lack to provide adequate and sufficient insights about the interaction (e.g., comparing two design options with respect to the reliability of the human-system cooperation).

The notion of movement is of particular importance for Kinetic Interactions, as it resides at the core of this type of interactions. Movement documentation is a very relevant and generally a very unresting problem for many fields such as dance choreography, movement rehabilitation, motion recognition and analysis, and human movement simulation. According to Kahol et al. [7], having such languages and notations features three main qualities: facilitate teaching and learning of movement styles, permit the writing of universally-understood scores of movement, and provide a universal language to communicate movements. Nonetheless, Kahol et al. [7] still acknowledge the lack of a formalized languages and notations of generic motion, matching our investigation results (especially D4). We share the same viewpoint as in [8], lacking adequate and formalized documentation lead inevitably to increase the gap between the design and (commercial) deployment of developed interaction techniques.

The importance of documentation

Designers recognize the importance of documentation for the users' experience, the acceptance of NUI techniques, and correctness of use. The most important use of documentation is for sharing NUI techniques (D7). Sharing is particularly important for different purposes such as communicating NUI to other peer designers, improving NUI functionality by other designers, adopting NUI techniques in various interactive eco-systems, and reaching user audience. Even though designers recognized these important roles, their documentation practices appear generally ignorant to this importance.

Documentation challenges in future ambient spaces

So far we have discussed the current NUI documentation practices, but the shift towards future ambient spaces imposes new requirements, and challenges the current practices.

This type of interactive systems aims at avoiding mismatch problems between user's needs and device's offers, by employing the best matching interactions to the given context, hence the user independence (acceptability by permitting customizability) and usability qualities required by Wachs et al. [10] are inherently enhanced. Pruvost et al. [9] noted that interaction environments are becoming increasingly heterogeneous and dynamic, hence they are no longer static and closed; the interaction context is becoming increasingly more complex; and, increasing adaptability is required for sustainable utility and usability.

Current NUI documentation practices, as discussed in this paper, are greatly challenged by such a system. The current documentation practices and strategies are not adequate and fail to meet the challenge of dynamically created documentation for interaction ensembles. Interactions are currently ego-centric and designed in isolations, so is the documentation. Such isolation implies a complete absence of information about the interaction's behavior as part of an ensemble in a dynamically changing eco-system.

FUTURE WORK

As part of our research roadmap, we will continue to explore this field by (1) extending our investigation to study the differences and similarities between NUI documentation in academic and commercial settings (as in motion-based and touch-based application market initiatives); (2) extending our analysis to include NUI users and their learning habits; and (3) extending our ongoing work on a dedicated tool for documenting NUI

CONCLUSION

We have presented an investigation on the applied practices and habits to document and share developed interaction techniques. The analysis included: (1) an online exploratory survey on documenting Natural User Interfaces (NUI) answered by 64 designer; and, (2) coding and analyzing a sample of 93 recently ACM published multitouch and motion-based interaction papers. Our study reveals that good documentation practices are rare and largely compromised. Our survey reveals that there is a lack of documentation tools, methods, and formal languages; designers almost never follow or apply any documentation standards; and designers never use available interaction authoring tools. Hence, the creation of a collective long lasting interaction heritage remains unachievable. Moreover, the gap between developing and rightly disseminating interaction techniques increases. Thus, engineering interactive systems for large-scale dynamic runtime deployment of existing and future interaction techniques is greatly challenged.

REFERENCES

1. Altakrouri, B., Gröschner, J., and Schrader, A. Documenting natural interactions. In *CHI '13 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '13, ACM (New York, NY, USA, 2013), 1173–1178.
2. Altakrouri, B., and Schrader, A. Towards dynamic natural interaction ensembles. In *Fourth International Workshop on Physicality (Physicality 2012) co-located with British HCI 2012 conference*, A. D. Devina Ramduny-Ellis and S. Gill, Eds. (Birmingham, UK, 09 2012).
3. England, D. Whole body interactions: An introduction. In *Whole Body Interaction*, D. England, Ed. Springer London, 2011, ch. Whole Body Interactions: An Introduction, 1–5.
4. Fogtmann, M. H., Fritsch, J., and Kortbek, K. J. Kinesthetic interaction: revealing the bodily potential in interaction design. In *Proceedings of the 20th Australasian Conference on Computer-Human Interaction: Designing for Habitus and Habitat*, OZCHI '08, ACM (New York, NY, USA, 2008), 89–96.
5. Hamon, A., Palanque, P., Silva, J. L., Deleris, Y., and Barboni, E. Formal description of multi-touch interactions. In *Proceedings of the 5th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, EICS '13, ACM (New York, NY, USA, 2013), 207–216.
6. Iacolina, S., Lai, A., Soro, A., and Scateni, R. Natural interaction and computer graphics applications. In *Eurographics Italian Chapter Conference*, E. Puppo, A. Brogni, and L. D. Floriani, Eds., Eurographics Association (Genova, Italy, 2010), 141–146.
7. Kahol, K., Tripathi, P., and Panchanathan, S. Documenting motion sequences with a personalized annotation system. *IEEE MultiMedia 13*, 1 (2006), 37–45.
8. Navarre, D., Palanque, P., Ladry, J.-F., and Barboni, E. Icos: A model-based user interface description technique dedicated to interactive systems addressing usability, reliability and scalability. *ACM Trans. Comput.-Hum. Interact. 16*, 4 (Nov. 2009), 18:1–18:56.
9. Pruvost, G., Heinroth, T., Bellik, Y., and Minker, W. *User Interaction Adaptation within Ambient Environments*, next generation intelligent environments: ambient adaptive systems ed. Springer, Boston (USA), 2011, ch. 5, 153–194.
10. Wachs, J. P., Kölsch, M., Stern, H., and Edan, Y. Vision-based hand-gesture applications. *Commun. ACM 54* (February 2011), 60–71.

A Concerted Model-driven and Pattern-based Framework for Developing User Interfaces of Interactive Ubiquitous Applications

Jürgen Engel
Augsburg University of Applied Sciences
Augsburg, Germany
juergen.engel@hs-augsburg.de

Christian Märtin
Augsburg University of Applied Sciences
Augsburg, Germany
christian.maertin@hs-augsburg

Peter Forbrig
University of Rostock
Rostock, Germany
peter.forbrig@informatik.uni-rostock.de

ABSTRACT
Modeling and building interactive user interfaces (UI) typically requires the skills of software developers and HCI experts who cooperate with platform and marketing experts in order to arrive at solutions with the required software quality, usability, and user experience. The combination of model-driven user interface development practices with pattern-based approaches that specify HCI- and software-patterns in a formalized way and respect emerging standards offers potentialities to facilitate and at least partially automate the user interface development process, therefore reduce the time-to-market and development costs, and lead to solutions that can easily be adapted to varying contexts and target devices. Such pattern-aided UI adaptation is not limited to design time decisions but can also be applied during runtime. This paper highlights the architecture and capabilities of the Pattern-Based Modeling and Generation of Interactive Systems (PaMGIS) framework to broadly support the construction and adaptation of user interface models. It is discussed, how pattern descriptions that capture important parts of the design knowledge should be organized in order to be automatically processed during the modeling process.

Author Keywords
User interfaces; interactive systems; model-driven development; pattern-based development.

ACM Classification Keywords
H.5.m. Information interfaces and presentation (e.g., HCI): User Interfaces.

