# Automation in Model-based Usability Evaluation of Adaptive User Interfaces by Simulating User Interaction

vorgelegt von
Diplom-Informatiker
Michael Quade
geb. in Wolgast

von der Fakultät IV - Elektrotechnik und Informatik
der Technischen Universität Berlin
zur Erlangung des akademischen Grades

Doktor der Ingenieurwissenschaften
– Dr.Ing. –

genehmigte Dissertation

Promotionsausschuss:

| | |
|---|---|
| Vorsitzender: | Prof. Dr. Henning Sprekeler (TU Berlin) |
| Gutachter: | Prof. Dr. Dr. h.c. Sahin Albayrak (TU Berlin) |
| Gutachterin: | Prof. Dr. Gaëlle Calvary (IMAG, Frankreich) |
| Gutachter: | Prof. Dr. Sebastian Möller (TU Berlin) |

Tag der wissenschaftlichen Aussprache: 07. Dezember 2015

Berlin 2015

# Acknowledgments

First of all, I want to thank my advisor Prof. Albayrak for giving me the chance to work on my phd topic over the last years in the encouraging environment of the DAI-Labor at the TU Berlin. I would also like to thank the rest of my thesis committee: Prof. Calvary and Prof. Möller, for their comments and constructive feedback that helped me to finish this thesis in a much better way.

This work would not have been possible without the help of my fellow colleagues with whom I spent many hours of discussing, planning and implementing. Without a specific order I want to thank Marco Blumendorf, Grzegorz Lehmann, Frank Trollmann, Mathias Runge, Dirk Roscher, Klaus-Peter Engelbrecht, Marc Halbrügge, Stefan Hillmann, Aaron Ruß, Peter Steinnökel, Timo Hanisch, Maximilian Kern, Carsten Wirth and Andreas Rieger. Each of you had their share in the outcome of this thesis. A big thank you also goes to the rest of the colleagues from the DAI-Labor. Working with you was highly motivating and encouraging.

My time of writing was accompanied by working on several interesting projects funded by Deutsche Telekom, German ministries and the DFG. Without their trust in our research I would not have been be able to conduct my work and write this thesis.

Of course, all of this would not have been possible without the extensive support of my wife and family. You, most of all, shared the sacrifices of this endeavor. Looking back it is hard to imagine that within the time of writing this thesis I got married and we got two wonderful children. You always give me the power to look forward and move on.

*Berlin, December 2015*

# Abstract

The goal of adaptive user interfaces (UI) is offering the opportunity to adapt to changes in the context of use and thus provide potentially improved interaction capabilities for different users in specific situations. But, this poses the challenge of evaluating usability aspects of many different variants of the resulting UI. Consequently, usability evaluations with real users or experts tend to become complex and time-consuming especially in the domain of adaptive UIs. Model-based usability evaluations and specifically automated tools and approaches have proven to correctly predict usability relevant aspects in early stages of development. However, the creation and provision of required models and information tends to be complex and time consuming as well and further requires a high degree of expertise for the specific tool and applied method.

This thesis describes an integrated approach that provides automation in model-based usability evaluation based on already existing development models of adaptive UIs. The approach is based on required information for describing the UI surface information and the interaction capabilities of the UI. With the help of this information usability relevant criteria are predicted using specific tools of automated usability evaluation.

The implementation of the approach presents integration of an existing runtime framework for adaptive UIs with a cognitive user behavior model for simulation. Information required for simulating interactions is created automatically with the help of the UI development models and by this means saves time and costs when preparing and running simulations. Additionally, with the help of two studies, the resulting predictions are further improved by directly using information encoded in the existing development models without requiring specific expertise from designers and usability experts.

# Zusammenfassung

Adaptive Nutzerschnittstellen sind in der Lage sich an den Umgebungskontext anzupassen, um auf diese Weise für verschiedene Anwender in der jeweiligen Situation besser bedienbar zu sein. Dies führt jedoch dazu, dass bei Evaluationen der Usability zur Bestimmung der Bedienbarkeit und Gebrauchstauglichkeit viele verschiedene Varianten der Nutzerschnittstellen betrachtet werden müssen. Aus diesem Grund sind gerade Evaluationen mit echten Anwendern und Experten mit hohen Kosten und einer hohen Komplexität verbunden. Modellbasierte Evaluationen und automatisierte Verfahren sind Ansätze, die sich in der Praxis bereits zu frühen Zeitpunkten der Entwicklung bewährt haben. Jedoch sind auch diese Verfahren mit Kosten bei der Vorbereitung und Bereitstellung der Modelle und Informationen verbunden. Darüber hinaus erfordert deren korrekte Anwendung zumeist einen hohen Grad an Expertise und Erfahrung mit dieser spezifischen Evaluationsmethode.

In dieser Arbeit wird ein integrierter Ansatz beschrieben, der Automatisierung bei modellbasierten Evaluationen mit Hilfe von bereits existierenden Entwicklungsmodellen der adaptiven Nutzerschnittstellen beschreibt. Der Ansatz basiert auf notwendigen Informationen zur Beschreibung der Oberflächen und Interaktionsfähigkeiten der Nutzerschnittstellen. Mit Hilfe dieser Informationen und existierender Werkzeuge zur automatischen Usability Evaluation werden im Anschluss Vorhersagen getroffen, die für die Bestimmung der Usability während der Entwicklungszeit dienen.

Die Implementierung des Ansatzes beschreibt die Integration eines existierenden Frameworks für modellbasierte adaptive Nutzerschnittstellen und einem kognitiven Nutzerverhaltensmodell, das für Simulationen genutzt wird. Dafür notwendige Informationen werden automatisch aus den Entwicklungsmodellen erzeugt, um auf diese Weise eine Zeit- und Kostenersparnis bei der Vorbereitung und Durchführung der Simulationen zu erzielen. Darüber hinaus wird mit Hilfe zweier Studien aufgezeigt, wie die Vorhersagen auf Basis von Informationen aus den Entwicklungsmodellen automatisch weiter verbessert werden können, ohne, dass hierfür notwendige menschliche Expertise zu Rate gezogen werden muss. Auf diese Weise wird neben der Ersparnis an Zeit auch eine erleichterte Anwendung ermöglicht.

# Contents

# List of Figures

# List of Tables

# Abbreviations

| | |
|---|---|
| **ACT** | Adaptive Control of Thought |
| **ACT-R** | Adaptive Control of Thought-Rational |
| **AUE** | Automated Usability Evaluation |
| **AUI** | Abstract User Interface |
| **COMET** | Context Mouldable Widget |
| **CTT** | Concurrent Task Tree |
| **CUI** | Concrete User Interface |
| **DIN** | Deutsches Institut für Normung |
| **DMP-UI** | Distributed Migratable Plastic User Interfaces |
| **GLEAN** | GOMS Language Evaluation and Analysis |
| **EPIC** | Executive Process-Interactive Control |
| **ETS** | Enabled Task Set |
| **FUI** | Final User Interface |
| **GOMS** | Goals Operators Methods Selection Rules |
| **GOMSL** | GOMS Language |
| **GUI** | Graphical User Interface |
| **GUMO** | General User Model Ontology |
| **GUMS** | General User Modeling System |
| **HCI** | Human Computer Interaction |
| **ISO** | International Organization for Standardization |
| **KLM** | Keystroke-Level Model |
| **LC** | Logical Consistency |
| **LM** | Logical Model |
| **MASP** | Multi Access Service Platform |
| **MDE** | Model-Driven Engineering |
| **MEMO** | Mental Models (Workbench) |
| **MHP** | Model Human Processor |
| **MTTE** | MASP Task Tree Editor |
| **NGOMSL** | Natural GOMS Language |

| | |
|---|---|
| **OWL** | Web Ontology Language |
| **PM** | Physical Model |
| **PTS** | Presentation Task Set |
| **PUM** | Programmable User Model |
| **SNIF-ACT** | Scent-based Navigation and Information Foraging-ACT |
| **SOAR** | State, Operator Apply Result |
| **SIM** | System Interaction Model |
| **STM** | System Task Model |
| **UI** | User Interface |
| **UIDE** | User Interface Design Environment |
| **UIM** | User Interaction Model |
| **UTK** | User Task Knowledge |
| **UTM** | User Task Model |
| **UIRS** | User Interface Runtime System |
| **UX** | User Experience |
| **VA** | Visual Attention |
| **VUM** | Virtual User Model |
| **XML** | Extensible Markup Language |

# List of Publications

The work described in this thesis builds on the following research papers by the author or with contributions from the author:

**Ch. 3 - Required Information for Automated Usability Evaluation**

**Quade, M.** ; Rieger, A. ; Albayrak, S.: Requirements for Applying Simulation-Based Automated Usability Evaluation to Model-Based Adaptive User Interfaces for Smart Environments. *In: Streitz, N. (E.) ; Stephanidis, C. (Ed.): DAPI/HCII 2013, LNCS 8028*, pp. 235-244, 2013.

**Ch. 5 - Approach**

**Quade, M.** ; Lehmann, G. ; Engelbrecht, K.-P. ; Roscher, D. ; Albayrak, S.: Automated Usability Evaluation of Model-Based Adaptive User Interfaces for Users with Special and Specific Needs by Simulating User Interaction. *In: Martín, E. (Ed.) ; Haya, P. A. (Ed.) ; Carro, R. M. (Ed.): User Modeling and Adaptation for Daily Routines.* Springer London, 2013 (Human-Computer Interaction Series). - ISBN 978-1-4471-4777-0, pp. 219-247.

**Quade, M.**: Model-based Evaluation of Adaptive User Interfaces. *In: Wichert, R. (Ed.) ; Van Laerhoven, K. (Ed.) ; Gelissen, J. (Ed.): Constructing Ambient Intelligence Bd. 277.* Springer Berlin Heidelberg, 2012. - ISBN 978-3-642-31478-0, pp. 318-322.

**Quade, M.** ; Blumendorf, M. ; Lehmann, G. ; Roscher, D. ; Albayrak, S.: Evaluating User Interface Adaptations at Runtime by Simulating User Interaction. *In: 25th BCS Conference on Human Computer Interaction - HCI2011,* 2011.

**Ch. 6 - Implementation and Ch. 7. Evaluation**

**Quade, M.** ; Halbrügge, M. ; Engelbrecht, K.-P. ; Albayrak, S. ; Möller, S.: Predicting Task Execution Times by Deriving Enhanced

Cognitive Models from User Interface Development Mode *In: Proceedings of the 2014 ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, New York, NY, USA : ACM, 2014 (EICS '14). - ISBN 978-1-4503-2725-1, 139-148.

Steinnökel, P. ; Scheel, C. ; **Quade, M.** ; Albayrak, S.: Towards An Enhanced Semantic Approach For Automatic Usability Evaluation. *In: Proceedings of the Computational Linguistics-Applications Conference, 2011.* - ISBN 978-83-60810-47-7, pp. 85-91.

**Ch. 8 - Conclusion**

Halbrügge, M. ; **Quade, M.** ; Engelbrecht, K.-P.: A Predictive Model of Human Error based on User Interface Development Models and a Cognitive Architecture *In: Taatgen, N. A. (Ed.) ; Vugt, M. K. (Ed.) ; Borst, J. P. (Ed.) ; Mehlhorn, K. (Ed.): Proceedings of the 13th International Conference on Cognitive Modeling*, Groningen, the Netherlands: University of Groningen, - ISBN 978-90-367-7763-6, pp. 238-243.

Ruß, A. ; **Quade, M.** ; Kruppa, M. ; Runge, M.: Rule-Based Approach for Simulating Age-Related Usability Problems. *In: Wichert, R. (Ed.); Eberhardt, B. (Ed.) ; VDE (Org.): Ambient Assisted Living Bd. 5.* AAL-Kongress 2012. Berlin, Germany : Springer, January 25-25 2012 (Advanced Technologies and Societal Change). - ISBN 978-3-642-27490-9, pp. 149-166.

Further research papers describing preceding work and work outside the scope of this thesis.

Halbrügge, M. ; **Quade, M.** ; Engelbrecht, K.-P.: How Can Cognitive Modeling Benefit from Ontologies? Evidence from the HCI Domain. *In: Bieger, J. (Ed.) ; Goertzel, B. (Ed.) ; Potapov, A. (Ed.): Artificial General Intelligence Bd. 9205*, - ISBN 978-3-319-21364-4, 261-271.

Engelbrecht, K.-P. ; **Quade, M.** ; Möller, S.: Analysis of a New Simulation Approach to Dialog System Evaluation. *In: Speech Communication 51 (2009), Nr. 12,* pp. 1234-1252. - ISSN 0167-6393.

Feuerstack, S. ; Blumendorf, M. ; Kern, M. ; Kruppa, M. ; **Quade, M.** ; Runge, M. ; Albayrak, S.: Automated Usability Evaluation during Model-Based Interactive System Development. *In: HCSE-TAMODIA '08: Proceedings of the 2nd Conference on Human-Centered Software Engineering and 7th International Workshop on Task Models and Diagrams.* Berlin, Heidelberg : Springer-Verlag, 2008. - ISBN 978-3-540-85991-8, pp. 134-141.

Engelbrecht, K.-P. ; Kruppa, M. ; Möller, S. ; **Quade, M.**: MeMo Workbench for Semi-Automated Usability Testing. *In: Proceedings of Interspeech 2008 incorporating SST 2008.* Brisbane, Australia: International Symposium on Computer Architecture (ISCA), September 2008. - ISSN 1990-9772, 1662-1665.

# 1. Introduction

One of the main challenges of human-computer interaction (HCI) is the translation process between human users on the one side and software applications on the other side. During this process the focus of good user interface (UI) design lies in bridging the *gulf of execution* and the *gulf of evaluation* (Norman, 1988). On the one hand, the gulf of execution describes the fact that users need to convert own goals into actions that can be performed with the application. On the other hand, the gulf of evaluation describes the fact that users also need to decide if the performed action moved the interaction process towards the intended goal. Both gulfs can be narrowed by applying UI design principles to the development process. This involves knowledge about human interaction capabilities; e.g. well-known metaphors or concepts of visual search and recognition of UI elements. Such knowledge helps designers in choosing amongst different versions of UI mock-ups. Furthermore, testing applications for usability within their intended field of use and with their intended users is an important task of good UI design and engineering. However, evaluations with real users can become complex tasks, which require high efforts in time, costs and expertise, but pay off by saving costs in the end (Nielsen, 1993b).

## 1.1. Research Context

With the constant increase of computation power, software applications and their potential usage for a vast amount of different activities are becoming more and more complex. Former stand-alone computer systems have been evolving into everyday appliances and are becoming parts of our daily lives, as envisioned by Weiser (1991) more than 20 years ago. These ubiquitous spaces are commonly described as smart environments and are characterized by lots of different networked interaction devices and applications with the goal of supporting users in manifold situations. Smart environments offer opportunities for providing assistance to users within their daily routines whether at work or at home; e.g. in the domain of Ambient Assisted Living (Aarts et al., 2002). However,

poorly designed devices and improper UIs of the applications can lead to frustration and irritation as well - especially in this domain.

Many users do not just expect to be able to use applications for reaching their current goals, but to be assisted proactively depending on their current situation. For this purpose, all UIs need to present required information properly and tailored to the current user's needs and (dis-)abilities. This challenge becomes even more demanding as these UIs can also be distributed over different devices with different interaction capabilities; e.g. smart phones, smart TVs and tablet devices. The complexity of using all these different devices and services tends to make high cognitive demands on users; e.g. by following different interaction strategies limited by the device's constraints. Especially this ongoing trend of ubiquitous computing has been leading to a shift in application purpose from mere working tool to ubiquitous interactive assistant in many areas of life.

The majority of these newly arising demands are currently approached by the development of *adaptive UIs* that aim at providing UIs that are able to adapt to users instead of requiring users to adapt to the UI. This can also be seen as a shift in how UI engineering needs to be conducted. Further knowledge on how to adapt to users in an assisting way can be gathered by integrating information about users and by enhancing new sensing possibilities from the surroundings of smart environments. Besides adapting the layout of the UI, the displayed content can also be altered depending on available information about users, their preferences and interaction history. Examples for these adaptations range from lots of different location-based services; e.g. when using smart phones, up to product recommendations when browsing in web stores.

## 1.2. Research Problem

In general, adaptations to UIs are performed on the basis of information about the *context of use*, which is summed up as information about the current user, the platform and the environment (Calvary et al., 2003). However, designers might not always be able to judge if an adaptation leads to a "usable" UI for each potential user within each situation defined by the context of use. This high complexity of the context of use and resulting adaptations to the UI directly lead to problems in fully evaluating adaptive UIs. Hence, all possible (side-)effects of adaptations to the UI and underlying *tasks* of the application can therefore hardly be evaluated with real users for each adaptation variant under the necessary conditions. Even though this would usually provide the best evaluation results, the involved costs and time simply become limiting factors in

common software development cycles. As a result from this trend to adaptive UIs, there arises an emerging need for dynamic evaluation and measurement methods for various accounts of usability and accessibility of adaptive UIs.

A main issue when evaluating usability is the formalization and understanding of interaction means and concepts. One way to address this, is integrating *UI models of the application* and *models of the user* into the development process, as proposed by model-driven engineering (MDE). On the one hand, UI models of the application (Vanderdonckt et al., 2009) are able to formalize the design, express the underlying concepts and make them interpretable by machines. On the other hand, user models (Kobsa, 2001) are commonly used for describing users' physical and cognitive abilities and for formalizing different groups of users based on these attributes. Above all, the interconnection of both approaches can also be used for evaluating usability. For this purpose, *automated usability evaluation* (AUE) emerges as a paradigm allowing exhaustive and at the same time cheap testing of different usability criteria (Ivory and Hearst, 2001). While there are AUE methods that assist in capturing user interaction in laboratories, most promising approaches are based on *predictive analytical modeling* and *predictive simulation*. With the help of underlying psychological theories, concepts and models, these approaches have proven to correctly predict criteria relevant for judging usability; e.g. execution time predictions and learning time estimations.

Still, there exist main shortcomings of AUE which form *barriers* to the adoption by the interaction design industry and a more widespread use:

**I. Costly and time-consuming modeling process**

Current AUE approaches require additional descriptions of the user, UI and tasks in their specific notation. In most cases, such descriptions of the UI and the tasks do not exist or cannot be automatically derived from the final UI or the source code. Thus, the required models need to be provided by the evaluators themselves. This is a time-consuming and potentially error-prone task that needs to be performed repeatedly for each adaptation variant and task.

**II. Complexity of modeling process**

Although some AUE methods are powerful for specific evaluation purposes, they are hard to apply for complex tasks and more general usability evaluations. Especially when evaluating adaptive UIs many different specific models for evaluation need to be created, which leads to a state space explosion (problem). Furthermore, these methods are not widespread because they usually require highly skilled evaluators, who must put high effort in creating the required models.

To sum up, on the one hand model-based UI development faces the need to evaluate the usability of many different adaptation variants with reasonable costs. On the other hand current AUE methods lack wide-spread usage due to their complex and costly modeling processes especially for novice users.

## 1.3. Thesis Scope

Below, the scope of this thesis is described, which limits applicable approaches, targeted usability attributes and the time for applying the presented approach.

#### Simulation-based AUE

The scope of this thesis is set on simulation-based interaction between models of the user and the application for reasons of automation and assistance in early evaluations during the development process. Especially when evaluating different adaptations it is essential to provide a simulation-based AUE method to automate the interaction process. Such an approach provides a variety of different interaction data, while, at the same time, minimizing the effort involved. Consequently, designers of adaptive UIs profit from such an approach. They do not need to provide the interaction paths by hand for each possible adaptation of the UI. By this means the high effort that arises from the state space explosion problem of interaction paths is tackled. However, such an automated interaction process needs to remain open by still allowing the designer to provide a predefined interaction path. This is required in case a specific solution path needs to be evaluated in more detail. Thus, a hybrid simulation-based approach is preferred to provide a maximum of flexibility during evaluation. Using such a simulation-based approach specifically narrows the scope and constrains the set of applicable methods and tools.

#### Pragmatic Usability Attributes

Different usability evaluation methods are suitable for predicting and uncovering different types of usability attributes. Hassenzahl (2004) distinguishes these usability attributes into pragmatic and hedonic categories. While the latter are mainly related to e.g. novelty and beauty of a design; they can usually only be provided with the help of extensive qualitative user tests and questionnaires.