INTRODUCTION
Highly interactive software has become a crucial ingredient of modern human life. Independent of time and location, people are used to interact with products built around interactive software components, such as web applications, telecommunication devices, car navigation systems, smart home appliances, wearables, or other electronic equipment. Nowadays users expect that software products run on a variety of heterogeneous devices with a consistent look and feel, invariable high usability, and an extremely high degree of appealing user experience. Additionally, users tend to be impatient and want to have the software available for their different devices at the same point in time. Therefore, time-to-market is vital for software suppliers.

It is nearly impossible to meet all requirements simultaneously when exercising traditional software engineering and development processes. A promising way out of this dilemma is the application of a model-driven approach that allows for describing the particular aspects of the intended user interface by means of distinct models at different abstraction levels which can be created - at least partially - by automatic transformations.

We have combined model-driven user interface development practices with pattern-based approaches that specify HCI-patterns in a formalized way [6]. Result is the fundamentally renovated PaMGIS 2.0 framework which is presented in the following sections.

RELATED WORK
Model-based user interface development environments (MB-UIDE) introduce models to the development process of interactive applications. A variety of existing MB-UIDE and model-driven approaches for facilitating the development process of interactive systems can be found in the literature. Related recapitulations and discussions are provided in [3] [5] [9] [12]. The models used by these approaches are usually task-based or object-oriented and contain functional domain and data requirements at different abstraction levels for the interactive system under development. Typically, the models are also used for mapping and linking the functional requirements of the business logic to the different abstract and concrete representations of the user interface with the intent to achieve high user interface quality, usability, and good user experience for the user of the final interactive application. Possible solutions to avoid practical problems and discrepancies between the automatic derivation of user interfaces and their usability are discussed in [11]. Benefits from using model-based user interface development and meaningful use cases are provided in [8].

Workshop on Large-scale and model-based Interactive Systems: Approaches and Challenges, June 23 2015, Duisburg, Germany.

The role of the various models used in MB-UID environments varies with respect to the modeling purpose. Typically more than one model is exploited during the development process to construct the desired solution interactively or (semi-) automatically. Some degree of standardization was brought into the diversity of MB-UIDEs by the CAMELEON Reference Framework (CRF) [1]. CRF proposes the use of domain, context-of-use, and adaptation models. Here, the domain model combines task and concepts sub-models, the context-of-use model consists of user, platform, and environment models, and the adaptation model is separated into evolution and transition models. With regard to model abstraction levels, CRF distinguishes task-oriented specification, abstract user interface, concrete user interface, and final user interface.

HCI patterns are a means to document design decisions based on established design solutions or best practice work and therefore capture fundamental principles for good design. In general, patterns represent a relation between a certain design problem and a solution in a given context. In addition, they are simple and easily readable for designers, developers and researchers, and they alleviate the collaboration between the involved people. In order to ensure a certain standard, patterns are organized in so-called pattern catalogs [1]. A catalog of related patterns that belongs to a common domain is called a pattern language [14]. Since many pattern authors pick their own formal description styles and formats with often different understanding of pattern attributes, several standardization approaches have been introduced, e.g. the Pattern Language Markup Language (PLML) version 1.1. PLML unifies the description schemes of different authors with the help of XML tags which represent the particular characteristics of the patterns. According to PLML 1.1 the documentation of a pattern should consist of the following elements: a pattern identifier, name, alias, illustration, descriptions of the respective problem, context and solution, forces, synopsis, diagram, evidence, confidence, literature, implementation, related patterns, pattern links, and management information [7]. A recapitulation and discussion of existing pattern description standardization approaches is provided in [4].

PAMGIS FRAMEWORK

Basic Concepts

The intention of the PaMGIS framework is to assist and support its users in the process of developing highly interactive user interfaces. As illustrated in Figure 1, the basic concept is to combine both model-driven and pattern-based development methods and techniques.

Hence, descriptions of HCI patterns are equipped with model fragments that on one hand can be used as building blocks for the diverse models and on the other hand allow influencing model transformations. In addition, usability evaluation results can be fed back in order to draw conclusions and improve the patterns, models, and the resulting user interfaces.

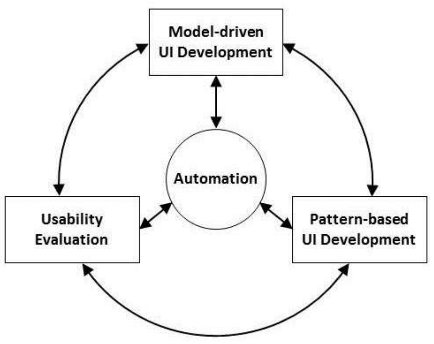


Figure 1. Basic concepts of the PaMGIS Framework

The framework supports our research with respect to the potentials and limits of automated UI development. In order to enable automatic processing, all model entities as well as pattern specifications are expressed and stored in an XML-compliant format.

Model-driven Aspects

The model-driven part of the framework as illustrated in Figure 2 is designed in the style of the CAMELEON Reference Framework. Particularly, the ontological domain and context-of-use models are used as proposed by the CRF. However, we decided to split the CRF platform model into a device model and a UI implementation model. While the former comprises all relevant characteristics of the respective end device the latter holds information about the UI elements that are available on the respective underlying software platform. This avoids redundancies especially in cases where the same software basis supports significantly different devices, e.g. Android on smartphones and tablet computer.

The framework is organized in six abstraction levels, i.e, domain, context of use, abstract user interface (AUI), concrete user interface (CUI), final user interface (FUI), and runtime level. As the most abstract representation, the domain level embodies the domain model which in turn consists of the task and concept sub-models. The task model provides information of domain-specific user goals and the entirety of process steps and actions which must be executed in order to attain these goals. The concepts model can be understood as a type of data model describing all UI-relevant data elements and artifacts which are required in the course of task completion. Hence, these two models are closely interrelated. In the context of PaMGIS, the task model is represented in a ConcurTaskTrees (CTT) notation [10] with some specific adaptations and enhancements which primarily refer to the specification of relationships between certain tasks and the data elements that are required for the execution of these tasks. The concept model is specified on the basis of XML Schema Definition (XSD). The context-of-use model consists of the user, environment, and the already mentioned device and UI implementation sub-models. While the user model holds information about particular characteristics of individual users or clusters of users, e.g. preferences or possible disabilities, the environment model describes environmental influence factors, e.g. lighting conditions, noise, or air pollution.

The knowledge captured within the domain model is used to construct an abstract user interface model which is a canonical representation of the rendering of the domain concepts which is independent from the actually available UI elements as specified within the UI implementation model. At this juncture, the concepts model indicates which AUI objects are required while the task model's hierarchical structure and inherent temporal dependencies enforce the definition of the relationships between these objects.