When using simulation-based approaches, it is hard to predict hedonic attributes, because AUE methods allow reasoning about human performance measurements mainly, which are more related to the category of pragmatic attributes. For this reason hedonic attributes fall out of scope when applying current AUE methods. Nevertheless, different quantitative and qualitative usability criteria can be applied when predicting pragmatic usability attributes; e.g. interaction execution time, number of required interaction steps and uncovering interaction errors by tracing the simulated path.

#### Evaluation Phase during Development Time

Development time refers to the time of developing an application; i.e. when designers perform an analysis and design phase and then develop the UI and perform an evaluation. Thus, development time explicitly includes evaluation of the prototypes and integrating information from these tests. The described approach and methods of this thesis are applied during the evaluation phase at development time and require access to information from the development models and specific tools for evaluation.

> ***The scope of this thesis narrows down to simulation-based AUE for automating the interaction process of a simulated user with adaptive UIs to decrease high effort that arises from the state space explosion problem. Predicted usability evaluation results from the category of pragmatic usability attributes serve to objectively judge and compare different UI variants during evaluations at development time.***

## 1.4. Thesis Statement and Contribution

In this thesis the described shortcomings of AUE for adaptive UIs are addressed by investigating how already existing UI models from the development process can be integrated into simulation-based AUE. The main benefit of this concept lies in directly using these existing UI models to bypass manually creating specific AUE models of the application and thereby saving modeling costs. Furthermore, evaluation results can be improved by directly using modeled information from development within the AUE and thus further support designers and lower the required expertise. Finally, this allows evaluating and comparing different development stages of UI prototypes and various adaptation variants with reasonable costs of time and effort.

Consequently, this thesis is based on the following **statement**:

***Using development models from adaptive UIs leads to improved evaluation results and a simplified modeling process for predictive automated usability evaluation methods based on simulated interaction.***

The main **contributions** of this thesis are:

- **Definition of required information** that is relevant for simulation-based AUE of adaptive UIs.
- **Concept and design of an environment** for a model-based automated usability evaluation of adaptive user interfaces with the help of UI development models.
- **Implementation of the environment** as a proof of concept.
- **Validation of concepts of the implemented environment** with the help of case studies to demonstrate saved costs and improved predictions based on UI development models.

Below, the structure of this thesis is explained.

## 1.5. Thesis Structure

This thesis is structured as follows:

**Chapter 1** introduces the background of this thesis and motivates the need for automated usability evaluations during model-based UI development of adaptive UIs.

**Chapter 2** briefly explains the fundamentals and basic concepts of usability evaluations and adaptive UIs that are used throughout this thesis.

**Chapter 3** describes required information for simulation-based AUE.

**Chapter 4** gives an overview of the state of the art in related work that is highlighted by Chapter 3.

**Chapter 5** describes the approach of this thesis by referring to Chapter 3.

**Chapter 6** explains the implementation of the approach and required extensions based on existing frameworks for model-based UI development and automated model-based usability evaluation.

**Chapter 7** presents the evaluation of the implemented approach with the help of an analysis of saved modeling time and effort as well as a case study presenting improved

predictions from two user studies. For this purpose a running example is illustrated that is referenced in appropriate parts of the thesis.

**Chapter 8** concludes this thesis by summing up the work and results as well as giving an outlook on future work based on the approach.

As outlined above, the next chapter describes the fundamentals that help understanding the goals of this thesis and the relevant terms that are used for explaining the conducted work.

# 2. Fundamentals

In this chapter basic concepts of HCI and for evaluating user interfaces are described. As this work addresses the *usability* of *adaptive UIs* these terms are explained with the help of widely accepted definitions.

Usability and usability evaluations are described in Section 2.1. Subsequently, the context in which an application can be used and how applications can determine this context are explained in Section 2.2. With the help of these concepts, in Section 2.3 the notion of adaptive UIs is presented by describing challenges and frameworks for the development of adaptive UIs. Finally, this chapter concludes with a short summary.

## 2.1. Usability in Human-Computer Interaction

As the research field is perceived quite diverse, there exists a multitude of different views, explanations and definitions for the concept of usability. In general, usability can be seen as the central core aspect of HCI dealing with ergonomics. It is influenced by knowledge from different research areas, mainly software engineering and product design as well as cognitive and labor psychology. Sometimes usability is simplified to the aspect *ease of use*, which, however, does not take into account the whole complexity; e.g. flexibility of the software application and human learning capabilities.

An early approach for describing usability refers to it as the discrepancy between the potential use of an application (consisting of hardware, software and technical support) and the extent to which a user is able to exploit this potential (Eason, 1984). Eason also states, that the higher this discrepancy is, the more likely the user will not be able to use parts of the application; i.e. the user will underutilize the application. While this early definition does not explicitly relate to the context of the interaction with the application, it does take into account the user, the task and the system. Together, they are combined as variables for measuring usability in a cost-benefit analysis. Further views on usability; e.g. by Nielsen (1993b); include *learnability*, *memorability*, *errors*, *efficiency* and *satisfaction*.

### 2.1.1. Usability according to DIN EN ISO 9241-110

Today, there are widely accepted specialized guidelines for usability from different companies and institutions such as the World Wide Web Consortium.[1] In an attempt to homogenize these different views and definitions of usability, the International Organization for Standardization (ISO) is dealing with a broader view on usability and is giving precise definitions. A first definition of usability was given in DIN 66234 which was later adopted; e.g. in DIN EN ISO 9241-10 and DIN EN ISO 9241-11; and is currently available in a revised form in DIN EN ISO 9241-110 (2008):

> ***"Usability is the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use."***

This definition provides two main aspects of usability that require further investigation. On the one hand, it describes the focus of usability by introducing the participating factors, namely the user, the product (e.g. an application with adaptive UIs), the goals of the user (e.g. a specific task) and the context of use. Especially, the context of use has to be defined properly for an evaluation. In addition, DIN EN ISO 9241-110 (2008) states that not every context of use can be covered by this definition; e.g. safety critical systems and collaborative work. One can imagine that different context situations might infer different goals for interaction by the user; e.g. users might not want to share private information from online-banking while there are unknown people around. In the remainder of this thesis these four factors - application, user, context of use and task - are being constantly referred to when describing information required for determining usability. On the other hand, the definition provides the concepts used to measure the extent of usability; i.e. *effectiveness*, *efficiency* and *satisfaction* (Table 2.1).

**Table 2.1.:** Concepts for usability adapted from DIN EN ISO 9241-110 (2008).

| | |
|---|---|
| **Effectiveness** | Describes the accuracy and completeness with which the user achieves a specified goal. |
| **Efficiency** | Is the cost in relation to the accuracy and completeness with which the user achieves a specified goal. |
| **Satisfaction** | Describes the joy of use and absence of interferences while the user achieves a specified goal. |

[1] http://www.w3.org/ - Accessed in November 2012.

Hassenzahl (2004) distinguishes between *pragmatic* usability attributes; e.g. efficiency and effectiveness as described by DIN EN ISO 9241-110 (2008); and *hedonic* usability attributes that deal with factors like novelty and beauty. While both types of attributes influence the perceived usability, studies show that they depend on the situation under which they are measured (Hassenzahl et al., 2008); e.g. in safety-critical situations a pragmatic UI is preferred to the hedonic UI because it usually allows for a more efficient and effective interaction process. This relates to the fact that for determining pragmatic usability attributes, a task is required, which users need to perform. On the other hand, it is possible to measure hedonic attributes without a specific task because they relate more to aesthetics than to functionality. The scope of DIN EN ISO 9241-110 (2008) is not to assist designers in aspects like aesthetics, corporate design or marketing, which are usually part of style guides. More strictly than the described hedonic view on usability, the definition of usability according to DIN EN ISO 9241-110 (2008) requires that interaction needs to be goal-driven. Thus, the designer has to be aware of potential interaction goals in an early stage of development; e.g. in a task analysis phase. Furthermore, DIN EN ISO 9241-110 (2008) specifies seven axes along which the usability of a dialogue can be expressed on a finer level of granularity (Table 2.2).

**Table 2.2.:** Usability measurement axes adapted from DIN EN ISO 9241-110 (2008).

| | |
|---|---|
| **Suitability for the task** | Is provided if the dialogue assists the users to achieve their goals effectively and efficiently. |
| **Self-descriptiveness** | Is provided if every dialogue step is directly comprehensible or confirmed by the application (if required by the user). |
| **Controllability** | Is provided if the user is able to start and control the dialogue flow as required. |
| **Conformity with user expectations** | Is provided if the dialogue is consistent and applies to the domain knowledge it is used for. |
| **Error tolerance** | Is provided by a dialogue if the specified goal can be achieved with manageable correction effort even though erroneous input data was given by the user. |
| **Suitability for individualization** | Is provided if the dialogue can be configured to the current task and the current user's needs and abilities. |
| **Suitability for learning** | Is provided if the dialogue assists the user in the acquisition of the application. |

### 2.1.2. Measuring Usability

Having a clear definition of usability is crucial for evaluating the application along certain aspects of this definition. Nevertheless, concrete measurements have to be applied to make a statement about the usability of an interactive application.

Nielsen's famous ten usability heuristics can also be seen as a specific method that is applied to evaluate the usability of an application.[2] The main concepts behind these heuristics give a good insight on what can be named a usable interface (Table 2.3). However, evaluators still rely on (their own) expert knowledge when applying measurements for determining the extent of each of these qualitative criteria. Furthermore, Shneiderman and Plaisant (2004) give a short list of usability criteria that is focused on a practical point of view and includes qualitative and quantitative measurements. These include *time to learn*, *speed of performance*, *rate of errors by users*, *retention over time* and *subjective satisfaction*. Usually, not all of these measurements can be considered equally in an evaluation process.

**Table 2.3.:** Nielsen's ten usability heuristics.

| | |
|---|---|
| **1.** | **Visibility of system status** |
| **2.** | **Match between system and the real world** |
| **3.** | **User control and freedom** |
| **4.** | **Consistency and standards** |
| **5.** | **Error prevention** |
| **6.** | **Recognition rather than recall** |
| **7.** | **Flexibility and efficiency of use** |
| **8.** | **Aesthetic and minimalist design** |
| **9.** | **Help users recognize, diagnose, and recover from errors** |
| **10.** | **Help and documentation** |

Table 2.4 lists examples for quantitative usability measurements. But, depending on the focus of the chosen criteria and measurement methods, it is possible that usability is judged differently by different experts. Whiteside et al. (1988) not only propose absolute scales for these criteria but setting the scales individually; e.g. based on former evaluations with the same user and system, competitive systems or earlier prototypes. In addition to the examples listed in Table 2.4, Dix et al. (2003) state that it is important that the whole functional architecture needs to be considered in the evaluation process. This also includes the users' cognitive capacities. Thus, different usability measurements

[2] Available at: http://www.useit.com/papers/heuristic/heuristic_list.html - Accessed in November 2012.

ought to be combined to prevent a limited evaluation scope on the UI surface features only; i.e. human user and application have to be considered together.

**Table 2.4.:** Criteria for measuring usability and for determining usability methods adapted from Whiteside et al. (1988).

| | |
|---|---|
| **1.** | **Time to complete the task** |
| **2.** | **Per cent of task completed** |
| **3.** | **Per cent of task completed per unit time** |
| **4.** | **Ratio of successes to failures** |
| **5.** | **Time spent in errors** |
| **6.** | **Per cent or number of errors** |
| **7.** | **Per cent or number of competitors better than it** |
| **8.** | **Number of commands used** |
| **9.** | **Frequency of help and documentation used** |
| **10.** | **Per cent of favorable/unfavorable user comments** |
| **11.** | **Number of repetitions of failed commands** |
| **12.** | **Number of runs of successes and of failures** |
| **13.** | **Number of times interface misleads the user** |
| **14.** | **Number of good and bad features recalled by the user** |
| **15.** | **Number of available commands not invoked** |
| **16.** | **Number of regressive behavior** |
| **17.** | **Number of users preferring your system** |
| **18.** | **Number of times users need to work around a problem** |
| **19.** | **Number of times the user is disrupted from a work task** |
| **20.** | **Number of times user loses control of the system** |
| **21.** | **Number of times user expresses frustration or satisfaction** |

Finally, these measurements are not exhaustive and usability is also influenced by concepts that are hard to measure, but can be addressed using questionnaires. For example Cooper (1999) states that software systems ought to be designed to behave politely because people tend to interact with new media like they would deal with a real person. Accordingly, Cooper lists favorable characteristics that polite software applications need to incorporate; e.g. interest in the user, responsiveness and trustworthiness.

### 2.1.3. Common Usability Evaluation Methods

Usually, the best way for conducting usability evaluations is carrying out tests with real users; e.g. by applying the *Think Aloud* method (Nielsen, 1993a) where participants are asked to tell what is on their mind while interacting with (a prototype of) the application. Such methods belong to the class of *formative testing* and usually do not explicitly use computer systems and algorithms for automating the evaluation process. Though, tools can be used to support in the process; e.g. logging and camera recordings.

Besides tests with users, usability evaluation can also be conducted by experts who are checking the application against predefined criteria; e.g. cognitive walkthrough methods (Polson et al., 1992) and heuristic evaluation (Nielsen and Molich, 1990; Nielsen, 1992), which are approaches for *formative inspection*. Nielsen (1992) states that in general five different evaluators are required to find about 75% of the usability problems, while more evaluators do not bring much more effect.

Carrying out any of these methods is usually a time consuming and expensive task; e.g. choosing the right participants for evaluation or working through the video log data. Furthermore, a common problem when evaluating UIs is having an objective point of view on the evaluation. Nielsen et al. (2005) describe how to integrate usability evaluation into the design process. They examined different usability evaluations and found out that often the designers and the evaluators are the same persons. A similar problem lies in the design of questionnaires that are handed out to participants. Using the right wording in the questionnaires is a crucial challenge that has a strong influence (Larsen et al., 2008). Both pitfalls obviously lead to a lack of independence in an evaluation.

Farenc et al. (1995) list typical problems that designers run into when applying guidelines and evaluation methods due to missing standards and completeness of existing approaches. When conducting usability evaluations with real users, there are also problems that have to be considered. One of them is to evaluate as closely as possible to the context of authentic use (Abowd and Mynatt, 2000), which means that users might behave unnaturally in a laboratory environment when being given specific instructions. Still, usability tests with real users and experts should always be part of the development process and have to be included into the different phases as soon as possible.

### 2.1.4. Analytical Modeling Approaches for Usability Evaluation

Analytical modeling approaches are applied by experts to predict aspects of HCI; e.g. by cognitive modeling of the interaction process. These methods alone are not automated but build the basis for tools that incorporate these analytical modeling concepts. All of these approaches are performed during development with the goal of predicting aspects of human interaction with the application.

#### GOMS

The GOMS approach for analytical modeling is based on models of *goals*, *operators*, *methods* and *selection rules*. It was initially proposed by Card et al. (1983) and is usually

used to evaluate required knowledge for accomplishing certain tasks. In a GOMS model, the direction of interaction is represented with the help of *goals*. These goals can be divided into hierarchically ordered subgoals for breaking down the complexity. In order to model how a human user would achieve these goals, *operators* are being performed on a perceptual, cognitive or motor-act level. Effects of these operators can lead to changes in the internal mental state of the modeled user or changes in the external environment. Execution times are bound to these operators to predict the overall interaction time. *Methods* describe sequences of operators to achieve subgoals. If more than one method can be used to achieve a goal, the *selection rules* are applied. These rules represent the user's knowledge depending on the current task. John and Kieras (1996) list different GOMS analysis methods with different application areas. Besides evaluating required knowledge for a task, GOMS models can be used for predicting execution times and the time it takes to learn interaction with a (new) UI.

#### Keystroke-Level Model

The Keystroke-Level Model (KLM) is a simplified approach of GOMS for modeling human performance (Card et al., 1983). The focus of this analytical modeling method is predicting the execution time of an expert user for a specified task. Larger tasks can be divided into smaller unit tasks to ease the application of this method. Unit tasks can be further subdivided into an acquisition phase and an execution phase. An expert writes down the method to perform the task and counts the keystrokes involved for the execution phase. The time to execute these keystrokes ($T_K$) and necessary preparation steps; i.e. pointing ($T_P$), homing ($T_H$), drawing ($T_D$) and mental preparation ($T_M$), is summed up together with the system response time ($T_R$):

$$T_{execute} = T_K + T_P + T_H + T_D + T_M + T_R \tag{2.1}$$

The execution times for these operations are based on user tests and provide estimations that have been proved by many experiments. However, there may be some fine tuning necessary by the evaluators to apply this method to different domains or tasks. Furthermore, Card et al. (1983) state that the amount and the placement of the mental operator can be ambiguous. Therefore, they provide a set of rules which stem from psychological assumptions and heuristics. Nevertheless, KLM can be used to evaluate different design choices based on the execution time of a specific task.

### Programmable User Model

The idea for a *Programmable User Model* (PUM) was first described by Young et al. (1989). The basic concept of this predictive analytical modeling approach is programming a user model capable of simple human problem solving strategies and common sense knowledge. This user model needs to be extended by the designer to be able to interact with the UI of the application and its specific domain. The main idea behind this concept is showing that interaction with the UI would be intuitive and therefore easy to implement into a PUM. For this purpose the designer needs to consider what knowledge is required for a goal-directed interaction and if the intended user has to have this knowledge in advance. Initial implementations based on production rules from the SOAR[3] architecture (Newell, 1990) led to the insight that human knowledge and cognitive processes were hard to model this way.

Following the maximum rationality hypothesis by Newell (1990), Blandford et al. (2004) enhanced the PUM approach by assuming that users interact in a rational way when having a goal to obtain. The knowledge of the state of the world is formalized as beliefs that can be altered through operations. Operations are represented by beliefs about actions and their effects. Preconditions can be chosen to constrain these operations.

## 2.2. Context of Use

Besides the definition of context of use from DIN EN ISO 9241-110 (2008), which is specialized on usability and interaction aspects as described in Section 2.1, there exist more general definitions for context when dealing with adaptive UIs.

Abowd et al. (1999) define context in a generic way. They state that any information usable to characterize the situation of a person, place, or object relevant to the interaction between user and an application is used to define the context. More specifically, this also includes the application and the users themselves.

Schilit et al. (1994) refine *context environment* into *user environment*, *computing environment* and *physical environment* (Table 2.5). This is comparable to the view used by Calvary et al. (2003) who relate to the similar concepts of *user*, *platform* and *environment* that are used in this thesis. Chen and Kotz (2000) add *time* as a further aspect. An advantage of this perspective is that information about the current time of day, week, month and year could provide a higher reasoning on other available context information and even enable a *context history* (Chen and Kotz, 2000).

[3] State, Operator Apply Result

**Table 2.5.:** Definition of context environment adapted from Schilit et al. (1994).

| | |
|---|---|
| **Computing environment** | Describes the surrounding devices that are accessible for user input and display, the network capacity and connectivity and finally the available processors as well as costs of computing. |
| **User environment** | Sums up information about the location of the user, a collection of nearby people, and the user's social situation. |
| **Physical environment** | Is a description of general surrounding information like the current lighting and noise level. |

*Context-awareness* describes an attribute of software systems that are able to react to changes in the current context of use. Schilit et al. (1994) propose to include information about changes over time in the location of use, the collection of nearby people, hosts and accessible devices. Abowd et al. (1999) propose to enhance this view with relevancy to the current user's task to exclude irrelevant information. Depending on this information, changes in context need to be identified. Here, usually *entering a context* and *leaving a context* are considered while *being in a context* might also be reacted to (Schmidt, 2000).

A slightly different approach is described by Crowley et al. (2002). They define context as a composition of situations. Each of these situations consists of a particular assignment of *entities* (observable variables) to *roles* (actions that can be performed in a task) they can fulfill and a set of *relations* that can be defined over properties of such entities. Below is a definition of context, according to Crowley et al. (2002), for a User $U$ and a Task $T$:

$$Context(U,T) \Rightarrow \{Role_1, Role_2, ..., Role_n; Relation_1, ..., Relation_m\} \tag{2.2}$$

The idea behind this definition of context for context-aware systems is that whenever an assignment of entities to roles or an existing relation changes, only the situation changes while the context remains the same. On the other hand, context changes are defined by a change of the set of roles or relations; i.e. adding or removing a role or relation. This gives context-aware applications the chance to react to minor changes differently than to major changes in the context.

When defining an architecture based on this ontology, Crowley et al. (2002) propose to add a meta-supervisor component. This component observes all entities and their assignments to roles as well as their relations to track changes in situations and context and to react to these changes appropriately.

Calvary et al. (2003) distinguish between *predictive* context of use, which can be foreseen by the designer during development time, and *effective* context of use, which really occurs during runtime of the application. Context-aware systems need to provide means for coping with this potential gap; e.g. by assigning an effective context to a predictive context.

#### User Factors vs. User as Part of Context of Use

While information about the user is also part of the context of use, in this thesis two types of information about the user are distinguished and described separately:

- User factors relate to information about the user that is used for AUE. This information relates to how interaction is simulated by a user model and which information about the user is required.
- User information that is part of the context of use and used by the application (e.g. for adaptations of the UI).

## 2.3. Adaptation of User Interfaces

Before the concept and main aspects of *adaptation* are presented, the terms *runtime* and *development time* of applications need to be specified. However, this requires starting with a definition of the term *application*.

This thesis follows the definition of Bass et al. (1992), who relate to an application as the total system that is developed for its end-users. They further divide an interactive application into two parts: the *application domain software*, which defines the domain the application was developed for, and the *user interface software*, which consists of the UI runtime system (UIRS) and a UI toolkit. This thesis does not explicitly distinguish between application (consisting of a functional core and a UI) and the more general term *system*. Instead a focus is set on (adaptive) UIs and relevant information for AUE.

### 2.3.1. Adaptability and Adaptivity

In general, the term *adaptation* is used for describing the ability of UIs to alter aspects of their appearance to the context of use at runtime. More specifically, according to Totterdell and Boyle (1990) adaptation can be achieved in two ways. On the one hand, *adaptability* denotes approaches that integrate users into the adaptation progress

and letting them customize the UI. On the other hand, *adaptivity* is used for approaches that are capable of automatically adapting their UI without further user requests. This thesis focuses on necessary means for adaptivity of user interfaces, because the approach is applied during the evaluation phase at development time.

### 2.3.2. Plastic User Interfaces and Executable Models

The ability of UIs to adapt to different variations of the context of use and thereby being able to preserve certain usability aspects within a predefined range of properties is defined as *plasticity* (Calvary et al., 2002). These properties usually need to be defined by a designer at development time. Calvary et al. (2002) suggest using properties according to Gram and Cockton (1997); e.g. observability and predictability.

#### CAMELEON

Calvary et al. (2003) present the CAMELEON reference framework for structuring the development process of plastic UIs. On the one hand, this process can be characterized depending on the amount of different variations of context of use and the limiting boundaries the UI is able to cope with. On the other hand, the process can also be characterized by the human effort required in the adaptation process.

Furthermore CAMELEON adds aspects for classification and multi-target UIs and defines a set of *ontological models* consisting of meta-models; e.g.:

- **Domain** described by concepts and tasks.
- **Context of use** consisting of user (user environment), platform (computing environment) and environment (physical environment).
- **Adaptation** describes the adaptation process to changes in the context of use.

A set of models conforming to these ontological meta-models are summed up as a *config*. An adaptive application can have different configs; e.g. for each adaptation of the application. Furthermore, CAMELEON describes that models from each config can be assigned to different levels of abstraction (Figure 2.1):

**Concepts and Task Models** consist of descriptions from the designer and define possible actions of the user and the application. The tasks can be ordered e.g. by a temporal or hierarchical relation to reflect the interaction process. An example of an implemented task model is depicted and further described in Figure 7.2 on page 110.

**Abstract User Interface (AUI)** is an intermediate model (transient) and independent from the specifications of the target UI and platform. The AUI channels the models from the concepts and tasks to a logical structure by defining abstract *interactors*. Figure 7.1 on page 109 depicts a UI screen of a cooking application that is annotated with interactors of the corresponding AUI model.

**Concrete User Interface (CUI)** is an intermediate model; e.g. a mock-up of the UI that cannot be run independently, but specifies the final look by implementing concrete interactors.

**Final User Interface (FUI)** is the actual source code that can be compiled and run on a runtime infrastructure.

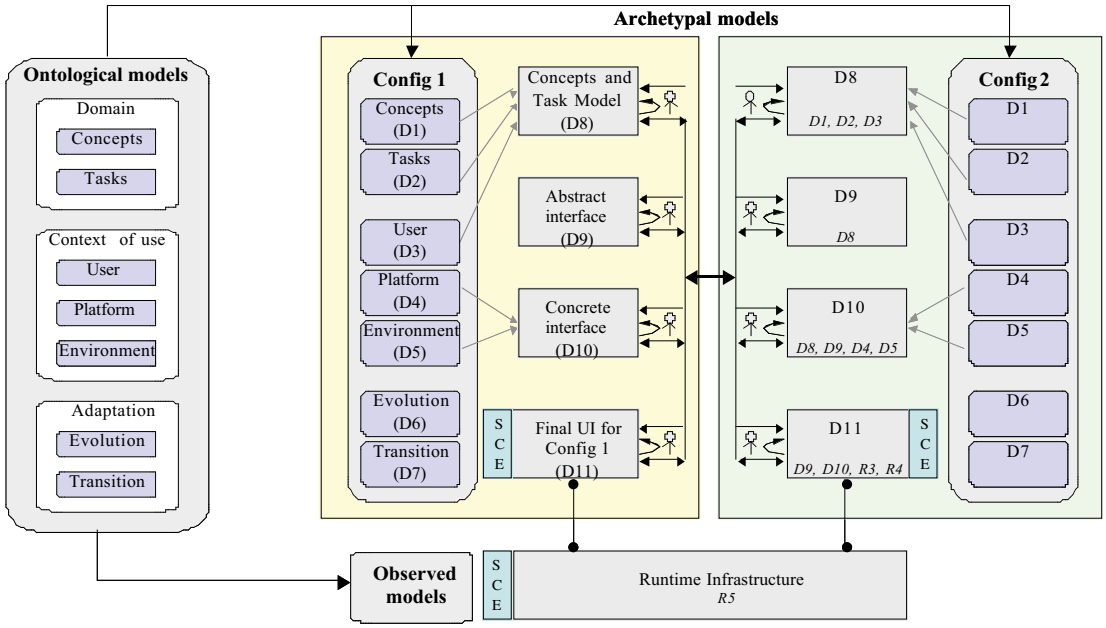


**Figure 2.1.:** CAMELEON reference framework for structuring the development process of plastic UIs taken from Calvary et al. (2003).

An important concept of CAMELEON is that, starting with any of these models, all other models can be created by a process of vertical transformation; i.e. *reification* from abstract models to more concrete models and *abstraction* from concrete to more abstract models. Models of the same reification level can be horizontally transformed between different configs; e.g. translating between different programming languages (Figure 2.1).

A main benefit of using a model-based development approach conforming to this framework is that using these explicit UI models takes the design process to a computer-processable level, on which design decisions become understandable for automatic sys-

tems. There exist UI description languages; e.g. UsiXML[4] (Limbourg et al., 2005) and TERESA (Mori et al., 2004); which use this approach for generating the final UI code from models and for adapting the UI by transforming their models. There also exist tools that are based on these description languages and assist the developer in creating the required models for the UI (Michotte and Vanderdonckt, 2008; Collignon et al., 2008). Section 4.2.1 presents these and further approaches and explains the usage of models conforming to CAMELEON in more detail.

### CAMELEON-RT

Balme et al. (2004) extended CAMELEON from a runtime point of view. They propose CAMELEON-RT, which is a conceptual architecture reference model for distributed, migratable and plastic UIs, further referred to as DMP-UI. While the concept of plasticity was already explained, migration and distribution still need to be defined. According to Balme et al. (2004):

- **Migration** of a DMP-UI is the transfer of (parts of) the UI to different interaction resources from potentially different platforms.
- **Distribution** of DMP-UIs relates to the fact, that a UI can use distributed interaction resources; e.g. mouse and keyboard for input interaction and TV or monitor as output interaction resources.

A crucial point is that migration and distribution can be achieved *statically* or *dynamically*; i.e. off-line between sessions or on the fly. Furthermore, CAMELEON-RT follows the definition of situations and context of use according to Crowley et al. (2002), which is presented in Section 2.2.

Finally, Balme et al. (2004) distinguish between *closed adaptations* and *open adaptations*. The former describes that an application handles the adaptation process by itself, because the adaptation lies in the domain of the application. The latter relates to components in the DMP layer (evolution engine, components retriever and a configurator) of the runtime infrastructure, which adapt the application in case that the application cannot cope with the current context of use (e.g. because it was not foreseen during development time). However, such applications need to provide interfaces for being open-adaptive.

For maintaining open adaptations CAMELEON-RT suggests that the properties of situations are being observed by a situation synthesizer component. This component triggers

[4] http://www.usixml.org/ - Accessed in March 2014.

an evolution engine that provides information on how to adapt the underlying models. Finally, an adaptation producer component carries out the adaptation by reificating, abstracting or translating the models of the application.

### Executable Models

In this thesis the term *executable* (models) is used in a broad interpretation by referring to an approach that is capable of providing a follow-up state of the whole interactive application (and the models) in a given state. This needs to be applicable especially for interaction with the models and also for changes in models such as the context of use model to provide adaptations.

Sottet et al. (2008) propose *mappings* for linking a graph of models at runtime and give examples for reverse engineering abstract models from the final UI. CAMELEON-RT gives examples for approaches that relate to components such as *reificators* to provide executable code from the development models or *abstractors* in order to gain information from components to models and also provides the DMP layer, which deals with providing adaptations at runtime. Further approaches (Blumendorf et al., 2008; Lehmann et al., 2011) incorporate executability directly into the models and are thus able to bridge development time and runtime (see Section 4.2.1 and Section 4.2.3).

## 2.4. Summary

This chapter presented the main concepts and terms used throughout the remainder of this thesis and relevant for the described approach. Different views and a definition of usability were presented to reflect the multitude of facets from this research field. Existing measurements for describing, predicting and comparing the usability of different applications and their UIs were presented as well as the research field of usability evaluations that are applied during development. Subsequently, context of use was focused because interaction and evaluation are performed within a specific context of use. In addition, context of use builds a cornerstone of the research field of adaptive UIs, which is also the domain for the general concept of this thesis.

In the following chapter required information for a simulation-based automated model-based usability evaluation of adaptive user interfaces is presented.

# 3. Required Information for Automated Usability Evaluation

In this chapter basic required information for automated usability evaluation during the evaluation phase at development time of adaptive UIs is derived. Conforming to the definition of usability stated in Section 2.1, detailed information about four basic factors are identified and described in more detail:

1. **Application Factors**, such as UI information and interaction logic (Section 3.1),
2. **Context of Use Factors** that influence the interaction and the adaptation process (Section 3.2),
3. **User Factors** that are relevant for simulating user behavior and testing the application according to this behavior (Section 3.3), and
4. **Task Factors** or a set of goals that the user tries to accomplish (Section 3.4).

The distinction into these four describing factors is also kept in the remainder of this thesis. While the descriptions of required information are mostly generic, **this thesis focuses on the evaluation of adaptive graphical user interfaces**. Thus, detailed descriptions and examples used for illustration are from this domain.

## 3.1. Application Factors

Specific information about the application factors for simulation-based AUE methods can be separated into information about the UI, the interaction logic and the application state space.

### 3.1.1. Access to UI Surface Information

An accessible description of the application needs to be provided because simulation-based AUE methods rely on interactions between user and application. More specifically,

detailed information about the UI (and its interaction logic) is required and needs to be provided in a computer-processable way.

When applying AUE methods to GUIs, required information comprehends all visible UI elements along the path of interaction and their specific layout attributes, which are:

- Type of the UI element,
- Size and position on screen, and
- Caption of the UI element.

For example, information about the caption of the UI elements (in combination with its position on screen) allows estimations of the learning times for novice users (e.g. using GOMS). Specific attributes such as the color and the size can serve as input in case of evaluations for elderly or people suffering from visual impairment; e.g. UIs may be checked for people with dyschromatopsia. Thus, the first required information is:

**(Req. Inf. 1)** **UI Surface Information**
*An approach of simulation-based AUE requires access to application-specific UI surface information.*

### 3.1.2. Interconnection of UI Information with Interaction Logic

For a simulation-based AUE approach it is essential to establish a connection between the UI elements and the task that the user is currently performing from the application's point of view; i.e. as intended by the designer. This way an automated simulation of user interactions and appropriate system behavior can be achieved. It also implies that it has to be traceable what happens next, if a specific UI element has been interacted with; e.g. by clicking on a button the next UI screen gets rendered. More precisely, this interconnection allows reasoning about the effects of using specific UI elements for a specific purpose within the current simulated task. When evaluating adaptive UIs, this also includes changes to the UI that are triggered automatically due to changes in the context of use (see Section 3.2). Consequently, this interconnection between UI elements and interaction logic builds the basis for the next required information:

**(Req. Inf. 2)** **Interaction Capabilities**
*An approach of simulation-based AUE requires access to interaction capabilities of UI elements and their purpose for specific tasks.*

**(Req. Inf. 3)** **Application State Space**
*An approach of simulation-based AUE (of adaptive UIs) requires access to information about all potential follow-up states of an application after simulated user interaction or adaptations occurred.*

Moreover, **(Req. Inf. 2 - Interaction Capabilities)** and **(Req. Inf. 3 - Application State Space)** serve as enablers for a goal-directed interaction simulation that is discussed in more detail in Section 3.4.

## 3.2. Context of Use Factors

When evaluating adaptive UIs the system response to user input may also depend on the context of use and thus might be different in different situations. This is leading to a multitude of unique situations for which an appropriate system response needs to be ensured. Usually, common approaches let specific components handle the analysis of these situations and then apply appropriate adaptations (see e.g. Balme et al. (2004)). Thus, if such components and a representation of the context of use, which can be simulated, are available, simulation-based evaluation can be applied to extensively test possible interaction paths.

As described in Section 2.2 the context of use is subdivided into information about the *user environment*, the *computing environment* and the *physical environment*.

Information about the *user environment* requires relevant information that can be sensed by the context-aware system via its sensor systems and an internal representation of the user; e.g. via a user profile. Depending on the integrated sensor data, this may include the current location of the user and other people around, as well as historical and social data from a user profile. All of this ought to be included into the simulation-based interaction process for triggering required adaptations.

Information about the *computing environment* needs to include at least relevant information about available interaction devices that can be used as input or output; e.g. keyboard, mouse, touchscreen or display. Some AUE approaches require this information to be integrated into information about the application (see Section 3.1), because in some cases no sharp distinction can be made between software and hardware components.

Information about the *physical environment* includes surrounding factors; e.g. acoustic or lighting conditions. In case of adaptive UIs the modeled surrounding factors need

to include at least those which are sensed by the context-aware system and are used as triggers for potential adaptations to the UI. Furthermore, side effects can be included into the evaluation, if this sensor data from the outside world is available to the usability evaluation. As an example serves a user performing other actions besides the current task; e.g. driving a car while interacting with a navigational application as described on the cognitive level by Salvucci (2009). In such a case, the traffic situation or radio music may have to be simulated as well. As a result, constraints arising from secondary tasks, which are related to the physical environment, need to be modeled to reflect distraction from the UI or the primary task.

Based on the provision of available information from the context of use to the simulation of the application's and the user's behavior, the fourth required information is:

**(Req. Inf. 4)** **Context of Use**
*An approach of simulation-based AUE (of adaptive UIs) requires access to a representation of the context of use that can be simulated and modified to the evaluation goals.*

## 3.3. User Factors

In order to provide a wide basis for potentially applicable AUE methods, required information regarding the user for simulating interaction have to be addressed. This information can be divided into modeling the user's expertise in the domain and representing the user's abilities for interaction with the specific application and interaction devices.

### 3.3.1. Information about Users' Expertise

On the one hand, users may differ in their expertise regarding the domain of the application, its UI and the used interaction devices, but also regarding the domain of the specific task. On the other hand, there exist adaptive UIs that provide different adaptations depending on the availability of this information; e.g. UIs for experts may provide more configuration choices compared with default presets that suit for most users. As a consequence, user expertise ought to be simulated to evaluate the effect on the interaction process on the user side and on the system side.

Most AUE methods are exclusively validated for expert users in the domain of the application and therefore optimal interaction paths are evaluated only. Just a few approaches

model novice users, but then usually require that the simulated user has no prior knowledge of the domain; e.g. Teo and John (2011). A main difference between modeling expert users and novice users is that in the latter case interaction problems due to a wrong understanding of the application can be accounted for. These errors have usually been excluded from AUE methods, but have gained increasing attention lately (Teo and John, 2011; Blackmon et al., 2005; Steinnökel et al., 2011; Halbrügge et al., 2015b).

Unfortunately, errors due to a lack of experience with the application or domain are hard to predict using AUE. A notable exception is the simulation of browsing behavior, where category labels are evaluated based on semantic similarity between goal concepts and UI element captions; e.g. as by Teo and John (2011) and Blackmon et al. (2005). If errors cannot be predicted automatically, at least known errors can be modeled and their consequences can be evaluated using a simulation approach. A practical application of an AUE method is then to develop knowledge about error types and their precondition, either in a general way, or during the task analysis phase preceding the design of the actual application.

Wood and Kieras (2002) point out that the user behavior following an error can be hard to predict. It involves (among other things) that the user notices that an error has occurred and reasons about recovery possibilities. Therefore, the description of the user also profits from including some kind of mental representation of the application (and knowledge of the world), which allows predicting expected consequences of actions and comparing them to the actual outcome. Such an internal description of the application is sometimes referred to as a *mental model* (Norman, 1983).

Finally, different users may have different preferences regarding interaction devices and techniques; e.g. using shortcuts or preferring certain modalities over others. These preferences ought to be included into the representation of the user and then have an influence on the chosen actions during simulation and evaluation.

Thus, these different needs can be summed up as required information, which describes the importance of a user representation and its effect for simulation-based AUE:

**(Req. Inf. 5)** **Users' Expertise**
*An approach of simulation-based AUE requires access to a user representation capable of describing users' expertise.*

### 3.3.2. Information about Users' Abilities

Besides users' expertise, additional factors are required for a more beneficial combination of AUE and adaptive UIs. This becomes especially true when evaluating the usability for users with special needs and disabilities, such as visual or motor impairments (Keates et al., 2000; Trewin and Pain, 1999).

Having this enhanced information about the abilities allows simulating different user groups and evaluating the effects of:

- Different layout variants or adaptations in combination with information from the application's UI surface and interaction logic (see Section 3.1), and
- Different interaction devices and surrounding effects in combination with information from the context of use (see Section 3.2).

Consequently, AUE profits from a modeling of different user groups based on abilities, as it allows reasoning about the effects of different adaptations:

**(Req. Inf. 6)** **Users' Abilities**
*An approach of simulation-based AUE (of adaptive UIs) requires access to a representation of users' abilities.*

## 3.4. Task Factors

User interaction usually serves one or more specific goals, which users try to achieve during interaction. Simulation-based AUE requires these goals to be part of the domain of the application in order to simulate interaction towards a final goal-state. Thus, goals for interaction may be specified in form of a task that the user wants to accomplish.

When conducting usability tests with real users, goals are usually predefined and the participants receive a description of the task to perform and (depending on the applied method) a clear description when the goal is achieved. Like real user tests, most AUE methods based on simulated interaction require such predefined tasks; even though the description and provision of the task differ between different methods (see Section 4.1).

If the task is to be used in simulations, the actions performed by the user for reaching the goal state can be provided within the task description. This solution path is comparable to a step-by-step walkthrough. However, in case of some AUE methods, these steps are not contained in the task description or it is not desired to have this information

in advance; e.g. when evaluating behavior of novice users. Instead, the required steps have to be determined on the fly based on information describing how users would try to proceed; e.g. with the help of rules describing user behavior, available knowledge or further semantic information required for accomplishing a task.

Thus, the final required information is:

**(Req. Inf. 7) Goal-State**
*An approach of simulation-based AUE requires access to a clear definition of a goal-state and a description with information relevant for accomplishing this task, whether as a list of actions to perform or via an integrated solution approach.*

## 3.5. Summary

In this chapter required information for a simulation-based AUE approach is derived that can be applied to the evaluation phase of adaptive UIs during development time (Table 3.1).