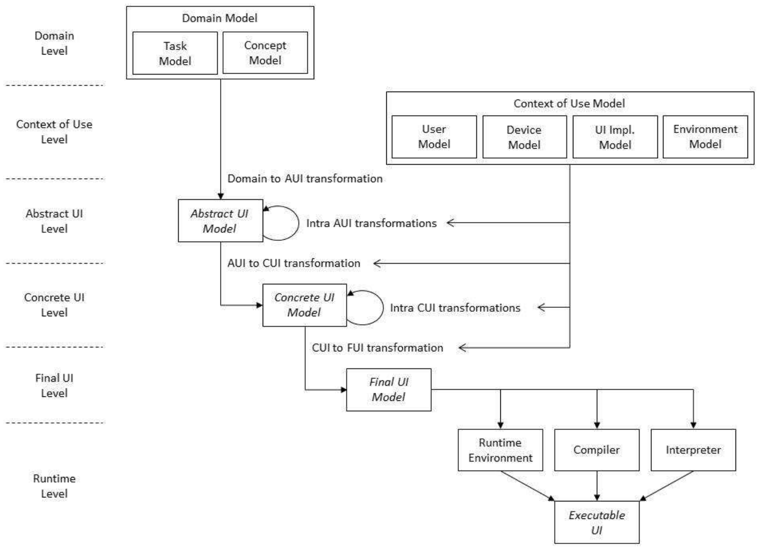


Figure 2. Overview of utilized models and abstraction levels.

A list of feasible AUI objects is provided Table 1.

Abstract UI Object	Description
Activator	Activates another object or initiates a call of a business logic function
Navigator	Facilitates the navigation to another dialog
Output	Displays (read-only) objects of diverse data types
Editor	Similar to Output, but manipulable by the user
SingleChoice	Selection of exactly one item out of several
MultiChoice	Selection of none, one, or more items out of several

Table 1. Examples of supported abstract user interface objects.

The information contained within the context-of-use model is used to control the subsequent transformations of the diverse UI models and to substantiate deliberate design decisions. For instance, some tasks or sub-tasks might be undesired, impractical or impossible to be carried out within a certain context of use due to user-, device-, and/or environment-related restrictions. In this case, the corresponding parts of the AUI have to be eliminated. Furthermore the design of the dialog structure is defined in consideration of the given context of use by means of dialog graphs [13].

Once the AUI model is completed, it can be transformed into a concrete user interface model. For this purpose, the abstract user interface objects are replaced by appropriate concrete ones. In this sense, the most appropriate CUI object is the one that fits best to both the requirements and restrictions which result from the various aspects of the context-of-use model.

Further, a first impression of the final look-and-feel is created by roughly determining the layout, i.e., positioning, and the appearance, e.g. color, font, and size, of the CUI objects. In a last step, the final user interface can be automatically generated from the CUI model.

Figure 3 recapitulates the necessary transformation steps between the four different levels of abstraction as specified in the CAMELEON reference framework. The process starts with the domain model followed by the abstract and concrete model levels and finally arrives at the final user interface. Please note, that the framework user may perform manual adjustments at any step of the development process.

From a runtime perspective, there are three general options how to deal with FUIs. Firstly, the FUI is available as source code that can be transformed into an executable format by means of a compiler. Secondly, the FUI has the format of a script that can be executed by an interpreter. Thirdly, the FUI can be executed by a runtime engine provided with the development framework. The advantage of such a runtime engine is that it is not necessarily bound to the FUI level, but can also create at least executable UI prototypes from higher abstraction levels, i.e., CUI and AUI models, and therefore enables framework users to identify design problems in early stages of the development process.

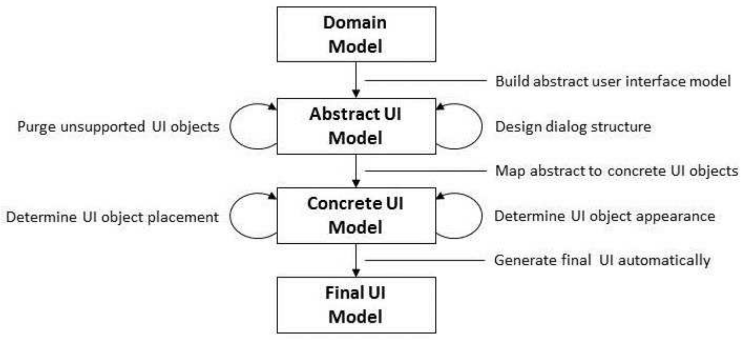

Domain Model
Build abstract user interface model
Abstract UI Model
Purge unsupported UI objects
Design dialog structure
Map abstract to concrete UI objects
Concrete UI Model
Determine UI object placement
Determine UI object appearance
Generate final UI automatically
Final UI Model

Figure 3. Overview of PaMGIS model transformations.

The utilization of default values within the respective model allows executing UI prototypes on the basis of not yet finalized models. In addition, the use of a runtime engine also allows for implementing model-based responsive designs and runtime adaptive behavior of the user interface.

Pattern-based Aspects

Within our combined development approach, patterns are used as means to alleviate the complexity of the model-driven processing. The patterns provide pre-assembled building blocks which can be used for domain and UI model construction. In addition, certain patterns provide valuable input to the various model transformation steps shown in Figure 3.

For this purpose it is essential to specify the patterns in a uniform and machine-readable manner and equip them with the required information. Further, it must be possible to compose pattern languages, i.e., to define the interrelationships between the patterns.

Hence, we developed the PaMGIS Pattern Description Language (PPSL) which is suitable to fulfil the aforementioned requirements.

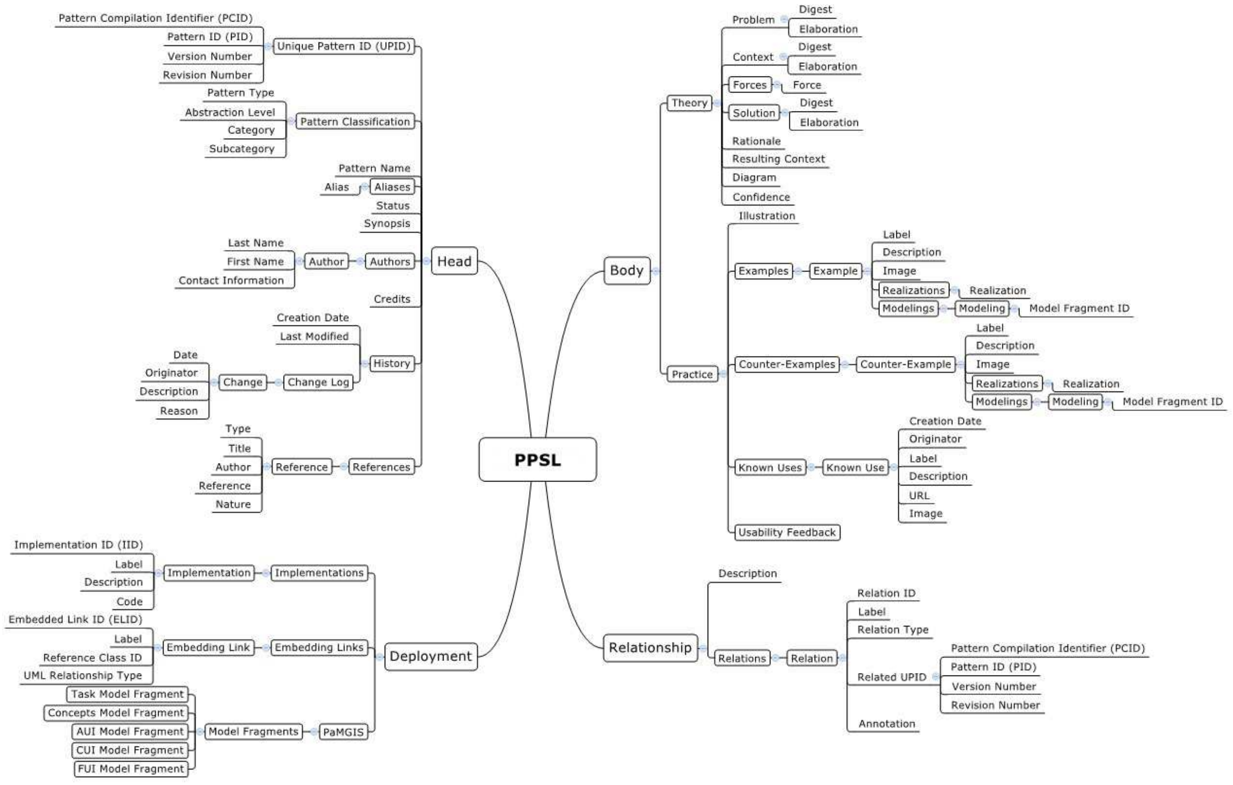


Figure 4. Overview of the PaMGIS Pattern Specification Language.