**Table 3.1.:** Required information for simulation-based AUE that can be applied to adaptive UIs.

| | | |
|---|---|---|
| **Req. Inf. 1** | **UI Surface Information** | **Application Factors** |
| **Req. Inf. 2** | **Interaction Capabilities** | |
| **Req. Inf. 3** | **Application State Space** | |
| **Req. Inf. 4** | **Context of Use** | **Context of Use Factors** |
| **Req. Inf. 5** | **Users' Expertise** | **User Factors** |
| **Req. Inf. 6** | **Users' Abilities** | |
| **Req. Inf. 7** | **Goal-State** | **Task Factors** |

While the focus was set on how application-specific information can be provided, relevant required information regarding the user factors was stated as well. Especially the defined user factors account for a high variety during the simulation of interaction and might be interrelated with the adaptation to the application's UI and behavior.

Before the approach conforming to this required information is described in Chapter 5, an overview of the state of the art in related work highlighted by this chapter is presented next.

# 4. Related Work

This chapter presents an overview of related work, which covers the research areas from the previous chapter. The descriptions focus on approaches using models during development time in order to represent information about the four basic factors derived in the previous chapters - *application factors*, *context of use factors*, *user factors* and *task factors*.

Section 4.1 gives an overview of model-based usability evaluation (Section 4.1.1) and specifically presents AUE approaches from the domain of cognitive architectures for predictive simulations (Section 4.1.2) and existing AUE tools (Section 4.1.3). All of the described AUE methods are commonly used for evaluations during development time to predict interaction parameters and thereby evaluating the application's usability. Required information for each method is highlighted and typical results of each evaluation method are presented. Section 4.1.4 presents a workbench for (semi-)automated usability evaluations that shares concepts with the approach of this thesis. Section 4.1.5 gives a comparative overview of the discussed AUE approaches.

In Section 4.2 current approaches of model-based UI development for adaptive UIs are explained. The section starts by focusing on model-based UI development in relation to the required information from Chapter 3 in general and then leads over to plastic UIs (Section 4.2.1) and approaches for evaluation during model-based UI development (Section 4.2.2). Subsequently, a detailed description of a specific approach (Section 4.2.3) is presented that is applied to the implementation of this thesis as a proof of concept how existing application models can be used. Finally, a discussion concludes this section (Section 4.2.4).

In Section 4.3, current user modeling concepts and methods are described with a focus on which required information is modeled and how this information is provided during development (Section 4.3.1). Subsequently, examples from user modeling in the domains of adaptivity and evaluations in smart homes are presented (Section 4.3.2).

Both research fields, user modeling and model-based UI development, directly influence the work proposed in this thesis. For examples of use, the domain of smart home

environments is targeted. Especially this domain demonstrates crucial use cases for the work proposed here, because smart environments have a strong need to adapt to their users based on the context of use for reasons of acceptance and ease of use.

The main conclusions from the presented related work are drawn in Section 4.4 and a final summary of this chapter is presented in Section 4.5.

## 4.1. Automated Model-based Usability Evaluation

This section presents the research field of model-based usability evaluation and focuses on automated approaches and tools. A general discussion of AUE, a commonly used taxonomy and the typical process and involved costs are presented first. The remainder of this section presents current approaches of AUE structured by their capabilities for simulation and thus automation.

For the sake of brevity and in direct reference to the approach described in this thesis, the scope of the described approaches is narrowed according to Section 1.3 and the required information from Chapter 3. Consequently, AUE methods based on a theory of how human users are interacting with applications (*predictive simulation*) are focused.

### 4.1.1. Taxonomy, Benefits, Costs and Limitations

In contrary to common usability evaluation methods like user testing, AUE is carried out with assistance of computer-aided tools. Ivory and Hearst (2001) propose a general taxonomy along which they characterized 128 different usability evaluation methods. This taxonomy is based on (1) *the usability evaluation method*, (2) *the type of automation* and (3) *the human effort involved*. They also list general advantages of AUE methods, some of which are:

- Uncovering **various types of errors** more consistently,
- **Increasing** the coverage of evaluated **features**,
- Enabling **comparisons** between **alternative designs**,
- **Predicting times and errors** across an entire design,
- **Reducing** the need for evaluation **expertise**,
- Can be **embedded** directly **into** the **development process**, and
- **Reducing** the **cost** of usability evaluation.

When it comes to applying AUE methods, there are several costs that have to be weighed against the benefits. Usually, a high effort is involved for creating the required models, as described in the following.

John and Suzuki (2009) describe the process for creating an AUE tool based on a solid theory (Figure 4.1). They highlight the importance of validating any modifications of the underlying theory, tools and models against observed data from real users; e.g. when entering a new domain the theory was not designed for.

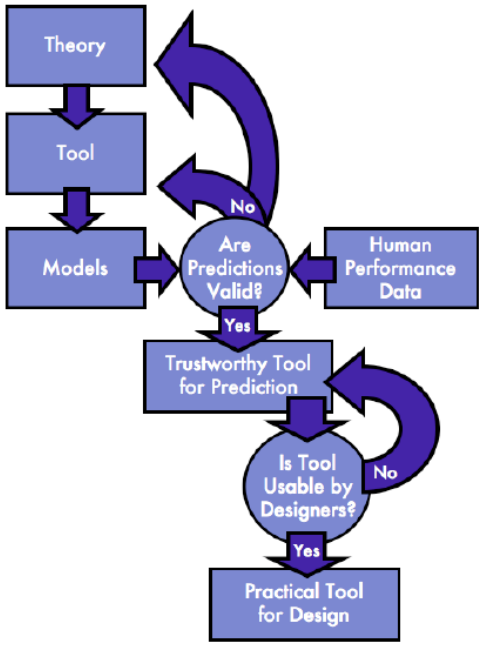


**Figure 4.1.:** Process for providing a practical tool for modeling human performance with AUE from John and Suzuki (2009).

Based on the described process, John and Jastrzembski (2010) list a non-exhaustive set of costs a designer needs to invest when applying AUE to the development process:

- **Learning** and understanding **the modeling theory**.
- **Learning** how **to use the modeling tool** itself.
- Obtaining the **knowledge to create** an appropriate **cognitive model in the task domain** for the intended user group.
- The **time it takes to create the model** in the modeling theory with the help of the modeling tool.

#### Discussion of Limitations

Besides the advantages of model-based evaluations and AUE in general, several results cannot be obtained using these evaluations. Aspects like qualitative and subjective information related to User Experience (UX); e.g. user satisfaction; are hard to measure and to predict by any automated evaluation tools. Thus, today's AUE methods and tools are not able to fully substitute custom usability evaluation at all. Instead, these methods should be used complementary; e.g. when custom usability evaluation methods are too costly or too hard to apply (Ivory and Hearst, 2001).

Furthermore, UI designers usually start by sketching mock-ups of different versions of the UI with their preferred tools in an early stage of development. Functionality for importing from these tools would be desirable to minimize the effort a designer has when evaluating these first mock-ups. However, importing from (and exporting to) these different mock-up tools is an aspect that most AUE tools do not support even though some work has been done on this aspect; e.g. by Harris et al. (2010) and Swearngin et al. (2012).

Finally, the tool and theory itself need to be learned and easily applicable by the designers who have to employ them for validating their UI designs.

### 4.1.2. Cognitive Architectures for Predictive Simulations

In difference to analytical modeling approaches (Section 2.1.4), cognitive architectures provide means to automatically run the developed models and generate user actions from provided input. Usually, these architectures are much more complex than models like GOMS and incorporate findings from different research areas. Their intended fields of use are also much more widespread.

#### ACT-R

In 1983 the Adaptive Control of Thought (ACT) was proposed as a psychological theory for explaining human cognitive activities. Since then, ACT has been extended into different directions; e.g. thinking, language and memory functions of the human brain. Areas of application range from neuroscience to education and cognitive psychology.

The *Adaptive Control of Thought-Rational* (ACT-R) described in detail by Anderson et al. (2004) is an architecture that is also used for describing cognitive aspects of human-computer interaction; e.g. the accuracy and required time for

performing a task. While the *R* stands for rational, the architecture describes rational behavior of the model using a cost-benefit analysis that is used for determining best actions in this *predictive simulation* approach. ACT-R is based on different internal modules and buffers that are able to process information in parallel and serially.

As can be seen in Figure 4.2 a manual module is controlling the hands for interaction, while perception is achieved with the help of a visual module. Rudimentary vocal and aural systems complement the visual module. Further modules consist of an intentional module that processes the current goals and intentions of the user and a declarative module for retrieving information from memory; i.e. facts that are stored in the memory. A central production system is coordinating the other modules. Within each cycle, the central production system recognizes patterns in the internal buffers that contain information about the external world and the internal state. As a result production rules are fired and the internal buffers are updated.

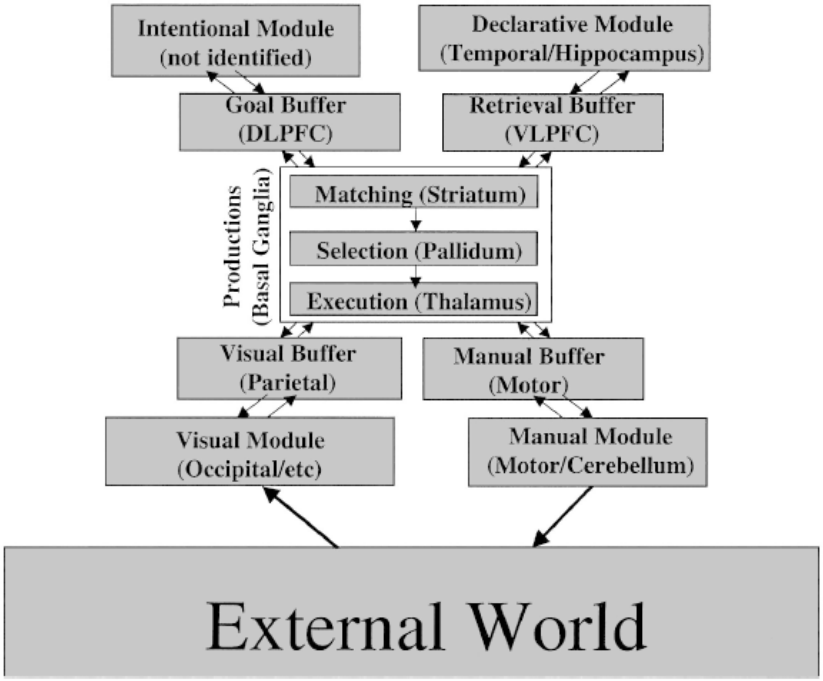


**Figure 4.2.:** Modules, buffers and processes of the ACT-R 5.0 architecture in relation to matching functions of the human brain from Anderson et al. (2004).

All of these modules and processes underlie restrictions for better reflecting human limitations; e.g. a limitation to a single declarative unit of knowledge (chunk) in the memory buffers. Furthermore, within each cycle only one production rule can fire at a time.

While ACT-R implements the limits of human information processing (e.g. restricted working memory), it allows for parallel processing in some cases, making ACT-R models powerful but complex compared with other approaches. Due to this complexity, usually it is only applied by experts in research.

### EPIC and GLEAN

Further cognitive architectures are the *Executive Process-Interactive Control* (EPIC) by Kieras and Meyer (1997) and the closely related GLEAN[5] (Kieras, 2006) that are both used for *predictive simulations*.

In contrary to ACT-R, the perceptual and motor mechanisms of EPIC and GLEAN have an equal status with cognition in accounting for human performance. This is a direct consequence of the assumption that limitations of human ability are not central but structural; i.e. the ability to accomplish tasks might be limited directly by peripheral, perceptual and motor mechanisms but not by a pervasive limit of cognitive-processing capacity. Thus, the cognitive processor is directly surrounded by the perceptual and motor processors that run independently and in parallel (Figure 4.3).

The memory modules are separate parts in the architecture. Outside of the cognitive processor, the long-term memory contains the declarative information and the production memory holds the production rules that are fired by the central processor. A sub-module of the cognitive processor is the working memory. It contains temporary information needed to be manipulated by and to manipulate the production rules. Goals, intentions and sensory input are also stored in the working memory.

For constructing the models in EPIC several steps are required. On the one hand, the task needs to be specified by the designer. For this purpose production rules, which represent the task procedures, have to be specified and task-specific information for perception and actions need to be added. On the other hand, the simulated UI with its own attributes and characteristics for interaction with the processors of the user model need to be defined as well.

Kieras et al. (2001) present a comparison between modeled keyboard and touchscreen interaction using the EPIC architecture. Different findings on the gulf of evaluation and the gulf of execution were drawn; e.g. more opportunities of overlap between perceptual and motor processing with a touchscreen narrow the gulf of execution.

[5] GOMS Language Evaluation and Analysis

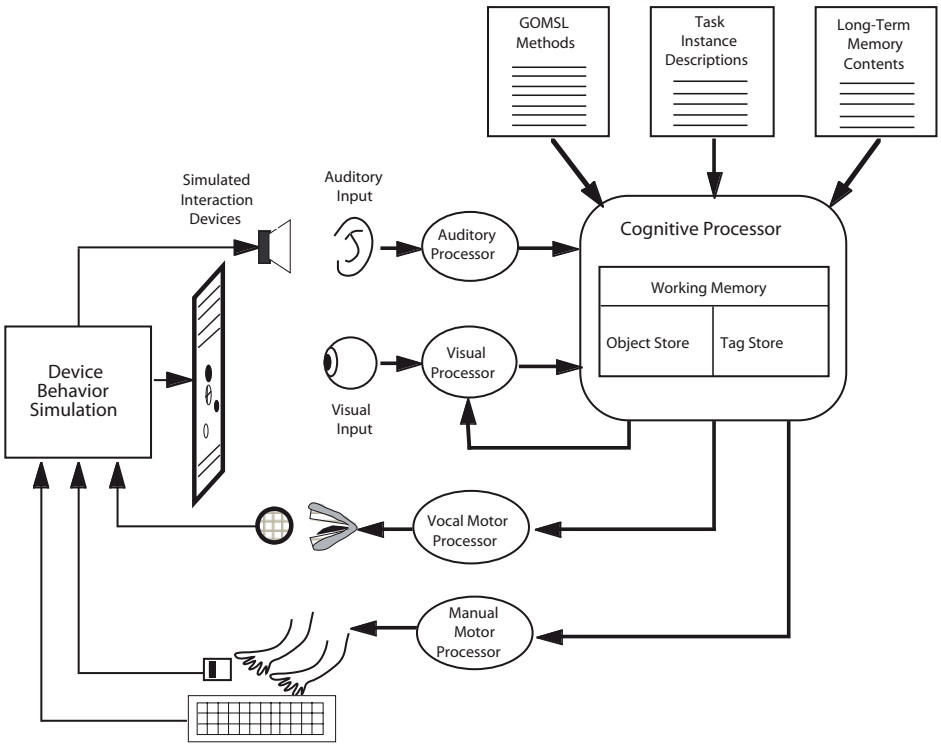


**Figure 4.3.:** GLEAN4 cognitive architecture from Kieras (2006).

Similar to the approach from Amant et al. (2005), GLEAN uses GOMSL[6] for describing the user task and interaction steps. This also leads to a crucial point of this approach, because the whole evaluation process relies on the correct identification of the task. Further challenges are that complex tasks are hard to model and to validate against collected data and the modeling of the simulated device (Kieras and Santoro, 2004).

### 4.1.3. Automated Usability Evaluation Tools

On the one hand, analytical modeling approaches provide methods for describing and predicting user interaction with a specific application. But, these approaches lack means for automation and computer-processable simulation. On the other hand, cognitive architectures provide automation by structuring information and processes within a simulated cognitive model.

Below, automated tools are presented that build upon these approaches to ease creating and running the models for evaluation purposes.

[6] GOMS Language

### AUE Tools based on GOMS, KLM and ACT-R

Tool support for GOMS models is implemented e.g. by USAGE[7] (Byrne et al., 1994). With USAGE an NGOMSL[8] model of the application's UI is generated and executed. This model can be run on a typical set of user tasks and provides execution time and learning time estimations. The system is based on the UIDE approach (Sukaviriya et al., 1993). The approach is limited to relatively low-level goals and tasks, because they cannot be ordered (hierarchically). Nevertheless, USAGE combines the modeling of applications with a formal usability evaluation method and aims at significantly decreasing the effort for creating GOMS models. Further tools based on GOMS are reviewed by Baumeister et al. (2000) according to different aspects like cognitive plausibility, required design information and available modeling constructs.

*G2A* is an approach described by Amant et al. (2005) that allows automatically generating different ACT-R models from a given GOMS model. The approach requires GOMSL specifications that incorporate e.g. mental operators and working memory storage. G2A allows for the evaluation of different ACT-R models by varying mappings from the GOMS model to different ACT-R production rules. The approach benefits from the fact that GOMS models are specified on a higher level of abstraction than ACT-R production rules. The main advantage lies in combining two methods for cognitive modeling with the goal of giving better predictions of human performance.

In a similar way, the *ACT-Simple* framework (Salvucci and Lee, 2003) applied by *CogTool* (John et al., 2004) uses a compilation approach for producing ACT-R models. Additionally, CogTool eases the complexity for creating required ACT-R models by manual demonstration of interaction on a storyboard of the UI (see Figure 4.4). This storyboard consists of mock-ups or screenshots of the UI that are connected by transitions. In the background a KLM model is generated from the input sequences the designer has specified. This KLM model consists of user actions that are translated to perceptual and motor operators like *look-at* and *point* (Card et al., 1983). *Think* operators that represent mental activity of the user are automatically added at every decision point. The resulting model is compiled together with information from the storyboard into an ACT-R model that is used to predict task execution times. The main benefit of modeling with CogTool is that creating the models becomes more intuitive and easily applicable for non-experts. Further information about CogTool is described in Section 6.5 on page 103, because it is applied as part of the proof of concept in the implementation of this thesis.

[7] UIDE System for semi-Automated GOMS Evaluation
[8] Natural GOMS Language

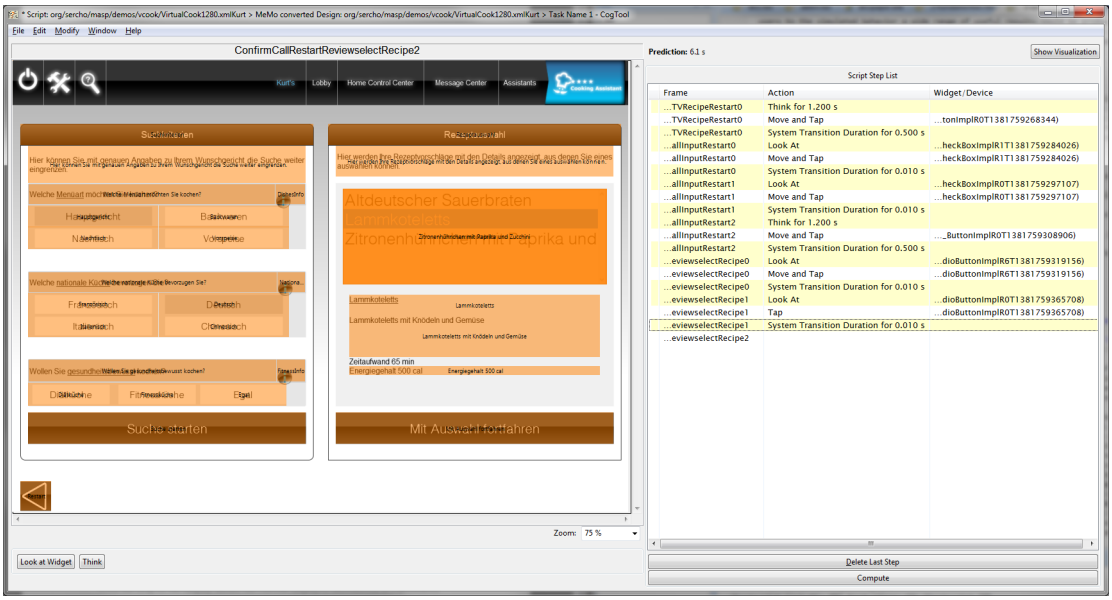


**Figure 4.4.:** Demonstration of an interaction sequence on a screenshot using CogTool.

A first approach towards applying CogTool to the field of mobile devices is described by John and Suzuki (2009). While some of the results are promising, comparisons with real users also revealed that changes to the underlying KLM model need to be made based on expert knowledge to cope with the domain of mobile devices.

### PUM Framework

The PUM framework developed by Blandford et al. (2004) in the Lisp programming language stores a user model, a model of the device and a model of the display. On the one hand, the user model holds information about the user's initial beliefs and initial goals as well as the operations available for interaction. Learning new beliefs is not fully covered in the described approach, because contradictions are strictly solved by replacing old beliefs. On the other hand, the system side is specified through the device model that holds information about the state of the device and actions for updating this state. With the help of the display model the current displayed information is represented. The display model can be seen as a shared resource between the user and the system and is therefore modeled as a belief of the user. Interaction is simulated with the help of operations. All operations that match the goals of the user will be performed. If preconditions prevent the operation from being performed, these preconditions will be transferred to the user as new subgoals to satisfy the preconditions. This way the simulation directly uses information from the application model.

#### AUE Tools Using Simulated Non-Expert Interactions

*CogTool-Explorer* (Teo and John, 2011) aims for predicting human interaction for novice users. The main difference to CogTool is that the designer does not specify an interaction path but provides goals of the modeled user in a textual description. Based on the concepts of *information scent* and *label-following*, which stem from the domain of information retrieval techniques, the user model compares the semantic similarity of a GUI and the goal description. CogTool-Explorer extends the initial CogTool with SNIF-ACT[9] 2.0 (Fu and Pirolli, 2007) and provides three strategies for interaction:

1. Clicking the best matching link.
2. Reading another link on the current page.
3. Returning to a previous page.

These strategies can be modified according to different theories; e.g. for visual perception and perfect memory. Furthermore, the performance of the approach is directly related to information scent scores, which are calculated based on different corpora.

Biswas et al. (2012) describe a simulator for evaluating assistive interfaces for disabled and able-bodied users by predicting likely interaction patterns. The user model is able to perceive information from the device space and maps them to its knowledge from the user space. During this process, learning for first time users is provided and execution time predictions are performed using GOMS. However, the list of low-level device operations and the different phases of simulation (perception and motor behavioral model) needs to be executed manually by the designer and thus lacks automation.

### 4.1.4. The Mental Models Workbench (MeMo)

Main goals of the *Mental Models workbench* (MeMo)[10] for (semi-)automated usability testing are designing and evaluating functional mock-ups of applications. Evaluations are conducted via simulated interaction and allow deriving usability related information such as effects of potential interaction errors. Below, the main models and concepts are described in more detail to coin out the modeling and evaluation with MeMo.

[9] Scent-based Navigation and Information Foraging in the ACT cognitive architecture

[10] The MeMo workbench was initially developed during a research project in 2006 - 2008 by Deutsche Telekom Laboratories, Deutsches Forschungszentrum für Künstliche Intelligenz (DFKI), Fraunhofer Institut für Angewandte Informatik (FIT) and the TU Berlin (DAI-Labor and Quality&Usability Lab). Further development was made during the research project SmartSenior and ongoing research projects by the Deutsche Forschungsgemeinschaft (DFG) and German federal ministries. The author of this thesis took part in the development process and implementation decisions since 2007.

### Working Phases and Models

In general, there exist three phases of a typical workflow with the MeMo workbench:

1. **Modeling** the relevant parts of the application's UI and logic as a deterministic finite-state automaton.
2. **Simulating** the interaction process between user model and application model with the help of probabilities that are calculated depending on user- and application-specific criteria.
3. **Reporting** the evaluation results with the help of an integrated report mechanism by examining logging data from the simulated interaction process and further derived reporting data.

During these phases MeMo makes use of models that are depicted in Figure 4.5.

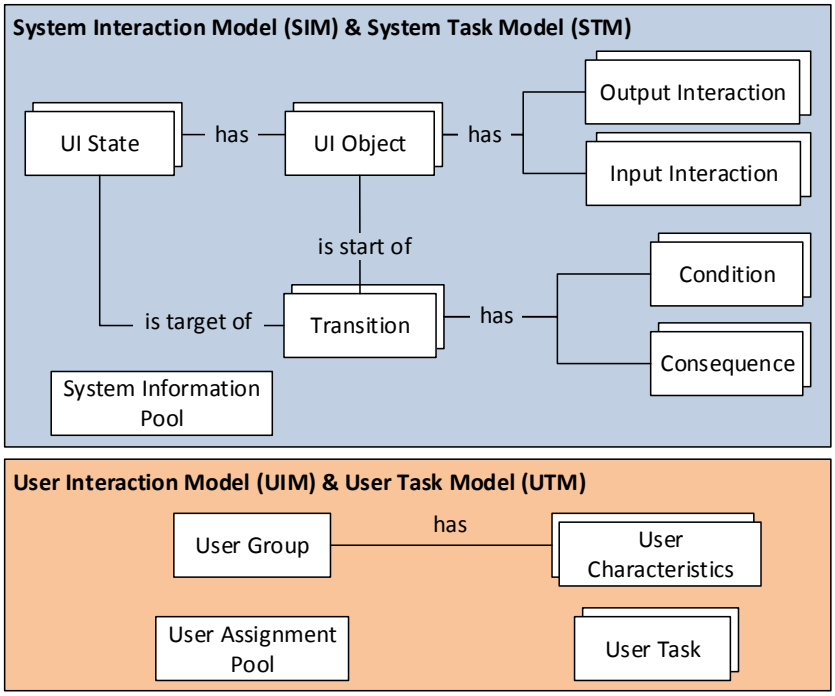


**Figure 4.5.:** Main evaluation models of the MeMo approach representing application, user and task factors.

The **system interaction model** (SIM) shares main aspects with a deterministic finite-state automaton (see e.g. Vidal et al. (2005)) with dynamically annotated probabilities for state changes during simulation. The states of the SIM (*UI states*) are interconnected by *transitions* (upper part of Figure 4.5). Each UI state reflects a unique state of the

application and comprehends required information in terms of all current UI elements (further called *UI objects* within the SIM). By this means, a graphical UI state consists of the background (image), all buttons, text fields, checkboxes, radio buttons and labels with their specific size and position on the screen.[11] During the modeling phase, the designer enhances UI objects with *input interactions* and *output interactions*. Both types of interactions are helper constructs within the SIM that enable the simulated interaction process between SIM and UIM. Via these types of interactions the UIM can apply the concept of *information transfer* using *information assignments* as depicted in Figure 4.6.

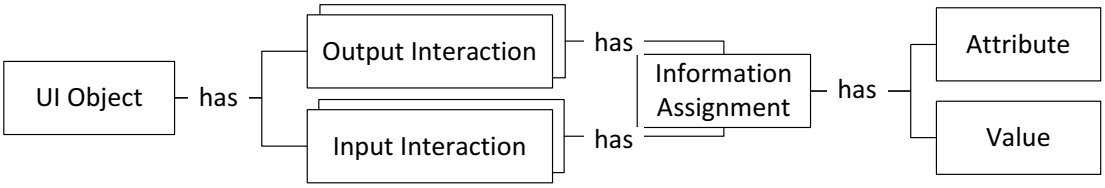


**Figure 4.6.:** Information transfer between SIM and UIM via information assignments that are attached to input interactions and output interactions.

On the one hand, an input interaction is used by the UIM as a specific action on a specific UI object; e.g. a left click on a button. This type of interaction can be enhanced by the designer with an information assignment from the *user assignment pool* which is part of the user task model. Such an information assignment describes a part of the *user task* that the user interaction model tries to exchange with the SIM. On the other hand, output interactions are used for transferring relevant information from the SIM to the UIM, such as the status of dynamic system variables; e.g. the current value of a counter or a text field that can be modified during the simulation and therefore cannot be modeled statically. These variables are stored in the *system information pool* (as a part of the system task model) that needs to be created and maintained by the designer. A single UI object may also have multiple input interactions (e.g. depending on the possible interactions such as left-click, right-click or mouse-over) and output interactions. The interaction logic is encapsulated within transitions that connect one UI state, more precisely one UI object of this UI state, as a source with the subsequent UI state, if a specific input interaction was performed (Figure 4.5). Furthermore, transitions may have *conditions* and *consequences* for constraining the interaction behavior. All conditions must be satisfied for the transition to be executed; e.g. a system variable must have a specific value. Accordingly, a consequence can be used for altering system variables after a transition has been executed.

[11] See Figure A.9 on page 148 for a screenshot of the MeMo workbench displaying a part of a modeled SIM with annotated UI objects.

The main purpose of the **system task model** (STM) is defining the application's interaction logic. This is partly achieved by modeling the transitions with conditions and consequences as described above. This way, an interaction with the modeled application is already limited to logical constraints and represents the intended application behavior as closely as possible. The additional system information pool is modeled by the designer with specific variables for defining dynamic behavior such as different initial values and thus starting points of the simulation.

The **user interaction model** (UIM) is a representation of a user for the purpose of simulating interaction with the SIM according to a specific user task. During the modeling phase the designer may choose specific *user characteristics* from a predefined set that describe the intended user. These characteristics, in combination with properties from the UI objects, can be used to influence the interaction process during the simulation run. Such a user profile is described as a user group to allow variance of the user characteristics during different iterations of the simulation. For specific characteristics (e.g. age) a range can be chosen. Later, during simulation a specific instance of a user from this user group is created with a random value from this range; e.g. age is set to 55 if the range is defined from 50 - 60.

Furthermore, the UIM is capable of gathering information from the current UI state indirectly via output interactions and directly from UI objects; e.g. static text from labels or dynamic text from text fields. For this purpose the basic concept of the *Model Human Processor* (MHP) (Newell, 1990) is adapted to the UIM. Perception, processing and execution are separated into different modules that can be implemented or altered to specific needs of the evaluation; e.g. (Ruß, 2011; Steinnökel et al., 2011). While the output interactions need to be annotated by the designer, further information is directly gathered from the UI objects during the perception process. Depending on the implemented strategy of the processing module the UIM then chooses an input interaction that triggers the associated transition. As a consequence the next UI state is presented to the UIM until the simulation is finished.

The **user task model** (UTM) consists of tasks that the UIM performs with the SIM. A user task has a start UI state of the SIM and a set of initial information assignments to the system information pool to reflect the exact starting point of the simulation from the application's perspective. Additionally, information assignments are provided that the UIM tries to transfer during the simulation. A user task is successfully simulated if all information assignments were exchanged between UIM and SIM.

#### Discussion - Evaluating Adaptive UIs with MeMo

The models required for evaluation using the MeMo approach were developed according to the paradigm of an early evaluation of graphical and voice user interfaces with a limited scope of interaction logic. Providing a model of an interactive application that reflects the functional behavior usually involves much more than just modeling the UI elements. On the one hand, the system information pool and the user assignment pool have to be modeled and maintained during simulation. These pools are constantly altered and checked during simulation; e.g. when evaluating conditions and applying consequences. Especially in case of applications with adaptive behavior or when evaluating medium to large size applications this approach has disadvantages that become obvious during modeling. On the other hand, there exist approaches for importing UI elements from web pages to automatically create MeMo UI elements (as parts of the SIM), which eases the modeling. However, the interaction behavior (modeled in the SIM and the STM) still needs to be modeled by hand and thus does not provide any automation. Especially after redesigning the UI, all changes have to be remodeled in the MeMo workbench as well, which can become a tedious and error-prone task. When trying to preserve the intended interaction logic this could require a complete start over in the modeling process due to the complexity of the SIM.

### 4.1.5. Analysis of Modeling with AUE Approaches and Tools

Below, a comparative overview of the described AUE approaches and several tools is summed up. All of these approaches have in common that a representation of the application and a representation of the user and the task are required. In Table 4.1 their functionality and usage are compared according to the following criteria:

- **Details of user model and task model** describes which models are required for applying interactions.
- **Placement in design process** describes at which stage of development the approach can usually be applied.
- **Separation of models** describes how the designer needs to provide the models and how these models interact with each other.
- **Effort for creating the models** describes required knowledge, provided support and amount of time for conducting an evaluation.
- **Reusability** describes if parts of the models can be reused for further evaluations and thus saves time and effort involved.

**Table 4.1.:** Comparison of different model-based and AUE methods.

| Criteria | GOMS | KLM | PUM | Cognitive Architectures | MeMo |
|---|---|---|---|---|---|
| **Details of User Model and Task Model** | ○ Distinction between goals and tasks is possible<br>○ Goals can contain further subgoals | ○ Goals and tasks are not distinguished as part of the user model | ○ Goals and tasks can be distinguished<br>○ Goals can contain further subgoals | ○ Operate on atomic level of perception, processing and action<br>○ Goals are encoded in atomic information units | ○ User model simulates a user task<br>○ Goals are defined as knowledge in user task |
| **Placement in Development Process** | ○ Main functionality of UI has to be defined<br>○ Requires different interaction ways for reaching the same goal | ○ Main functionality of UI has to be defined<br>○ Not the whole UI has to be finished | ○ Similar to GOMS, device descriptions, information perceived on UI and user's beliefs, goals and operations are required | ○ A functional simulation of the UI and interaction logic is required | ○ A mock-up reflecting the interaction logic and UI is required |
| **Separation of Models** | ○ Goals and procedural knowledge can be modeled independent from the system | ○ Information is directly included in one model | ○ Separate user model and system model<br>○ User model includes beliefs about system operations | ○ Separate user model and system model<br>○ Production rules and declarative knowledge can include prior knowledge of the system | ○ Separate user model, user task model and system model<br>○ User model simulates interaction with system model |
| **Effort for Creating the Models** | ○ Different evaluators tend to create different models<br>○ Tool support simplifies modeling | ○ Tool support is provided by e.g. CogTool | ○ Requires input of user knowledge for interaction with the modeled application<br>○ No automation | ○ Tool support for ACT-R mitigates the effort for creating the models; e.g. Heinath et al. (2007) | ○ Workbench provides tool support for modeling, simulating and evaluating |
| **Reusability** | ○ Some standardized methods can be reused; e.g. GLEAN offers a library for standard interface methods (Kieras et al., 1995) | ○ Is not explicitly intended, though some methods for specific platforms could be reused | ○ PUM framework provides general model parts that can be reused if the same application is evaluated | ○ Production rules, goals and knowledge are usually specific for each evaluation while some generic parts are provided by the cognitive architecture | ○ User tasks and user model can be reused on different system models<br>○ Different user tasks and user models can be simulated with system model |

## 4.2. Model-based UI Development of Adaptive UIs

An approach for addressing adaptivity in UI development is model-driven engineering. This shift from object-oriented to model-driven development is characterized by using the concept of models as a basis for software engineering during different phases. MDE has been applied as a possible approach to reduce the complexity of software development (Vanderdonckt, 2008). The research area of model-based UI development emerged from applying concepts of MDE to the development of (adaptive) UIs.

Model-based UI development specifies information about the UI and interaction logic within the models that are defined by the designer. The main advantage of using *productive* models (Sottet et al., 2008) is that by having a defined syntax and semantics, the models provide computer-processable information about the design decisions. This satisfies **(Req. Inf. 1 - UI Surface Information)** because information about UI surface information of the application from the development process can be used as a basis for applying different AUE approaches. Section 2.3.2 refers to CAMELEON as a tool for describing and structuring the development process and the required models for plastic UIs. Within CAMELEON (and also in UsiXML) the CUI models represent information about UI surface information and are thus applicable for this approach.

**(Req. Inf. 2 - Interaction Capabilities)** states that a connection between UI surface information and interaction logic is required. The approach described in this thesis focuses on task models, AUI models and information from the CUI models or the final UI. Via the (generic) processes of *abstraction* and *reification*, the required interconnection between interaction logic (task model) and UI layout (AUI, CUI and FUI) can be established and thus satisfies **(Req. Inf. 2 - Interaction Capabilities)**. In order to apply these processes, they need to be maintained automatically; i.e. the runtime framework needs to provide means for tracing these interconnections between the models.

**(Req. Inf. 3 - Application State Space)** demands that the models need to be able to dynamically represent required information about the application in all possible states of interaction. Thus, it becomes possible to simulate and evaluate the behavior of the application for various user input and even under a varying context of use. Consequently, this also implies that development approaches that require human effort to establish the links and information transfer between these models during runtime would rarely profit from the proposed benefits of an automated simulation-based approach.

In traditional model-based UI development approaches, task models usually build the starting point of the development process and are further refined with AUI and CUI

models. This does not always have to be the case, as modeling according to CAMELEON also allows deriving each model from any other model using the processes of abstraction and reification in the required sequence. This gives the developers the flexibility to start with the best fitting models; e.g. in case a specific set of models is already available.

Most task notations describe the tasks that the user (*user tasks*), the application (*application tasks*) or both together (*interaction tasks*) perform. This flexibility allows separating information required for input by the user and the actions being performed by the application. As a consequence, this separation of information partially satisfies **(Req. Inf. 2 - Interaction Capabilities)**, which demands for an access to input capabilities of UI elements and their purpose within a specific task.

The Concurrent Task Tree (CTT) notation[12] and similar variants that are applied within several approaches for plastic UIs (see Section 4.2.1) allows the grouping of tasks to *abstract parent tasks*. Additionally, tasks on the same level can be related to each other by (temporal) operators. These operators have a main influence on the interpretation and simulation of the task tree; e.g. by defining that one task precedes another or that several tasks can be performed in parallel. An example of such a task structure is depicted in Figure 7.2 on page 110. Finally, the CTT notation allows the definition of information flow between tasks with the help of *task objects*. By defining which task objects are needed for a task or are produced during the performance of a task, it becomes possible to analyze the exchange of (domain) information between the application and the user. In combination with the temporal relationships this allows for analyzing dependencies between semantic information that are reflected in the application's UI.

Starting with plastic UIs, this section examines current approaches for model-based UI development according to their intended field of use, underlying concepts and value for an AUE.

### 4.2.1. Approaches to Plastic User Interfaces

In Section 2.3.2 the concept of plastic UIs is explained as an approach that tries to preserve certain usability constraints while adapting the UI to different situations defined by the context of use. Below, implementation approaches for plastic UIs are described according to their main features.

[12] http://www.w3.org/2012/02/ctt/ - Accessed in December 2014.

### Rule-based Transformation and Mappings of Models

The runtime environment for plastic UIs described by Sottet et al. (2007) is based on a model-driven approach and enables adaptations of the UI through a model manager, an evolution engine and a model interpreter for concrete and final UIs conforming to CAMELEON. Sottet et al. (2007) define five principles along which their approach can be classified (most relevant concepts are accentuated):

1. An interactive system is **defined by a graph of different models**, which give different perspectives on the system and its functionalities. All **models are available at development time and are kept at runtime**. The **models are interconnected by mappings**.
2. **Transformations** on **and mappings** between the models are models themselves and **can be modified at development time and at runtime**. **Mappings** can **express usability properties**.
3. **Tools from development time** can be used as services during runtime in order to **define new adaptations** on the fly.
4. **Transformations** on the models can be **automated by the system**, semi-automated and manually performed by the designer or the end-user.
5. **Applications** ought to be closed- and open-adaptive in order to **cope with different variations of the context of use**.

At runtime of the application, static rules that were defined by the designer; e.g. according to Bastien and Scapin (1993), are applied to the models through a transformation engine via transformations and mappings. These mappings help the designer in selecting appropriate UI transformations. Figure 4.7 depicts this mapping-based approach between the different models.

At runtime, the mappings allow the adaptation of the UI based on different trigger events; e.g. changes in the provided models of the model manager. The approach also proposes to use the concept of a *meta-UI* to keep the user informed about the current state of the interactive application and to give information why certain adaptations were performed. By following this concept, decisions of the system are made comprehensible for users and they can define or modify own adaptations. In a similar way, rule-based adaptation is also demonstrated by Rossi et al. (2005) for adaptations in context-aware applications.