We reviewed existing pattern description standardization approaches as well as pattern tools in order to define PPSL in a way that patterns which are specified in the related formats can be transformed to PPSL. Thus, the entirety of all PLML 1.1 description elements is covered in PPSL, where required in a restructured or modified form. The only exception is the PLML element *Evidence* which is not directly included, but whose two sub-elements *Example* and *Rationale* are part of PPSL. Further, the PLML description element *Literature* can be mapped to *References* and *Related-Patterns* to *Relations*.

In addition, we introduced new description attributes for storing the supplementary information required by PaMGIS. An overview of the description elements of PPSL is provided in Figure 4. Pattern specifications are organized in four top level elements, i.e., *Head, Body, Relationships*, and *Deployment*.

The *Head* element incorporates metadata such as unique pattern identification, pattern classification, pattern name and aliases, information about pattern authors, credits, pattern evolution, and references to further sources and literature. The *Body* element is split into the two sub-elements *Theory* and *Practice*. The former provides theoretical background, including – amongst others – descriptions of the underlying problem, the context in which the pattern can be applied, and the proposed solution of the given problem. The latter demonstrates how the pattern was applied in practice by means of illustrations, examples, and counter-examples. The *Relationships* element serves as resource for the specification of the relationships between the various patterns and therefore allows the construction of pattern languages. Finally, the *Deployment* element contains – amongst others – model fragments of different types and abstraction levels as usability feedback. The model fragments are used as building blocks for the domain and the diverse UI models.

The model fragments are stored within the *Deployment/PaMGIS/ModelFragments* element and provide ready-to-use modelings of the pattern's inherent solution. During the process of constructing the domain model, the framework user can search, select, and apply patterns, i.e., automatically insert the respective task and concept model fragments into the domain model. It is also possible to store prefabricated AUI, CUI, or FUI model fragments with the pattern which can be directly embedded into the UI models of the corresponding abstraction levels. While patterns typically contain only one task and one concept model fragment, they might possess multiple AUI, CUI, and FUI model fragments for different contexts of use. This allows both applying different UI design solutions during design time and even during runtime, i.e., substituting one model fragment by another one. This mechanism is not limited to model fragments of the selfsame pattern. In fact, it is even possible to substitute whole patterns by alternative ones.

Regarding the process of finding appropriate patterns the framework offers multiple methods: pattern browsing, keyword search, free text search, exploiting pattern relations, or evaluating formal context descriptions which are stored as logical expressions within the *Body/Theory/Context/Digest* element.

Usability Evaluation Aspects

Running user interfaces – either on the basis of a complete FUI or in form of a prototype based on more abstract UI models – can be evaluated in terms of their usability and user experience using pertinent techniques and methods. The evaluation itself is not in the scope of PaMGIS. Hence, the framework does not offer any support for evaluation preparation, execution, and post-processing. But it is possible to document relevant insights within the system. Since the origin of model elements is captured inside the PaMGIS domain model and the various UI models, it is possible to locate the respective pattern and post the evaluation results to the pattern definition. For this purpose we introduced the pattern description element named *Body/Practice/UsabilityFeedback*.

A second, more automated option is to specify and utilize special usability evaluation (UE) patterns. They can be integrated in the domain model where they add some measuring instrumentation. For instance, the *Textual User Usability Feedback Dialog* pattern ensures, that an appropriate dialog is available allowing the user to record and send his or her opinion about certain aspects of the user interface at hand back to the PaMGIS framework. In the simplest case, this dialog is composed at least of an *Output* object providing some textual explanations for the user, an *Editor* object for the acquisition of the actual textual user feedback, and two *Activators* for either submitting the feedback or canceling the action. The aforementioned pattern includes the required task and concept model fragments as well as AUI and optionally less abstract UI model fragments. In this sense, the underlying domain-specific pattern language can be enriched by such UE patterns in order to capture usability feedback. At least in the case that the user interface is executed by means of the runtime environment, it is possible to automatically attach the user feedback directly to the respective pattern. Otherwise the information can be temporarily stored in a log file outside the scope of PaMGIS and fed back manually or in a semi-automatic way at a later point in time.

The collected usability feedback can be used to improve the quality of the patterns, the diverse models, and therefore of the final user interface.

Functional Framework Architecture

The PaMGIS framework consists of several logical function units, each supporting the various users in different fields of activities. An overview of the functional framework architecture is provided within Figure 5.

The core components are the two repositories, the *Pattern Repository* for storing the pattern specifications and the *Model Repository* to accommodate the diverse models as shown in Figure 2.

Access control is managed by means of the *User Database* which is administered via the *Framework User Administration* component. PaMGIS distinguishes several general types of users, i.e., unregistered users, registered users, pattern authors, power users, and administrators. Unregistered users are allowed to access a restricted part of the pattern specifications solely in read-only mode. In addition, they may register themselves to the framework. Registered users gain more insight into pattern details, have very limited write permissions, and may use certain collaboration functions, such as sending messages to pattern authors. The pattern authors have full access to the pattern descriptions and may create new patterns and modify existing ones. Power users can read entire pattern specifications and are allowed to make copies to which they have full read and write access. In addition, they can use and control the model-driven part of the framework. Finally, administrators take over the responsibility of managing the framework, e.g. creating, modifying, and deleting users, granting and withdrawing access rights, and maintaining the PaMGIS meta-models via the *Pattern Meta Model Administration* and the *Model Meta Model Administration* components.

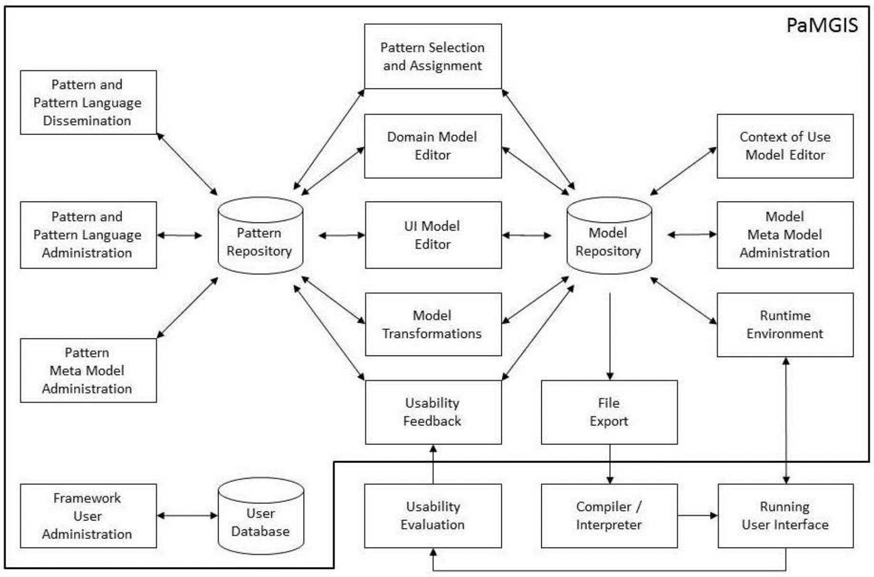

PaMGIS
Pattern Selection and Assignment
Pattern and Pattern Language Dissemination
Domain Model Editor
Context of Use Model Editor
Pattern and Pattern Language Administration
Pattern Repository
UI Model Editor
Model Repository
Model Meta Model Administration
Model Transformations
Runtime Environment
Pattern Meta Model Administration
Usability Feedback
File Export
Framework User Administration
User Database
Usability Evaluation
Compiler / Interpreter
Running User Interface

Figure 5. Overview of the functional PaMGIS architecture.