The MyUI system (Peissner et al., 2012; Peissner and Edlin-White, 2013) also aims for increasing the transparency of adaptations and means for control by the user. This is

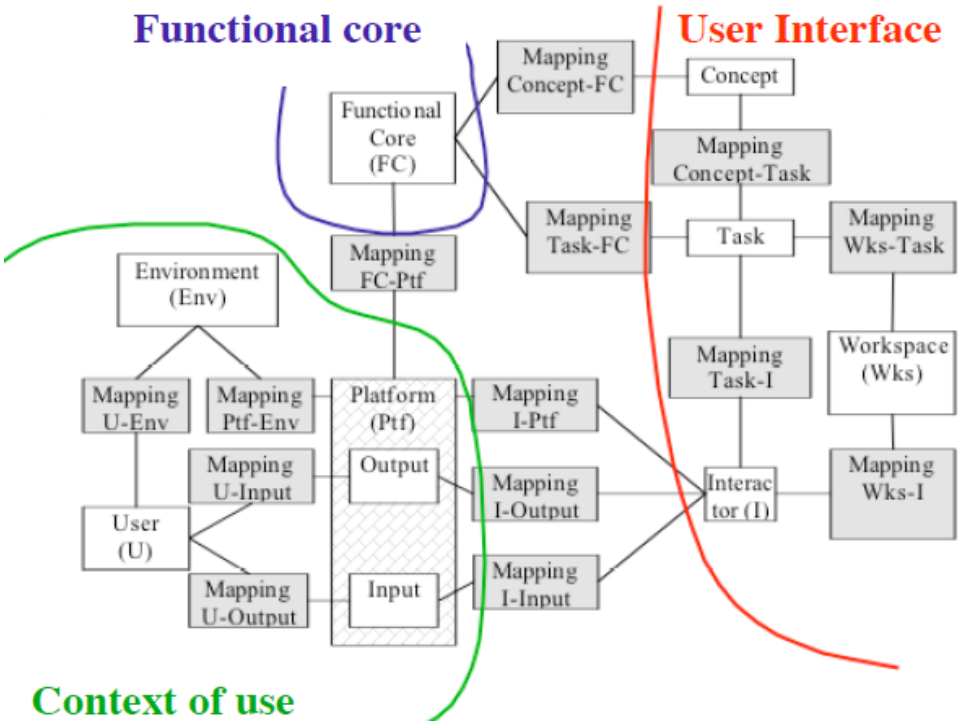


**Figure 4.7.:** Mapping-based approach for UI plasticity from Sottet et al. (2007).

partly achieved by adaptation rendering patterns that draw the attention of the user to adapted parts of the screen. Furthermore, adaptation dialogue patterns provide notifications and give the user control over the adaptation behavior (implicit and explicit).

### Semantically Grouped Task and UI Models

COMET[13] is a further approach to plasticity presented by Demeure et al. (2008). A COMET groups multiple presentations of a particular task that users can perform by encapsulating different types of user interaction and the functional logic of the task. Demeure et al. (2008) denote three facets of a COMET:

- Logical Consistency (LC) defines the task to be represented by the COMET and provides an API to the underlying semantics. Each LC contains one or more Logical Models and maintains consistency between these models.
- A Logical Model (LM) implements (parts of) the semantic API of a LC.

[13] Context Mouldable Widget

- A Physical Model (PM) is connected to a LM and provides an implementation for the LM.

As depicted in Figure 4.8 LC, LMs and PMs are all nodes which can be tagged to filter specific nodes of a COMET; e.g. by interaction path length of a PM.

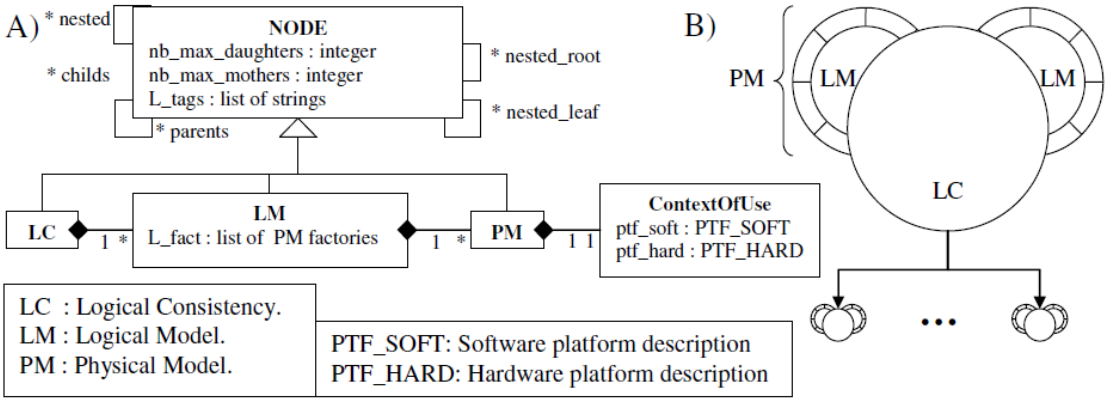


**Figure 4.8.:** UML class diagram (A) and graphical representation (B) of the COMET architectural style from Demeure et al. (2008).

With the help of COMETs, the designer is able to create plastic UIs for different variations of the context of use via tagging the models according to different criteria. At runtime, a graph of different COMETs builds up the whole interactive application. This graph can be manipulated; e.g. by adding or removing PMs.

### 4.2.2. Evaluating Usability during Model-based UI Development

Combinations of model-based UI development and usability evaluation methods have been demonstrated and proved to be helpful in the past. Abrahão et al. (2008) describe how existing usability evaluation methods can be applied to the development process using model-driven architectures.

Figure 4.9 depicts how usability evaluations can be conducted on the basis of already existing information during the development process (conceptual models, model compilation, final UI) and where platform (in-)dependent feedback can be included (system analysis and model compilation).

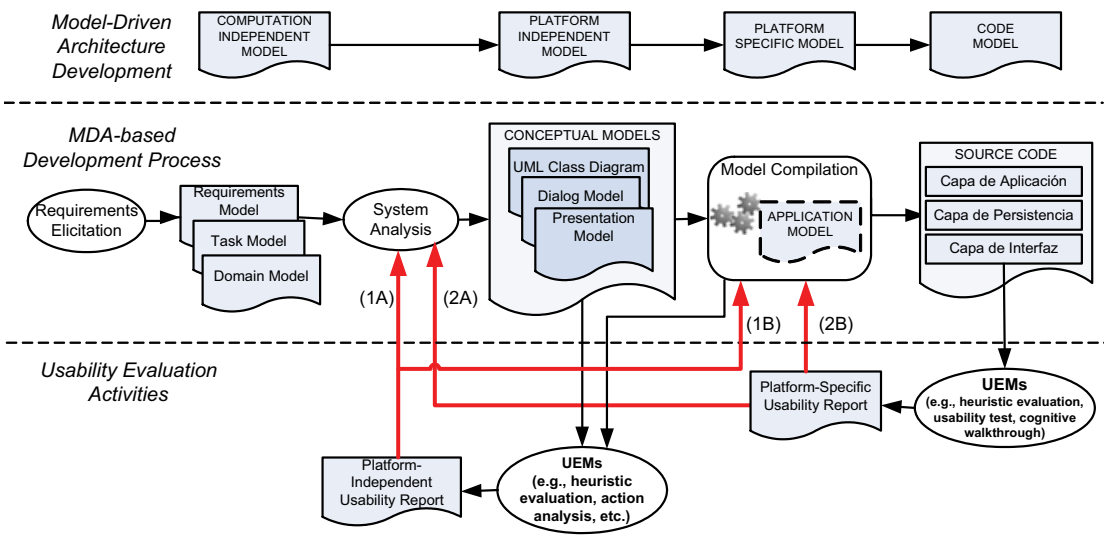


**Figure 4.9.:** Phases of development and usability evaluation activities during model-based development from Abrahão et al. (2008).

### Issues when Evaluating Model-based UIs and Interaction Techniques

Palanque et al. (2011) raise seven main issues for evaluating and comparing different interaction techniques during model-based UI development. These can be summed up to:

1. Most approaches are unspecific and only described in an informal way (e.g. text, captures). For a comparative evaluation, **the UI and the interaction technique have to be defined in a formal way**.
2. **Testing** has to be performed **on a formal model of the UI** with the help of verification techniques.
3. An **abstract** and precise **description of interaction techniques** is required.
4. Usability **tests with real participants need to be recorded** in order to understand coherent interactions.
5. **Interpretations of usability tests are** challenging but **indispensable** in order to optimize the UI.
6. Existing **concepts and interaction techniques need to be reused** and refined for own purposes.
7. **Different interaction techniques need to be compared on the same scenarios** to check if the interaction technique or the UI lead to usability problems.

Especially the first two issues are important for the approach of this thesis as they can be aligned to **(Req. Inf. 1 - UI Surface Information)**, **(Req. Inf. 2 - Interaction Capabilities)** and **(Req. Inf. 3 - Application State Space)** by using productive models of the application.

### Simulation of User Interaction

The TERESA authoring tool (Mori et al., 2004) supports the user-centric development of interactive applications based on task models in the CTT notation and derived UI models (AUI and CUI). The tool enables an early evaluation of the designed applications by means of simulation. For example, the simulation of the task tree enables the designer to detect possible dead ends or unreachable states.

González-Calleros et al. (2013) use MatLab[14] to describe the system and UsiXML to describe the UI for predicting execution times using a cognitive architecture, but the approach is not automated (no direct interconnection of system and UI).

Integration of the simulator from Biswas et al. (2012) described in Section 4.1.2 with the GUIDE system for adaptive UIs (Biswas et al., 2013) aims for predictions of design decisions regarding accessibility especially for impaired users. User characteristics are applied to the adaptation of the interface layout during simulation. However, the UI is provided separately for the simulation using information such as button spacing, button color, button size and text size (Biswas et al., 2013). For this purpose, a UI management layer is described for providing required information from the UI surface for perception and performing interaction with the user model. While this allows for partial automation, integration with development models (e.g. task and AUI) is not targeted by this approach.

Feuerstack et al. (2008) describe an approach that uses an automated simulated user model. It evaluates the final UI provided by a model-based runtime framework and uses an external task description for simulating interactions. The approach requires the designer to manually interpret the simulation results, such as long interaction runs or aborted simulations due to dead ends.

### Supporting the Designer with Guidelines and Requirements

TRIDENT (Bodart et al., 1995) is a tool that provides semi-automated help during development time and makes use of a knowledge-base for offering the designer a set of

[14] http://www.mathworks.de/products/matlab/ - Accessed in March 2014.

appropriate presentation components based on design guidelines. Similarly, the semi-Automated Interface Designer and Evaluator (AIDE) (Sears, 1995) focuses on organizing the controls of an interface by incorporating five metrics (efficiency, alignment, horizontal balance, vertical balance and constraints) in the design process.

The approach presented by Palanque et al. (2011) aids the process of engineering interaction techniques. A special focus is set on the evaluation of interactive techniques with regard to predefined usability and UX requirements. Each interaction technique is modeled using the Interactive Cooperative Objects (ICO) formalism. Because ICO is based on Petri nets, the modeled interaction technique can be simulated and tested automatically. A simulation environment generates model-based simulation log data, which is then evaluated by experts.

González-Calleros et al. (2013) demonstrate how to check the FUI of the application against a set of usability related rules such as color combinations. The results of those checks are then presented to the designer for further optimizations of the UI.

### 4.2.3. The Multi Access Service Platform (MASP)

The MASP[15] is a runtime platform that is based on the concept of executable models for multimodal UIs within the domain of smart environments. This section gives a compact overview of the implementation details from the different models and summarizes the underlying concepts of the development process. Furthermore, this section also demonstrates how and why the application models of the MASP and similar approaches satisfy required information for representing UI information and thus can be used for providing application UI models as a proof of concept in the implementation of the conceptual architecture in Chapter 6. Furthermore, the running example of Chapter 7 is modeled using the MASP development process and executed in the MASP runtime framework (see Section 7.1).

#### Relevant Models and Components of the MASP for an AUE

MASP UIs are defined as sets of models compatible with CAMELEON, spanning task models and domain models, as well as AUI and CUI models and the final UI. Most

[15] The MASP and its underlying concepts are being developed at the DAI-Labor since 2005 and were part of several research projects funded by German federal ministries. Both, the models and the development process, are explained and discussed by Blumendorf (2009) and Feuerstack (2008), who are being referenced for more details when appropriate, because the focus of this thesis lies on the AUE and the interconnection with the required methods and models.

important is the interconnection between the different UI models and the other models via executable mappings that can pass information, which allows tracing the links between the models (Blumendorf, 2009, pp. 121-124). Furthermore, *proxy elements* are applied for making external information accessible to the models (Lehmann et al., 2011); e.g. information from the outside can be observed and reflected in the context of use model.

A main benefit of using the MASP is that the models can be held at runtime to dynamically derive the final UI; i.e. the models are executable. Accordingly, modifications to the models are directly reflected as changes in the UI and, vice versa, interaction with the (final) UI leads to changes in the underlying models. Consequently, the MASP and its API provide required information for application factors. In the following, all required application models for AUE are briefly explained and discussed to make specific how required information is modeled and can be used for evaluation.

The **task model** defines the tasks that the user is able to perform with the application; i.e. the tasks are modeled from the perspective of the application. On the one hand, there exist *interaction tasks* that define direct interaction between the user and the application and on the other hand *application tasks* define interaction logic that is executed without user interaction. Similar to other current approaches, the task model is implemented using the CTT notation with minor modifications (Blumendorf, 2009, pp. 101-103). Most relevant is the introduction of an execution state of the current task that carries information whether the task is *inactive*, *active*, *enabled*, *suspended* or *done*. This information is used to build and reflect on the enabled task set (ETS).[16] Additionally, the task model makes use of input and output variables that can be passed between tasks to modify specific content (e.g. from other models).

The MASP features an executable implementation of **UI models** conforming to CAMELEON. As the goal of the MASP is to provide dynamic multimodal interaction, the UI models are specified independent from the modalities and interaction devices on the level of the AUI model. Furthermore, modality-specific concrete definitions of the CUI models exist in parallel at runtime (Blumendorf, 2009, pp. 110-117). These UI models also encapsulate their current state; e.g. whether they are *inactive* or *active*, which is important when deriving dynamic information for evaluation, because adaptive user interfaces may change depending on the observed context of use. As explained above, changes in the ETS are propagated via mappings from the task model to the AUI model and CUI model and then via platform-specific channels to the final UI. These connec-

[16] An ETS includes all tasks that can be performed in one state of the application at a given point of time.

tions are also used when applying adaptations at runtime and in the opposite direction for passing user input from the final UI to the task model.

The **context model** of the MASP serves two main goals relevant for an AUE approach (for a full description see Blumendorf (2009, pp. 133-136)). During development it is primarily used for identifying relevant context information, meaning the designer defines which information can be observed. Later, during runtime, the main purpose is populating the context model with sensor data and to provide an abstract interface to further models. Similar to most context-aware systems, this information can be used to reflect on the current environment such as information about the user and the available interaction devices. Thus, the context model of the MASP can also be adjusted to simulation needs of an AUE approach and thus serves required information for the context of use factors.

Within the MASP, the **domain model** serves as a dynamic storage for domain relevant information (Blumendorf, 2009, pp. 103-104). During development time the domain model is populated with a structure that holds relevant information within classes, subclasses, objects and attributes. Relevant for an AUE is that this information is used for linking domain data to UI elements for input and output. By this means, submitted interaction data from the user becomes dynamically available to the other models.

#### Model-based Development Process Using the MASP Approach

According to Feuerstack (2008, pp. 62-105) the development process using the MASP approach is structured in a similar way as traditional model-based UI development is conducted. Thus, it can be regarded as a general example for demonstrating how AUE can be placed into model-based UI development processes.

In the beginning of the development process, a requirement analysis regarding the user and the interactive system is performed that targets the functional tasks and objectives of the whole interaction process. After this analysis, the modeling ought to be conducted in an abstract-to-concrete process, which carries the benefit of having the whole system in a single (composed) model at one point of time during development.

By applying this concept to the models of the MASP, a common work flow comprehends the following steps that are applied for each adaptation variant and can be iterated (Feuerstack, 2008, p. 63):

1. **Defining the task model** based on an analysis of the requirements and scope of the application and specifying the general task order and interaction flow.

2. **Defining the AUI model**, the domain model and a service model according to the previously defined task model.
3. **Defining the CUI models** for the specific modalities; e.g. in conjunction with a layout model for GUIs.
4. **Deploying the models** to the MASP runtime framework to perform testing and debugging of the application for refining the models in an iterative design approach.

During these steps several tools can be used, such as the MASP Task Tree Editor (MTTE) (Feuerstack, 2008, pp. 74-78) that is based on the ConcurTaskTree Editor by Paternò (1999). The MTTE supports the designer in defining the task hierarchy via a GUI that allows structuring the tasks according to their (temporal) relationships. Furthermore, the MTTE can be used for simulating task flows that were defined according to a scenario analysis.

The main purpose of such a simulation of the task model by hand is finding out if all scenarios are covered and work as intended; i.e. it is an enhanced consistency check of the task model. Currently, there exists no further support regarding neither usability criteria nor a degree of automation suitable for detailed tests.

Besides the MTTE, Feuerstack (2008) describes several other testing steps during development of the models and specific tool support; e.g. testing the domain model for consistency with the task model and the UI models or using the MASP Builder to create CUI models from AUI models for specific devices. To sum up, the main focus of these tools and testing steps lies on developing the interactive adaptive UIs but they do not facilitate a usability evaluation or an automation of the evaluation.

#### Possible Placement of AUE in the MASP Development Process

An AUE approach based on simulated user interaction and the underlying models conforming to the development process of the MASP can be included best after each cycle of prototyping and developing the models. Only at the stage of deploying the models to the runtime environment all relevant information of the UI surface and the interaction logic are available; i.e. at the evaluation phase during development time.

More specifically, evaluations of the task models or regarding the UI surface information could be conducted in prior stages as well, but would lack the benefits described in this thesis; i.e. automation and decreased effort in providing the models for AUE could not be reached without deployed (executable) models. Nevertheless, the evaluation results

and the interpretation of these can be applied to all prior stages of development and thus serve as input for an improved version or adaptation variant in the iterative design process (Figure 4.10).

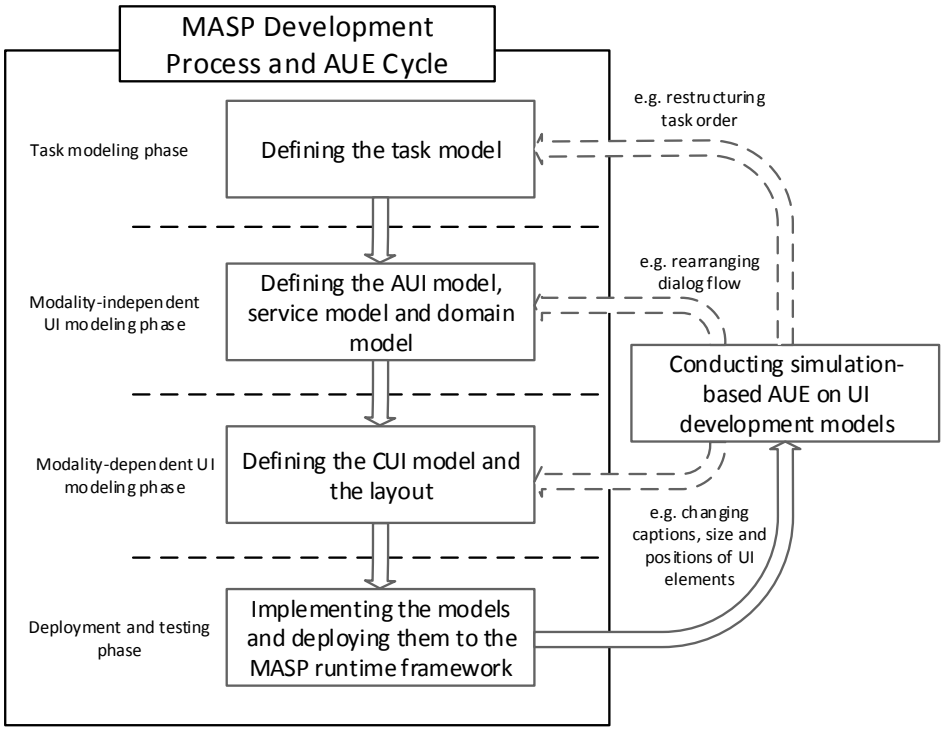


**Figure 4.10.:** Simulation-based AUE can be placed best at the end of each development process when the models are deployed to the runtime framework, while the results of the AUE can be reintegrated to each of the phases for the next development cycle.

### Comparable Platforms for Executable Runtime Models

The context-aware runtime architecture DynaMo-AID by Clerckx et al. (2004) uses task models and context models for generating dynamic UIs. In DynaMo-AID, similar to other architectures, the developer defines the context situations to which the models adapt. In a different approach, Morin et al. (2009) combine model-driven and aspect-oriented techniques to specify and execute dynamically adaptive software systems. The Cumbia platform, presented by Sanchez et al. (2008), is based on extensible runtime models and provides reusable monitoring and control tools.