On the one hand, the *Pattern and Pattern Language Administration* unit supports pattern authors in creating and modifying patterns. On the other hand, power users can copy particular patterns to a private workspace where they can modify them according to their needs and build up pattern languages by specifying interrelationships between patterns.

The *Pattern and Pattern Language Dissemination* tool can be used by unregistered users to browse, search, and display certain aspects of the pattern descriptions which are released for this purpose. Additionally, it allows registered users to view more pattern details, send feedback and comments to pattern authors, and attach information about existing implementations to the pattern specifications. For this purpose we introduced the *Body/Practice/KnownUses* description element.

The *Domain Model Editor*, *Context-of-Use Model Editor*, and *UI Model Editor* allow power users to create and modify the respective PaMGIS models manually. In contrast, the *Pattern Selection and Assignment* component helps power users to search and find adequate patterns which can be selected and applied, i.e., insert the attached model fragments automatically into the domain, context-of-use, and/or different UI models. The *Model Transformations* unit supports the execution of the model transformations summarized in Figure 3 and can be configured to a certain extent.

The *Runtime Environment* is a means to execute user interfaces in the form of final UIs or prototypes as described above. The *File Export* component is used for exporting models in the form of text files for further external processing or documentation purposes.

Finally, the *Usability Feedback* unit offers support regarding the import of usability evaluation results into the framework and write it back to the respective patterns and/or models.

CONCLUSION

In this paper we presented our concerted pattern-based and model-driven approach for the development of interactive ubiquitous systems and provided an overview of the functional architecture of the related PaMGIS framework.

We strongly believe that the mélange of model- and pattern-related methods and techniques has the potential to alleviate weaknesses of the individual approaches and can create benefits in terms of reducing complexity and realizing reuse of already existing design knowledge.

The implementation of the framework is indeed work in progress, but major components already exist at least in prototypical form. Many patterns and several pattern languages have been developed, amongst others a pattern language for the domain of public transportation ticket selling.

The framework is a cornerstone for our further research on the potentials and limits of automated UI development. Moreover, we will intensify our work on supporting wearable computers with the PaMGIS framework.

REFERENCES

1. C. Alexander, S. Ishikawa, and M. Silverstein. 1977. A Pattern Language.Oxford University Press.
2. G. Calvary, J. Coutaz, L. Bouillon, M. Florins, Q. Limbourg, L. Marucci, F. Paternò, C. Santoro, N. Souchon, D. Thevenin, and J. Vanderdonckt. 2002. The CAMELEON Reference Framework. Retrieved April 15, 2015 from http://giove.isti.cnr.it/projects/cameleon/ pdf/CAMELEON%20D1.1RefFramework.pdf.
3. Paulo Pinheiro da Silva. 2001. User Interface Declarative Models and Development Environments: A Survey. In *DSV-IS'00 Proceedings of the 7th International Conference on Design, Specification, and Verification of Interactive Systems*, 207-226.
4. J. Engel, C. Herdin, and C. Märtin. 2012. Exploiting HCI Pattern Collections for User Interface Generation. In *Proceedings of PATTERNS 2012*, 36-44.
5. J. Engel, C. Herdin, and C. Märtin. 2014. Evaluation of Model-based User Interface Development Approaches. In *Proceedings of HCII 2014*. 295-307.
6. J. Engel and C. Märtin. 2009. PaMGIS: A Framework for Pattern-based Modeling and Generation of Interactive Systems. In *Proceedings of HCI International '09*. San Diego, USA, 826-835.
7. S. Fincher and J. Finlay. 2003. Perspectives on HCI Patterns: Concepts and Tools (Introducing PLML). *Interfaces, Vol. 56,* 26-28.
8. G. Meixner, G. Calvary, and J. Coutaz. 2014. Introduction to Model-Based User Interfaces. *W3C Working Group Note 07 January 2014*. Retrieved May 27, 2015 from http://www.w3.org/TR/mbui-intero/.
9. Brad A. Myers. 1992. State of the Art in User Interface Software Tools. *Advances in Human-Computer Interaction*, Vol. 4, Ablex Publishing.
10. F. Paternò. 2000. The ConcurTaskTrees Notation. In *Model-Based Design and Evaluation of Interactive Applications*, Springer Berlin Heidelberg, 39-66.
11. A. Pleuss, B. Hauptmann, D. Dhungana, and G. Botterweck. 2012. User Interface Engineering for Software Product Lines: The Dilemma Between Automation and Usability. In *Proceedings of the 4th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*. Copenhagen, Denmark,. 25-34.
12. Egbert Schlungbaum. 1996. Model-based User Interface Software Tools - Current State of Declarative Models. *GVU TECH REPORT*. Graphics, Visualization and Usability Centre, Georgia Institute of Technology.
13. E. Schlungbaum and T. Elwert. 1996. Dialogue Graphs: A Formal and Visual Specification Technique for Dialogue Modelling. In *Proceedings of the 1996 BCS-FACS Conference on Formal Aspects of the Human Computer Interface FAC-FA'96*, Sheffield, UK.
14. A. Seffah. 2010. The evolution of design patterns in HCI: from pattern langauges to pattern-oriented design. In *Proceedings of the 1st Interational Workshop on Pattern-Driven Engineering of Interactive Computing Systems* (PEICS'10), 4-9.

Model-driven UI Development integrating HCI Patterns

Enes Yigitbas
University of Paderborn
s-lab – Software Quality Lab
eyigitbas@s-lab.upb.de

Bastian Mohrmann
University of Paderborn
basti86@mail.upb.de

Stefan Sauer
University of Paderborn
s-lab – Software Quality Lab
sauer@s-lab.upb.de

ABSTRACT
An important criterion for user acceptance of interactive systems is software ergonomics. Therefore, a variety of HCI or usability patterns has been defined in the past. Although HCI patterns promise reusable best-practice solutions, the lack of formalization and effective tool support hinder their usage in a model-driven development process. To overcome this deficit, we propose a model-driven user interface development (MDUID) process that integrates HCI patterns. For showing the feasibility of our approach, we formalized and implemented a set of GUI patterns, a particular category of HCI patterns, based on IFML. We present our pattern application concept and our tool-support based on a customized MDUID process for generating rich internet applications (RIAs).

Author Keywords
HCI, Model-driven UI Development, Pattern, GUI Pattern

ACM Classification Keywords
H.5: Information interfaces and presentation (e.g., HCI): H.5.2: User Interfaces.

INTRODUCTION
An important criterion for user acceptance and user experience, particularly in the context of interactive systems, is software ergonomics. Therefore, a variety of HCI and usability patterns has been defined in the past [1]. Similar to software development patterns, HCI patterns are reusable best-practice solutions. The difference is that HCI patterns address the usability domain and the improvement of software ergonomics rather than general software architecture or code structure. One particular category of HCI patterns are GUI patterns. In [2] GUI patterns are described as patterns that "specify one or more abstract interaction objects, their relationships, and their interactive behavior" and that these patterns "are primarily aimed at good usability". The integration of GUI patterns in the MDUID process appears to be a promising way to overcome the lack of usability of automatically generated user interfaces. However, this solution entails two problems.

The first is that HCI patterns are mostly described informally in practice (1). However, model-driven approaches are based on formalisms like MOF meta-models or XML schemes. These formalisms are needed for automatized model-to-model and model-to-code transformations. The second problem is that there is barely no tool support for applying or instantiating HCI patterns, particularly GUI patterns in practice (2). In [3] it is reasoned that the lack of tools "hinders the use of HCI patterns within fully automated processes", like the MDUID approach.

In this work, we design and implement a customized MDUID process that integrates GUI patterns. The remainder of this paper is structured as following: First, we describe related work in the area of MDUID and HCI pattern integration approaches. Then we present our GUI pattern catalog and its formalization based on the abstract user interface language IFML. Afterwards we explain the implementation of our approach and the corresponding tool-support. In the end, we conclude our own contributions and outline future research activities.

RELATED WORK
Focusing on the topic of model-driven UI development (MDUID) integrating HCI patterns, multiple aspects have to be taken into account. Therefore our work is related to and influenced by a broad range of research fields in order to overcome the gap between HCI and MDUID. In the following we will briefly sum up existing MDUID approaches and pattern integration approaches and set them in relation to our own solution.

MDUID Approaches
MDUID brings together two subareas of software development, which are model-driven development (MDD) and user interface development (UID). The core idea behind MDUID is to automatize the development process of UI development by making the models the primary artifact in the development process rather than application code. An MDUID process usually involves multiple UI models on different levels of abstractions that are stepwise transformed to the final user interfaces by model transformations.

The CAMELEON Reference Framework (CRF) [4] provides a unified reference framework for MDUID differentiating between the abstraction levels Task & Concept, Abstract User Interface (AUI), Concrete User Interface (CUI) and Final User Interface (FUI).

Workshop on Large-scale and model-based Interactive Systems: Approaches and Challenges, June 23 2015, Duisburg, Germany.

There are various state-of-the-art modeling languages for covering the different abstraction levels of the CRF. For example MARIA XML (Model-based lAnguage foR Interactive Applications) [5] and IFML (Interaction Flow Modeling Language) [6] provide both an AUI modeling language and a tool-support to create and edit AUI models. Based on these AUI models further transformations can be performed to transform them into platform-specific CUI models which eventually are needed for generating the final user interfaces (FUI). The described MDUID approaches enable the specification and also support the generation of UIs, but they do not offer explicit mechanisms for specifying HCI patterns like GUI patterns. Therefore the existing MDUID tools show a lack of pattern formalization, instantiation and tight integration in the development process.

Pattern Integration Approaches

Engel [7] presents the concept of the PaMGIS (Pattern-Based Modeling and Generation of Interactive Systems) framework for pattern-based modeling. The PaMGIS framework combines model-based and pattern-based approaches on different levels of abstraction. The core component of the framework is the pattern repository, a collection of ``different types of patterns and pattern languages". Within the repository, the patterns are described by the PPSL (PaMGIS Pattern Specification Language). Beside the definition of HCI patterns, their meaning, their idea etc., PPSL also provides means to define relations between pattern models and other models. Such relations contain information about the particular pattern, the related FUI, (hierarchical) relationships to other patterns and back links to other object-oriented models, e.g. an AUI or CUI model of the interactive system. This information is necessary for model-to-model and model-to-code transformations. However, the PaMGIS approach leaves two issues open. First, it does not become completely clear if the mentioned model-to-code transformation can be defined on the model level or has to be defined for each instance over and over again. Secondly, no concepts for data binding have been discussed in this approach.

Radeke [8] proposes in his work a pattern application framework that describes a general concept of how patterns can be integrated in model-based approaches. This framework relies on three phases. In the first phase the user selects the pattern from the pattern repository that he wants to apply. The pattern repository contains hierarchically structured patterns and sub-patterns defined in a common pattern language. The generic part of the pattern is instantiated in the pattern instantiation phase with regard to the context of use. The outcome is an instantiated pattern that can be integrated in the development process. Although this approach suggests an interesting pattern instantiation concept, it integrates HCI patterns in a model-based rather than model-driven way. We overcome this deficit in our approach through a tight integration of the formalized GUI patterns by representing them in automatic model transformations.

PATTERN INTEGRATION CONCEPT

In order to overcome the previously mentioned problems (1) and (2), a general concept for integrating patterns in MDUID was developed that aims at increasing the usability of generated user interfaces. The main goal of this concept is the automatized application of GUI patterns within a model-driven process. Therefore, the CRF was extended by *instantiation parameters* and *application conditions* of GUI patterns like depicted in figure 1. Let us start with a short explanation concerning these two terms.

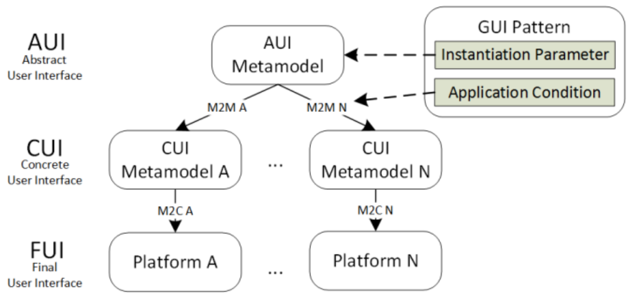


Figure 1. Overview of the pattern integration concept

Following the concepts of Wendler [12] and Radeke et al. [8], GUI patterns consist of a static and a dynamic part. The static part of a pattern describes the core solution idea of the pattern and can contain information about navigation, user interface elements or layout. It does not change among application scenarios. The dynamic part, however, depends on the prevailing pattern application context and therefore has to be set during the user interface modelling process. Since these dynamic parts determine the instantiation of a pattern, Wendler defines them as the instantiation parameters. The second important aspect is given by the conditions under which a pattern is advisable. In order to decide, when which pattern shall be applied, so-called pattern application conditions are helpful. Pattern application conditions are formal and describe situations in which a specific GUI pattern is reasonable. The advantage of formalised conditions is that they can be validated automatically, e.g. in the model-driven transformation process. Such a validation determines if a pattern is applied or not. After introducing the relevant terms, we will now explain the concept.

Referring again to figure 1, the pattern integration concept based on the CRF is depicted. It contains three abstract components: An *MDUID process implementation* with its different meta-models (AUI, CUI, Platform), an *instantiation parameter* extension for the AUI meta-model, and an *application condition* extension for the model-to-model transformation. These components have to be specified when the pattern integration concept is implemented. As explained above, instantiation parameters depend on a pattern's application context. Because of that, they have to be set during the initial user interface specification. In our case, the user interface is initially specified on the AUI layer and hence the instantiation parameters are integrated in the AUI meta-model by additional types and/or features. The application conditions are integrated in the transformation from the AUI to the CUI model by means of transformation rules. They are validated on the AUI model and therefore reusable for any target platform, like the AUI model itself. If the conditions are valid, the pattern is applied and the according platform-dependent CUI elements are generated.