### 4.2.4. Discussion

All of the described concepts and platforms make use of models representing the interaction logic and its UI information. For example COMET and MASP structure UI and task information explicitly. Furthermore, the described approaches provide development models and concepts for executing the models while observing and representing the context of use. So, theoretically these approaches satisfy **(Req. Inf. 1 - UI Surface Information)**, **(Req. Inf. 2 Interaction Capabilities)**, **(Req. Inf. 3 - Application State Space)** and partially **(Req. Inf. 4 - Context of Use)** from the previous chapter, while information about the user factors and task factors required for simulation are not directly available.

Beyond that, concepts for giving access to the adaptation logic were highlighted within the models and mappings. For example the concept of a meta-UI, applied by Sottet et al. (2007), can be used to make system decisions comprehensible for the user. Thus, such a meta-UI provides means to increase the *self-descriptiveness* and the *suitability for individualization* of an application according to DIN EN ISO 9241-110 (2008) while at the same time several of Nielsen's heuristics can be partially satisfied; e.g. *visibility of system status* and *user control and freedom*. Similarly, Xplain (García Frey et al., 2010) addresses the design of self-explanatory, adaptive UIs in which users get answers about the design and behavior of the interactive system. By this means Xplain makes the design rationale accessible and understandable at runtime. However, interpreting and manipulating such a meta-UI is a valuable concept that is more suitable for adaptability than adaptivity.

Furthermore, several ways are presented how current model-based UI development approaches can be evaluated and which issues an evaluator needs to be aware of - especially testing with formal models rather than rough descriptions. Existing tools, such as TERESA, allow for simulations of the task model and by this means satisfy **(Req. Inf. 3 - Application State Space)** and **(Req. Inf. 7 - Goal-State)**. However, none of the described approaches allows for fully integrated automated evaluations of the UI development models based on simulated interaction. For example, the approach of González-Calleros et al. (2013) does not cover an integration of UsiXML with the system model and with the cognitive architecture and therefore does not provide automation regarding the creation of the evaluation models.

## 4.3. User Modeling

This section provides an overview of the research area of user modeling. A definition of user models is presented, main areas of application are listed and current approaches for structuring and providing user information for specific purposes are described in more detail.

Rich (1979) gives a first definition of the term *user model* and explains the space of user models. The space of user models describes in general *who is modeled by whom*, *what basis is used for the models* and *what user characteristics* need to be modeled. User models are commonly used to provide the system with information about the user and therefore enable the system to be user-adaptive.

Early approaches are based on stereotypes and form the basis for later modeling approaches (Rich, 1979). These stereotypes are composed of attributes from real users that are mapped to different categories and can be ordered hierarchically. Another early approach is the *General User Modeling System* (GUMS) by Finin and Drager (1986). GUMS allows modeling users for recommendation purposes and thereby adapting an application to the user. The GUMS architecture uses stereotypes for modeling the user and in this way provides a knowledge base for the reasoning process. Based on these stereotypes GUMS is able to draw assumptions for recommendation or adaptation.

More recent generic user modeling systems can furthermore incorporate users' goals, beliefs and intentions as well as users' abilities and disabilities (Kobsa, 2001). More general application areas and challenges of user models in human-computer interaction are summed up by Fischer (2001). Fischer also states challenges for increasing the benefits of user modeling; e.g. supporting different modeling techniques in different problem domains and routines for dealing with inadequate model information and privacy issues.

### 4.3.1. Modeling Users with Ontologies

The General User Model Ontology (GUMO) by Heckmann et al. (2005) provides a generic description of user attributes that can be extended to specific needs. As a major advantage this allows exchanging data from a shared user model between different user-adaptive systems through a dedicated server. An implementation using GUMO is described by Heckmann (2006) and specifies different user attributes in OWL[17]. These attributes are classified into different dimensions (Figure 4.11). Within these dimensions further categories can be included.

[17] Web Ontology Language

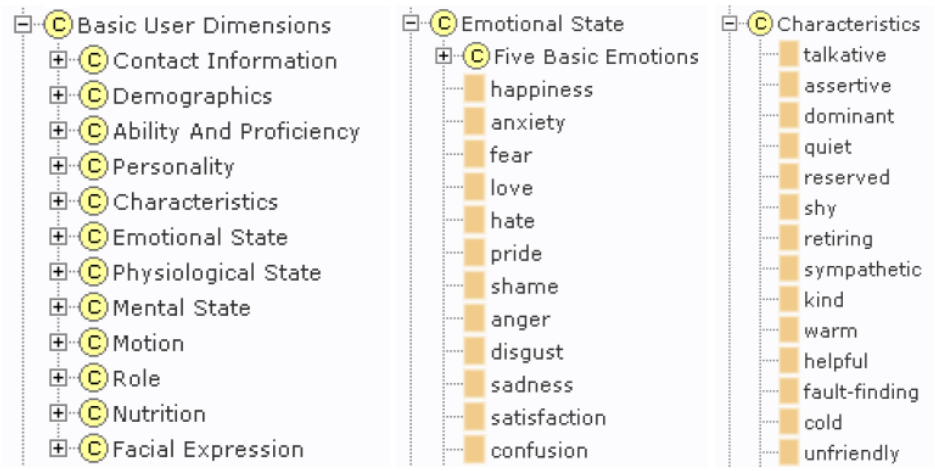


**Figure 4.11.:** Examples of user attributes by GUMO from Heckmann (2006).

#### Discussion

Approaches using ontologies of users provide means to interpret information in an automated way on a semantic level and thus satisfy **(Req. Inf. 4 - Context of Use)**, **(Req. Inf. 5 - Users' Expertise)** and **(Req. Inf. 6 - Users' Abilities)**. Nevertheless, such ontologies can only be tools that need to be used by further components and methods for specific purposes as described by Heckmann (2006).

### 4.3.2. Modeling Users for Simulation and Adaptation

Below, current approaches using different ways for representing users in the application domain covered by this thesis are presented - smart environments.

#### Modeling Users with Stereotypes and Personas

*Amigo* (Vildjiounaite et al., 2007) is a stereotype-based approach for user modeling and user profile provision in smart homes. Initial data for the user model is gathered by the system according to defined stereotypes. Later, the users are also able to modify stored information and adapt them to own needs.

Casas et al. (2008) state that user modeling in ambient intelligent environments is essential because the system has to be aware of the abilities and limitations of its users to enable proper interaction. For this purpose a persona concept is used (Cooper, 1999). The persona concept defines prototypical users based on generic statistics, similar to stereotypes. Figure 4.12 depicts the creation process based on personas.

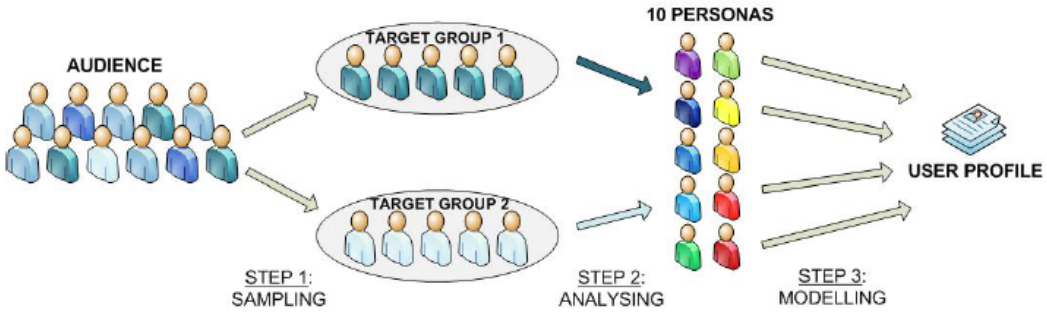


**Figure 4.12.:** Process for modeling users based on personas from Casas et al. (2008).

On top of the server-based user model storage system *Personis* (Kay et al., 2002), Carmichael et al. (2005) describe an enhanced approach for modeling locations of users, devices and sensors in a consistent manner. The authors also state that solving potential conflicts within the models is of big importance while privacy issues are only considered marginally in this approach.

### Tracking and Predicting User Interactions

Buchholz and Propp (2008) propose a Wizard of Oz approach for evaluating applications of smart environments by using a model-based approach. A usability expert tracks user activities with the help of predefined task models. The expert further records if the interaction could be performed with the prototype (using the task models) or not. Based on the results the prototype is implemented to suit the observed needs.

Bezold (2009) uses sequences of user interaction that are fed to trained probabilistic automata to predict likely next user actions. The approach shares similarities with the UIDE system (Sukaviriya et al., 1993) and uses task models based on CTT (Paternò, 1999). This task model can be enriched with statistical data from past interactions or other users for gaining information about most frequently taken steps. Finally, the user's activity is tracked to perform adaptations or recommending next interaction steps.

### Virtual User Models

Virtual user models (VUMs) aim for employing a model of the user into the development cycle of product design (Kirisci et al., 2011). VUMs share aspects with the approach in this thesis; however, they rely on the modeling of the human body and animation of the

user. Especially the animation serves for a simulation of actions with virtual objects (e.g. smart phones) within a simulated environment (e.g. using CAD software). So far, their main area of application is within the automotive and aerospace industry (Kirisci et al., 2011) and the focus lies on accessibility rather than usability issues.

A prototypical approach for inclusive design in consumer products from the VICON project[18] is presented by Matiouk et al. (2013). Similar to Modzelewski et al. (2012), the approach focuses on the design phase of the product using early sketches of the application and then proposes design recommendations regarding accessibility. Manual annotations of UI elements and further information (e.g. size and color) are required by the designer in order to evaluate the prototype using simulation. Furthermore, cognitive abilities of users are not modeled.

#### Discussion

This section presented current approaches for modeling users for adaptation routines in smart environments and for providing early simulations with virtual objects. Besides modeling users' characteristics using stereotypes and personas, approaches were highlighted for tracking user interactions and aligning them to task models. Thus, the described concepts and methods provide means to satisfy **(Req. Inf. 4 - Context of Use)**, **(Req. Inf. 5 - Users' Expertise)** and **(Req. Inf. 6 - Users' Abilities)**. Nevertheless, the described approaches for simulating interaction lack an automation of the provision from application information and require manual annotation.

Approaches using statistical data from observations limit the applicability to the observed situations and the observed behavior; i.e. unknown behavior cannot be simulated or explained. In a similar way, approaches based on prototypical user information only represent certain abstract aspects of the real user and therefore need to be enriched with more detailed aspects during runtime.

## 4.4. Conclusions

In this section main conclusions are drawn from applying required information for simulation-based AUE to related work from model-based UI development and user modeling. Furthermore, basic concepts for the approach from Chapter 5 are sketched.

[18] http://www.vicon-project.eu/ - Accessed in July 2015.

### 4.4.1. Representing Required Information of AUE with UI Models

Based on required information and the related work, the concept of models is chosen for the approach of representing information required for the AUE process. In general, using models within the UI development cycle has several benefits. As denoted in Sections 2.3.2 and 4.2 the main benefits of model-based UI development in the domain of adaptive UIs rely on using an abstract description-language with a specific syntax. Thus, a reduction of complexity and an enhanced reusability of the code and concepts can be achieved (Blumendorf, 2009). In addition, models are also used to separate different domains of information and to describe and compare the whole development process more clearly.

But models are of high importance for (automated) usability evaluations as well, especially when providing a description of the application to a specific AUE method. On the one hand, current AUE approaches require a structured input for the representation of the application and the task to be performed. As described in Section 4.1, models are the most widespread way of specifying this information in almost all approaches because the underlying routines are working on an abstract representation of the real application (see e.g. CogTool). On the other hand, reusability is important when comparing modifications made to the application during different iterations of development. It becomes an even more important factor when combining AUE and model-based UI development of adaptive UIs because of the necessity of evaluating and comparing different or improved adaptation variants under a varying context of use. Evaluating these different versions becomes much easier if the required UI representation and interaction logic does not have to be created several times by hand, but can be conducted on the same development models (see list of costs in Section 4.1). Accordingly, designers of plastic UIs profit from sharing their development models with AUE approaches in order to gain evaluation results during an earlier integration into the development cycle.

Finally, having models presenting required information already provides conceptual mechanisms for satisfying access to required information.

### 4.4.2. Task Models as Representation for the Interaction Logic

The access to the application's task model allows calculating all potential task sequences and therefore enables using the task model for usability evaluation purposes as described e.g. by Paternò (1999). Specifically, by simulating different task sequences, it becomes possible to detect potential usability problems regarding the structure of the interaction

process and to compare different variants of the task flow. The most relevant criteria reflecting these usability related problems are:

- Too long interaction paths,
- Interaction loops,
- Unreachable or untimely reachable tasks,
- Wrong task sequences,
- Dead ends, and
- Missing domain information access.

The detection of such usability related problems is of high importance as they highly influence the perceived usability of the application; e.g. trust in the system, consistency and error tolerance. Even when simulation-based approaches are applied, in most cases the evaluation of these problems requires intensive manual checking by the designer and specific definitions of threshold values; e.g. in case of determining whether an interaction path can be declared as too long. Furthermore, the complexity of task sequences can become high; e.g. when dealing with loops that are defined in the task model. In such cases invariant checking could be applied as a more fitting approach for detecting dead ends or unreachable tasks.

### 4.4.3. UI Models as Representation for UI Surface Information

Complementary to the application's interaction logic, which is incorporated within the task models, the AUI and CUI models provide information about the structure and looks of the UI of an application (see Section 2.3.2). More specifically, AUI models provide a platform-independent description of the UI and therefore hold information for an evaluation on a high level. CUI models describe the UI in much more detail for different interaction platforms; e.g. smart phone, tablet or TV.

#### Information from the AUI Model Relevant for Simulation-based AUE

On the level of the AUI, the designer refines tasks by defining modality- and platform-independent UI elements with an abstract type of interaction. In general, these abstract types can be distinguished between the concepts that are displayed in Table 4.2[19]

[19] As there exist several different naming conventions (c.f. UsiXML), in this thesis the describing names for the different AUI element types are used according to Blumendorf (2009) in order to align the descriptions of the approach with the implementation and evaluation in the latter chapters of this thesis.

together with a short reference to the underlying design question that a designer needs to be aware of when modeling the AUI.

**Table 4.2.:** Different types of AUI elements and underlying design questions.

| Type of AUI Element | Examples of Underlying Design Questions |
|---|---|
| **OutputOnly** | Is the UI element only representing information to the user? |
| **FreeInput** | Does the UI element provide free input capabilities to the user? |
| **Command** | Does the UI element trigger/perform an action of the system? |
| **Choice** | Does the UI element provide different options to the user from which to choose? |
| **ComplexInteractor** | Is the UI element aggregated from further AUI elements? |

Even though the AUI model usually is defined in an abstract way and is not exposed to a (simulated) user, having access to these concepts is highly relevant for an AUE based on simulated user interaction. The AUI model can be used to retrieve design decisions in a computer-processable way. For example the type of the AUI element can be used to semantically group UI elements related to the same task; e.g. displaying a set of information to the user for making a choice.

#### Information from the CUI Model Relevant for Simulation-based AUE

The CUI model defines a representation of the UI for a specific modality and thus provides more details than the AUI model. In this thesis the focus is set on evaluation of graphical UIs. Accordingly, information regarding the UI's surface is provided in the concrete UI (output) model, which satisfies **(Req. Inf. 1 - UI Surface Information)**. Table 4.3 lists examples for required application information that can be obtained from the CUI and FUI model and describes how it is used during evaluation. Usually, AUE approaches based on predictive analytical modeling and predictive simulation do not evaluate features such as font size, color and contrast values. For the sake of completeness, this information is covered as well, which can be used e.g. for automated consistency or accessibility checks. If required, the values can be accessed from or calculated on the basis of the CUI models.

**Table 4.3.:** Examples for descriptions of required information from the CUI and FUI models and their usage within simulation-based AUE.

| Required Information | Contained in Model | Description and Usage for Simulation-based AUE |
|---|---|---|
| **Type of UI element** | CUI Output & FUI | The type of the UI element specifies whether it is a non-interactive element; e.g. a text area; or if it is an interactive element; e.g. a button. Consequently, this information can be used during the simulation-based interaction approach to identify elements for interaction. Furthermore, if the type of the UI element is communicated, the user model can derive typical interaction concepts; e.g. highlighting elements from lists or groups before ultimately selecting the choice. |
| **Size of UI element** | CUI Output & FUI | The size of the UI element is required for perception by and interaction with the simulated user model. It has influence on the accessibility (e.g. if motor abilities are limited) and predicted execution times. |
| **Position of UI element on screen** | CUI Output & FUI | In general, the position of the UI element on screen is required for perception by and interaction with the simulated user. In addition, the position is also required for applying different search strategies as well as models and theories of human perception. |
| **Caption of UI element** | CUI Output & FUI | The caption of the UI element specifies information represented to the user with the help of the UI element. This caption is required for mapping the interaction goals to the specific UI element that has to be used. Furthermore, relevant information that may be required for accomplishing the task can be derived from these captions, such as specific instructions. |
| **Font size of UI element** | CUI Output & FUI | The font size is relevant for the perception of the caption of UI elements and for automated checks of the UI. |
| **Color and contrast of UI element** | CUI Output & FUI | The color and contrast are relevant for the perception of the UI element and can also be evaluated using automated checks of the UI. |
| **Interaction Capabilities** | CUI Input | The interaction capabilities of UI elements are required for simulation-based AUE to determine what interactions can be performed with the specific UI element; e.g. a button may provide left-click and mouse-over. |

The CUI model also defines the input capabilities of the UI in terms of interactions that users may perform with the application and how the input gets processed by the application. This specific information is maintained in the concrete UI (input) model (see Figure 5.2 on page 73). Thus, this model makes explicit when and how users may interact with a specific UI element and which type of interaction modality is used (Table 4.3). According to **(Req. Inf. 2 - Interaction Capabilities)** this is of high importance for simulating the application's behavior and providing information to the simulation-based AUE approach.

## 4.5. Summary

In this chapter related work in automated usability evaluation, model-based UI development and user modeling was presented. Current approaches were discussed with a focus on their advantages and limitations as well as overlaps from these research fields.

As a conclusion, the users' acceptance of interactive applications is highly depending on how well certain tasks in daily routines can be accomplished. There are approaches for adaptive UIs that are already covering different solutions for enabling adaptation of the UI. Most of these systems are making use of a subset of models of the UI, the underlying tasks and the context of use. These models generally serve as basis for decision strategies of adaptations. However, due to the high complexity, the evaluation of the final UI and the applied adaptations with respect to usability improvements for the current user remains a crucial challenge.

On the other hand, the described approaches for user modeling already allow giving detailed information about the user that can be used as basis for the adaptation process. A further advantage is that user models can also be applied in AUE methods for predicting human performance. But, so far, these approaches are usually applied distinct from each other because evaluations are usually performed using specific (cognitive) models and thus do not use existing UI models. As a consequence, developers of model-based adaptive UIs currently have to provide required models of their applications (and task models) for an AUE even though required information already exists. Furthermore, a main drawback of most described approaches is that they demand highly skilled evaluators to build the required models correctly. There only exist a few approaches which try to ease the process of building the models and reuse already modeled components; e.g. as described by Ritter et al. (2006) and by Heinath et al. (2007). The approach described in the next chapter shows how to circumvent this existing necessity and to automate the processes.

As a result from the described drawbacks, the work proposed here aims at a simulation-based AUE of UI models from adaptive UIs during the evaluation phase at development time.

# 5. Approach

This chapter presents an integrated approach that satisfies required information from Chapter 3 by combining development models for adaptive UIs with simulation-based AUE. The approach is explained by referring to each required information and the way how it is covered by simulating interaction on existing UI development models.

At first, the overall architecture for simulating interaction on the models is presented in Section 5.1 and later described in more detail.

The application models which serve for required information for the *application factors* and *context of use factors* are described in Section 5.2.

Section 5.3 addresses the *task factors* and a specific structure for providing information to the simulation. As a basis for this serve already established connections between UI layout and interaction logic that are combined with the proposed AUE method.

In Section 5.4 a representation of the *user factors* is described that is capable of simulating interaction with the provided representation of the application.

Finally, this chapter concludes with a summary of the main points in Section 5.5.

## 5.1. Conceptual Simulation-based AUE Environment

Conforming to the scope of this thesis (see Section 1.3) the focus is set on AUE based on simulated user interaction. The choice of the evaluation criteria builds the starting point of each evaluation; i.e. the evaluator needs to have a goal for the evaluation process. As a result, when using approaches based on simulated user interaction, different paths of interaction through the application can be simulated in an automated way when using *predictive simulation* approaches. Consequently, to describe the application's behavior as closely as possible, this simulation depends on well-defined and structured information provided in the underlying models of the application, the context of use, the user and the simulated user task. This general flow of information between all models and the AUE approach is depicted in (Figure 5.1).