GUI PATTERN CATALOG

The developed pattern integration concept was implemented for a choice of GUI patterns. Therefore, the abstract components introduced in the previous section were instantiated. The resulting customized MDUID process is depicted in figure 2. The AUI layer is realized with IFML and the model-to-model transformation is realized with an ATL [13] plugin. In order to integrate GUI patterns, a choice of GUI patterns was identified and then formalized by instantiation parameters and application conditions conforming to the extended components, the IFML meta-model and the ATL plugin. The formalized patterns are represented by the *extension* components in figure 2.

All integrated patterns were documented in a pattern catalog comprising the pattern's general meaning, its formalized instantiation parameters and application conditions. The formalisation of the instantiation parameters is described by means of an extension of IFML while the formalization of application conditions is described by means of transformation rules extending the ATL model-to-model transformation. Currently, the pattern catalog includes seven GUI patterns that were chosen based on their frequent use in interactive applications and their occurrence in pattern catalogs [1]. Further, the patterns in the pattern catalog are structured according to pattern categories taken from [9] and presented in a defined description scheme.

In the following, we want to present the *Wizard* pattern entry according to this description scheme in order to give an example of the pattern formalization:

Wizard

Description

The *Wizard* pattern is used when a user "wants to achieve a single goal but several decisions need to be made before the goal can be achieved completely" ([11]). Regarding a complex task inside a software system that is performed rather rarely and that is too long to fit into a single page, the Wizard pattern suggests to separate the complex task into several steps that are organized in a prescribed order. The user can deal with each of these steps in a discrete *mental space* and therefore has a simplified view on this task ([10] p.55).

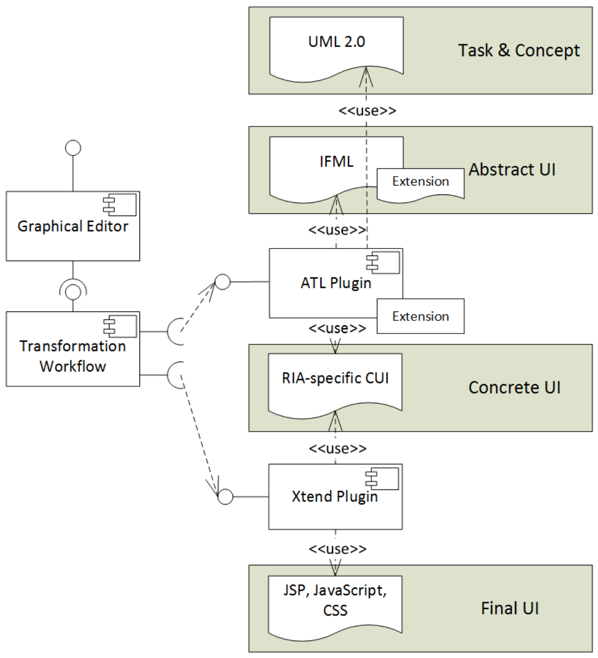


Figure 2. Architecture of the customized MDUID process

Instantiation Parameters

From the above description we can derive the following instantiation parameter when a task is separated into several decision steps: The *amount* of steps, the *order* of steps and the *content* of the particular steps. Like illustrated in figure 3, a step is formalised as a *Step* class that inherits from the *ViewContainer* class. Hence, the amount of steps and any view elements, like Events, Fields or Lists that are the content of a step can be defined. Furthermore, the inherited *outInteractionFlow* association enables the definition of *NavigationFlows* between steps and thus the order of the steps. In the related figure, the coloured classes are part of the IFML meta-model while the white class is a custom extension.

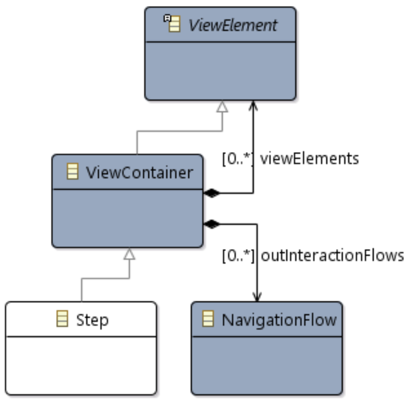


Figure 3. Simplified Wizard extension

Pattern Application Condition

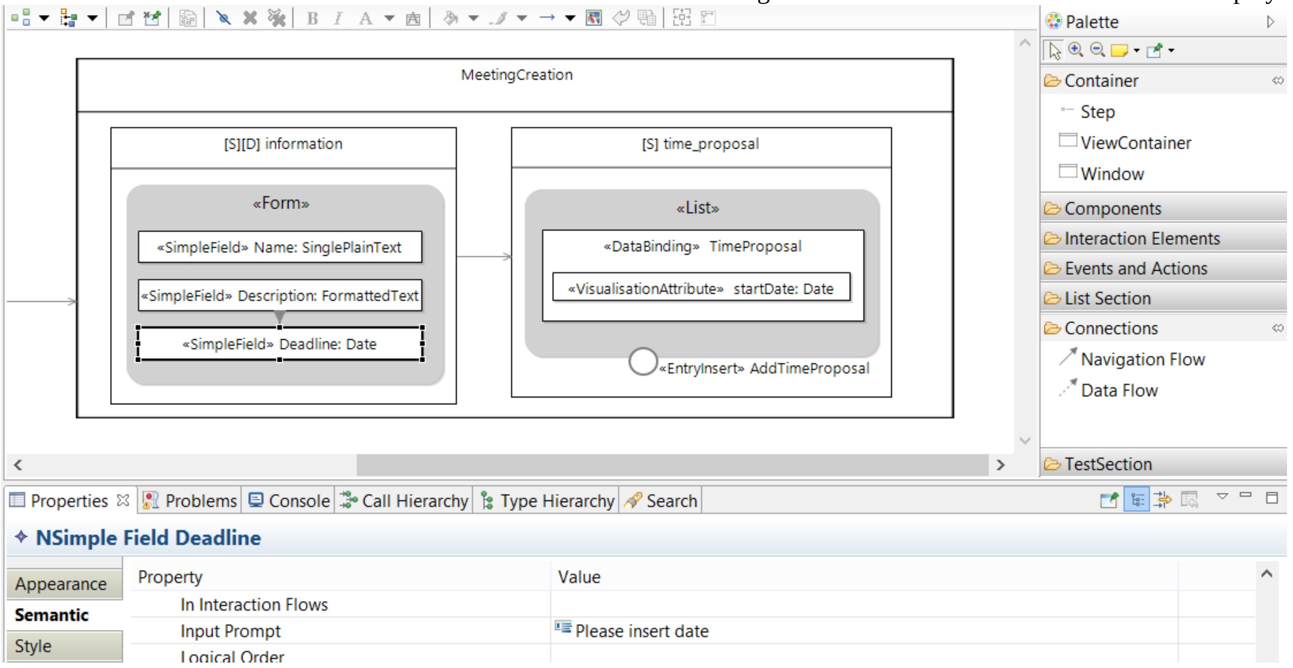
The Wizard pattern is applied whenever a *ViewContainer* element with at least two containing *Steps* is modelled. All contained *Steps* must be connected with *NavigationFlows*, so their order can be determined. Below, these conditions are implemented by means of an ATL transformation rule code snippet with a source pattern and a guard.

```
from
    s : AUI!ViewContainer(s.viewElements
            ->select(child | child.oclIsTypeOf(AUI!Step)
               and child.hasStepNavigationFlow())
            ->size() > 2)
```

IMPLEMENTATION AND TOOL-SUPPORT

In this section, the implementation of the pattern integration approach and the corresponding tool-support is presented in detail. The implementation is in a state where it already could be successfully applied in an industrial setting. The architecture of the implemented approach is depicted in figure 2. This architecture partially implements the four abstraction layers (Task & Concept, AUI, CUI, FUI) of CRF indicated by the colored rectangles. The UML 2.0 language on the Task & Concept layer enables the modeling of the application's domain, e.g. by a class diagram. As can be seen, the AUI layer is realized by IFML. In particular, we reused the *IFML-metamodel.ecore*, an implementation of the *IFML standard*, which can be downloaded from the official website and extended this meta-model by a choice of specific AUI elements and GUI pattern instantiation parameters. IFML provides dedicated extension points for this purpose. We realized the CUI layer with a custom meta-model, *RIACUI.ecore*, which is specific for rich internet applications. The *RIACUI.ecore* enables to describe user interface as they are perceived by the end user including the layout, colors and concrete interaction types. On the FUI layer, the user interface is finally represented by JavaServerPages, JavaScript code and CSS style sheets. The *Transformation Workflow* component manages the model-to-model and the model-to-code transformation. As can be seen in figure 2, the model-to-model transformation is realized with ATL and produces a RIA-specific CUI model from an IFML model and the related UML 2.0 domain model. ATL provides a feature called *rule inheritance*. Rule inheritance helps to reuse transformation rules and is similar to inheritance relations in the object oriented domain. Subsequently, the model-to-code transformation, realized in Xtend [16], generates application code from a previously produced RIA-specific CUI model. The advantage of Xtend is, since it is based on Java, a statically-typed programming language which employs template-based text generation. This is particularly helpful when it comes to code generation for application code organised in different files and programming languages as it is the case for the FUI of rich internet applications.

The tool support is given by a *graphical editor* that is an extension to the IFML open source editor based on EMF [15] and the *Sirius* [14] framework. The editor is available at Github and was extended within this work by graphical representations and create/read/update/delete operations for the IFML extensions. Figure 4 depicts a screenshot of the editor showing the working area, the palette and the properties tab. This editor is an *eclipse* plug-in [17]. In the working area the current IFML model is displayed,



NSimple Field Deadline

Property	Value
In Interaction Flows	
Input Prompt	Please insert date
Logical Order	

Figure 4. Screenshot of the extended IFML editor

represented in its concrete syntax. The use of meaningful icons and graphical representations helps for a better and faster understanding of the editor. The user can create new IFML model elements via Drag & Drop from the palette on the right hand side. The palette is structured in multiple sections where different V*iewElements* like *List*, *Window* and *NavigationFlow* are available. The *Step* entry in the palette also indicates that instantiation parameters of GUI patterns are configurable. The editing of IFML model elements mostly takes place in the properties tab located at the bottom. Here, all attributes and associations of model elements can be set, modified or deleted. Once such an IFML model is specified, it serves as the input of the transformation chain which can be triggered manually from the editor's context menu. The outcome is a RIA-specific CUI model in XML format and the FUI represented by multiple JavaServerPages, JavaScript files and CSS style sheets.

CONCLUSION AND OUTLOOK

In this paper, we presented the design and implementation of a customized MDUID process that integrates GUI patterns. As a basis of our solution concept we first described our general pattern integration concept. Then we presented our GUI pattern catalog and its formalization based on the abstract user interface language IFML. The feasibility of our approach was then shown by a tool-support which extends the existing IFML editor by integrated GUI patterns. The implementation of the customized MDUID process and the practical usage of the tool-support was shown in the context of generating rich internet applications (RIAs). With regard to future work we intend to evaluate our implemented solution in an industrial case study. In the evaluation we will especially focus on the influence of the integrated GUI patterns to the usability of the automatically generated RIAs.

REFERENCES

1. HCI Patterns. Retrieved April 2, 2015 from http://www.hcipatterns.org/patterns
2. Christian Märtin, Christian Herdin, and Jürgen Engel. 2013. Patterns and Models for Automated User Interface Construction – In *Search of the Missing Links, in: M. Kurosu (Ed.), Human-Computer Interaction, Part I, HCII 2013, LNCS 8004,* 401-410.
3. Kai Breiner, Marc Seissler, Gerrit Meixner, Peter Forbrig, Ahmed Seffah, and Kerstin Klöckner. 2010. PEICS: towards HCI patterns into engineering of interactive systems. In *Proc. of the 1st International Workshop on Pattern-Driven Engineering of Interactive Computing Systems* (PEICS '10).
4. Gaëlle Calvary, Joëlle Coutaz, David Thevenin, Quentin Limbourg, Laurent Bouillon, and Jean Vanderdonckt. 2003. A Unifying Reference Framework for Multi-target User Interfaces. In: *Interacting with Computers,* 289-308.
5. Fabio Paterno', Carmen Santoro, and Lucio Davide Spano. 2009. MARIA: A universal, declarative, multiple abstraction-level language for service-oriented applications in ubiquitous environments. *ACM Trans. Comput.-Hum. Interact.* 16, 4, Article 19 (November 2009), 30 pages.
6. IFML Spec. Retrieved April 2, 2015 from http://www.omg.org/spec/IFML/
7. Jürgen Engel. 2010. A model- and pattern-based approach for development of user interfaces of interactive systems. In *Proceedings of the 2nd ACM SIGCHI symposium on Engineering interactive computing systems* (EICS '10). ACM, New York, NY, USA, 337-340.
8. Frank Radeke and Peter Forbrig. 2007. Patterns in task-based modeling of user interfaces. In *Proceedings of the 6th international conference on Task models and diagrams for user interface design* (TAMODIA'07), Marco Winckler, Philippe Palanque, and Hilary Johnson (Eds.). Springer-Verlag, Berlin, Heidelberg, 184-197.
9. Marco Brambilla and Piero Fraternali. *Interaction Flow Modeling Language: Model-Driven UI Engineering of Web and Mobile Apps with IFML.* Morgan Kaufmann, 2014.
10. Jenifer Tidwell. *Designing interfaces – patterns for effective interaction design* (2. ed.). O'Reilly, 2011.
11. Matijn Van Welie. A pattern library for interaction design. Retrieved April 2, 2015 from http://www.welie.com/patterns/
12. Stefan Wendler, Danny Ammon, Ilka Philippow, and Detlef Streitferdt. A factor model capturing requirements for generative user interface patterns. In *PATTERNS 2013, the Fifth Int. Conf. on Pervasive Patterns and Applications, Valencia, Spain, IARIA*, Lecture Notes in Computer Science, pages 34–43, 2013.
13. ATL. Retrieved April 2, 2015 from https://eclipse.org/atl/
14. Sirius. Retrieved April 2, 2015 from https://eclipse.org/sirius/
15. EMF. Retrieved April 2, 2015 from https://www.eclipse.org/modeling/emf/
16. Xtend. Retrieved April 2, 2015 from https://eclipse.org/xtend/
17. Eclipse. Retrieved April 2, 2015 from https://eclipse.org/

UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR TELEMATIK

TECHNISCHE
UNIVERSITÄT
DRESDEN