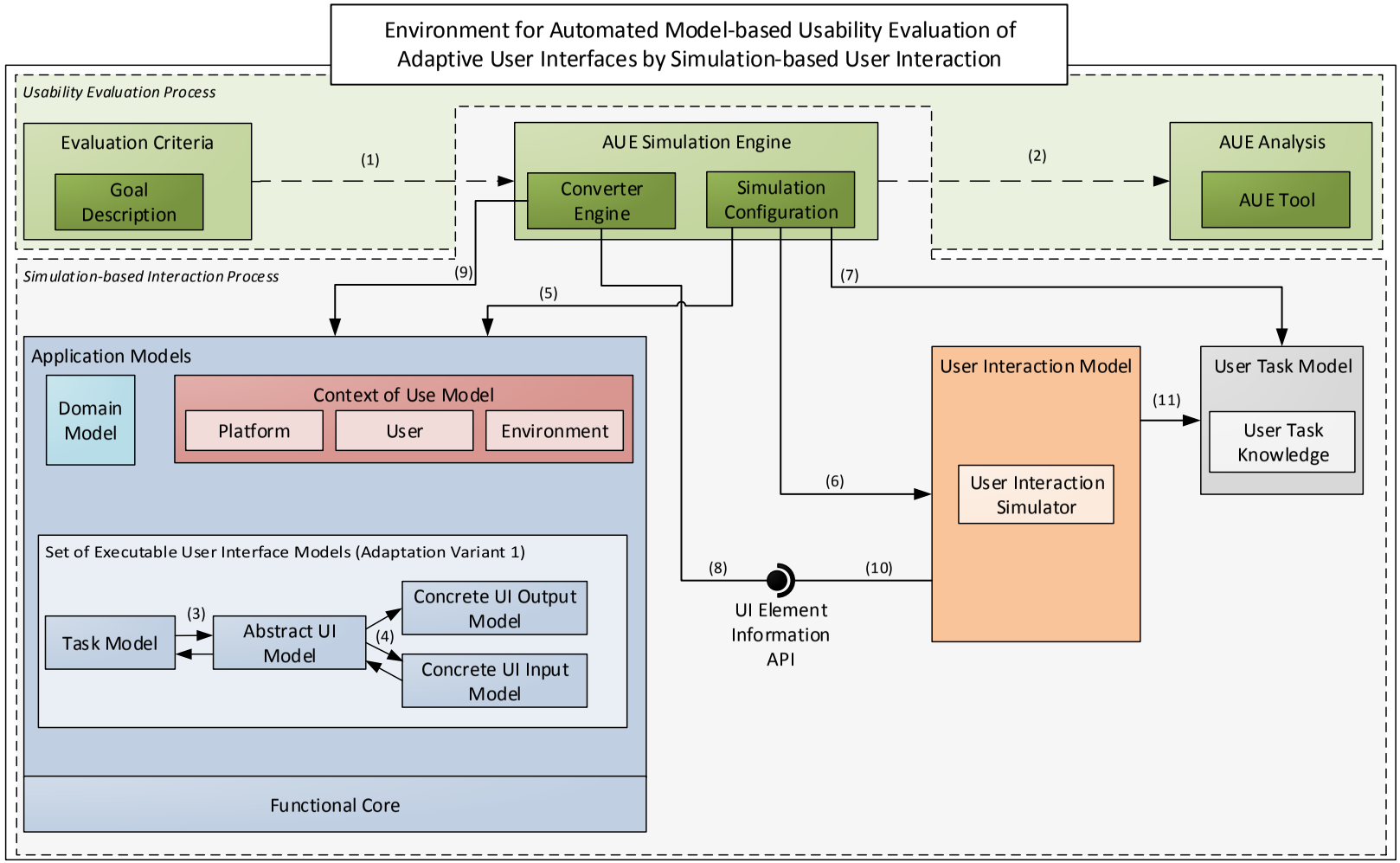


**Figure 5.1.:** Overview of the conceptual environment for automated model-based usability evaluation of adaptive UIs with all models, processes and the required flow of information.

The arrows, labeled from **(1)** to **(11)**, between all models describe the general flow of information. Dotted lines relate to the different phases of the usability evaluation process. They represent points in time, where manual input of the developer/evaluator may be desired; e.g. when defining a user task or when choosing an appropriate AUE tool/method to evaluate the results. Continuous lines link the different models that are required during the simulated interaction process; e.g. access of information from the user interaction model to the user task model.

Below, the meaning of each of these connecting lines is summarized in order to explain the function of the whole architecture on a generic level:

**(1)** depicts the **dependency between the chosen evaluation criteria and the AUE simulation engine**, providing the goal description for the simulation that needs to be defined by the designer.

**(2)** depicts the **outcome of the AUE simulation process** in terms of the simulated interaction path that needs to be analyzed and inspected in the subsequent step using an AUE tool.

**(3)** depicts the **mappings between the application's task model and AUI model** that are used to provide the linkage between UI layout (AUI and CUI models) and interaction logic (task model).

**(4)** depicts the **mappings between AUI model and CUI models** that are used to specify a modality-dependent UI and for the execution of the models during simulation; i.e. transferring user input into the executable application models and transferring the current state of the application's UI as output to the user model.

**(5)** depicts the required **access from the AUE simulation engine (simulation configuration) to the context of use model (reflecting context of use factors)** and thus giving the evaluator the possibility to define specific required situations by modifying either the platform, the user or the environment data.

**(6)** depicts the required **access for the AUE simulation engine (simulation configuration) to the user interaction model (reflecting user factors)** for controlling the simulation process.

**(7)** depicts the required **access for the AUE simulation engine (simulation configuration) to the user task model (reflecting task factors)** for specifying the user task knowledge according to the goal description and evaluation criteria.

**(8)** depicts the required **provision of an API for UI element information (reflecting application factors) by the AUE simulation engine (converter**

**engine)** for providing required UI information to the user interaction model and propagating user interactions.

**(9)** depicts the required **access for the AUE simulation engine (converter engine) to the application UI models (reflecting application factors)** for accessing required UI information and propagating user interactions.

**(10)** depicts the required **access for the user interaction model to an API of the UI element information (reflecting application factors)** for accessing required UI information during simulation.

**(11)** depicts the required **access for the user interaction model to the user task model** for accessing required user task knowledge during simulation.

The described conceptual environment allows providing usability evaluation criteria and an AUE tool for analysis. During the simulation-based interaction process, the AUE simulation engine accesses and modifies information from the provided models. Furthermore, information is shared between all models required for simulating a specific task with a specific user interaction model and specific application UI models in a specific context of use. Finally, the resulting simulated interaction path gets analyzed depending on the specific AUE analysis tool.

#### Derived Evaluation Results from Simulated Interactions

Meaningful usability criteria, as stated in DIN EN ISO 9241-110 (2008), that have proved to be useful in summative usability evaluation, can be derived from the resulting simulated interaction paths of the described architecture:

- **Efficiency** can be predicted and compared e.g. in terms of logged interaction turns, task execution times or the number of non-goal-driven interaction steps.
- **Effectiveness** can be predicted and compared e.g. by measuring task success rates; i.e. the ratio of successfully accomplished tasks to overall simulated tasks.

Due to their predictability, these measures are well-suited for a comparative analysis of different design alternatives or different adaptation variants when using predictive simulations or analytical modeling approaches; e.g. shorter task durations are not generally preferred depending on specific user needs and abilities.

The simulated interaction path is designed to provide data from different user models, different applications or adaptation versions and different tasks, which makes it possible

to compare the results of the simulation directly. In addition, these simulated interaction paths can also be used for a formative usability evaluation; i.e. finding specific interaction problems or inadequate application responses.

Starting with the application models, which represent the application factors and the context of use factors, in the subsequent sections each model of Figure 5.1 is described in more detail.

## 5.2. Application Models

Conforming to the conclusions drawn in Section 4.4 the application models of the approach are depicted in Figure 5.2.

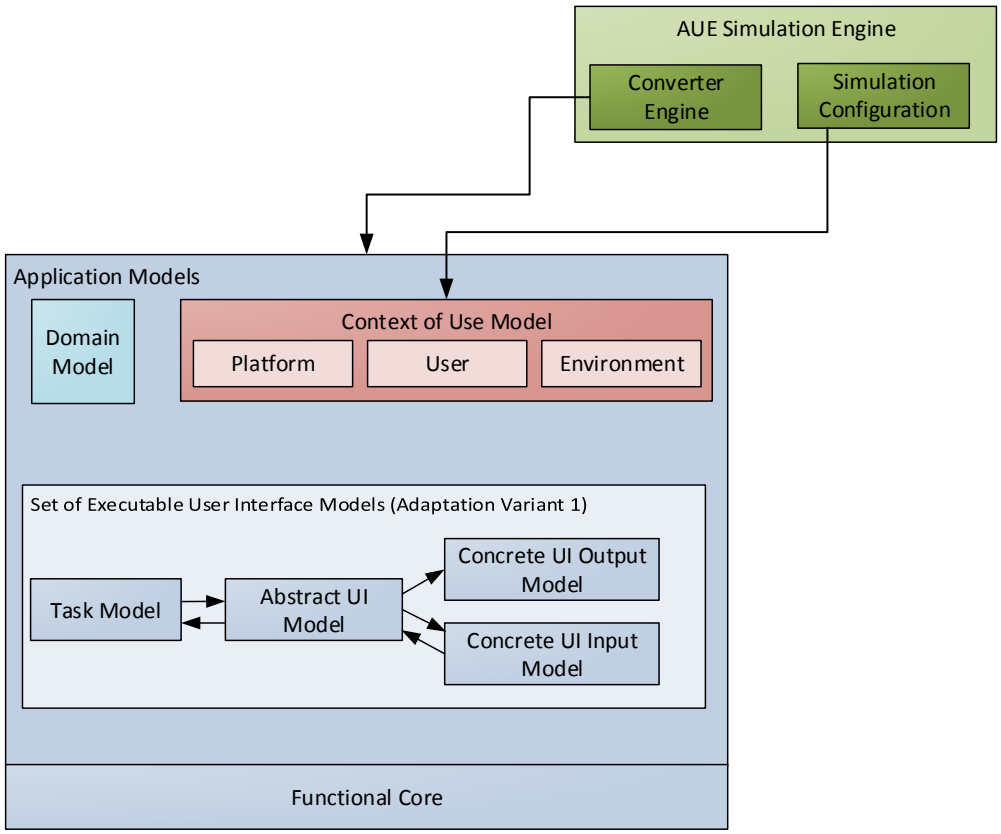


**Figure 5.2.:** Application models consist of the UI models, the context of use model and a domain model. A functional core provides means to maintain the application and adaptations. The simulation engine accesses the application models and sets the specific criteria for simulation.

As the approach aims for evaluating existing UI development models within a runtime framework these models need to get provided. Consequently, they are accessed by the conceptual AUE simulation engine to provide the simulation-based approach with required information about the UI, the interaction logic and the context of use from the application models.

More specifically, these application models conform to:

**Executable UI models**

Each adaptation variant of the UI consists of a set of tasks defined in the task model that is connected via the AUI model to the CUI models and a domain model. In this approach the CUI model consists of a *concrete UI output model* for defining the UI structure and its elements in the way how they are presented to the user and a *concrete UI input model* for defining the actions that can be performed with the UI. While the separation of input and output CUI models is not mandatory for the approach and is not stated in CAMELEON, it has the advantage of separating the interaction logic of UI elements from the surface information. This allows accessing the models separately during simulation; i.e. when perception of information is simulated and when interaction with a specific UI element is simulated. Finally, with the help of the interconnections, information can be passed in both directions and thus keeps all models in sync.

Furthermore, by maintaining the mappings between tasks and their AUI and CUI representations it becomes possible to trace modeled information of each UI element back to the task and thus gathering its purpose for interaction in the current state of the UI and task. As a consequence, the provision of UI surface information according to **(Req. Inf. 1 - UI Surface Information)** and **(Req. Inf. 2 - Interaction Capabilities)** in combination with accessible interaction logic conforming to **(Req. Inf. 3 - Application State Space)** enables simulation-based AUE on the application UI models.

A converter engine, as part of the AUE simulation engine, accesses required information from the UI models and presents it to the simulation-based approach in the specific required syntax of the evaluation method.

**The context of use model**

For the scope of the approach it is not specifically important how the adaptations are defined or how they are executed. More relevant is providing a context of use that can be edited to the needs of the evaluation.

**(Req. Inf. 4 - Context of Use)** demands for a model of the context of use that is connected to the models of the application and can be queried and altered by the AUE simulation engine. This way, the designer is able to simulate different situations defined in the context of use to which the functional core adapts the UI accordingly (compare e.g. components in the DMP layer in CAMELEON-RT). This, in turn, becomes accessible to the simulated user interaction model as a new variant of the application's UI models.

The approach allows simulating the adaptation behavior of an application with fixed or dynamic, but observable, parameters relevant to the context of use and for an evaluation. Especially, comparisons of different adaptation strategies can be evaluated this way. Thus, the concept of plasticity can be targeted by evaluating adaptations that occur during simulated interaction.

**The domain model**

The domain model is required to provide domain-specific information that is reflected in the UI and thus needs to be accessed by the simulation-based approach; e.g. for retrieving all potential domain information of the application.

Finally, in addition to the application models, the functional core of the application is required for maintaining the application and e.g. adaptations. It is mandatory to the approach to ensure evaluation of the interaction behavior of the application, but it is not directly used or accessed by the AUE simulation engine, which relies on the models only.

## 5.3. User Task Model

Conforming to **(Req. Inf. 7 - Goal-State)**, in the approach a user task model is required to simulate interaction between user interaction model and application models. The user task model specifies information to be transferred between the user interaction model and the application models for changing the application's state towards the goal of the task. In the following, this required information for accomplishing a user task is called *user task knowledge* (UTK).

Below, the structure of the UTK is described together with the process how it is provided to the simulation-based approach (Figure 5.3).

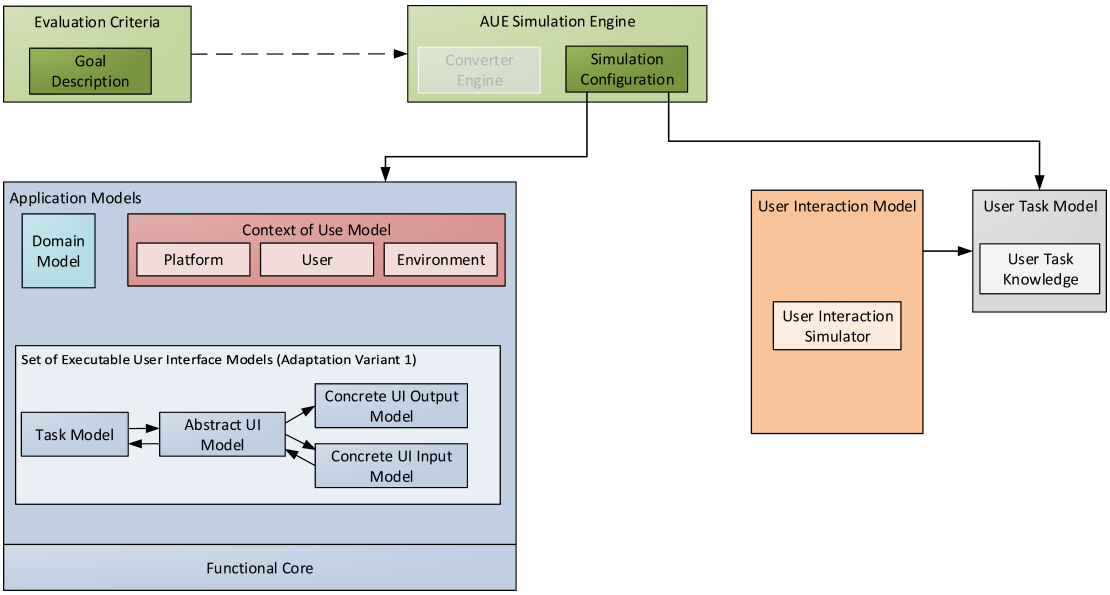


**Figure 5.3.:** The user task model consists of user task knowledge that is used by the user interaction model. The simulation configuration provides the required user task knowledge from the goal description and accesses the application models for providing further information.

### 5.3.1. Modeled Information in the User Task Knowledge

The UTK is accessed during simulated interaction by the deployed user interaction model. For this purpose, the UTK is structured with the help of attribute-value pairs. These attribute-value pairs can be ordered as a set, a list or hierarchically. When provided as a set there is no specific order defined. But when provided as a sorted list the attribute-value pairs define a specific sequence of information that needs to be exchanged. Finally, in a hierarchical order the attribute-value pairs represent connections and dependencies between information; e.g. tasks that have subtasks.

Furthermore, each attribute-value pair specifies information for identifying a UI element for interaction. The value provides a specific assignment to the attribute and thus refines this information; e.g. the caption of a UI element that needs to be interacted with. By having this broad definition of attribute-value pairs it becomes possible to simulate interaction on various levels of detail.

On the one hand, such an attribute-value pair can be instantiated with an attribute that reflects a category of information while the value is a specific instance of that category. An example gives a radio button group reflecting the attribute and the user interaction model needs to select a specific value, which is described by the latter part

of the attribute-value pair. Such an instantiation allows simulations on a semantic level close to the UI of the modeled application.

On the other hand, the same instantiation can be defined more technically by using information from the UI development models; e.g. the attribute is used as an internal identifier of the radio button group and the value as an internal identifier of the specific radio button element that is required for selection in the current task. This more technical instantiation is comparable to a simulation of application task models described in Section 4.4.2 but rather uses their information as an additional source for automatically retrieving the UTK. In summary, both approaches lead to the same consequence of having required information for choosing between UI elements for interaction. Figure 5.4 depicts an example using a tree-like structure of the UTK.

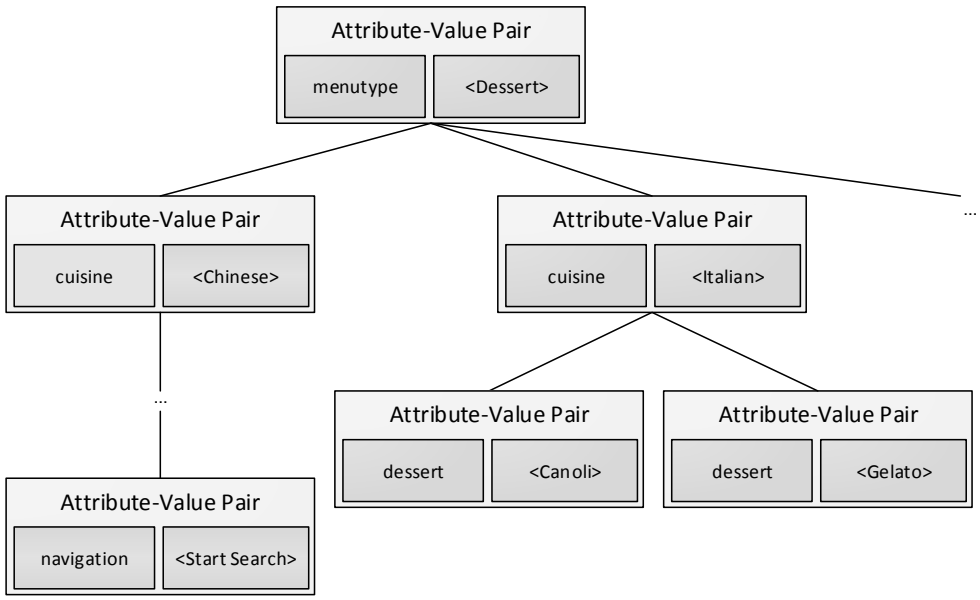


**Figure 5.4.:** Example of a hierarchical tree structure of attribute-value pairs representing user task knowledge required for accomplishing a user task.

Next, a description of processes for providing required information for the UTK is presented.

### 5.3.2. Providing the User Task Knowledge

According to **(Req. Inf. 7 - Goal-State)** two ways for simulating with the initially required UTK can be applied. The first alternative represents a list of predefined interactions (encoded as attribute-value pairs) that can be provided manually by the designer or

automatically by using the development models via a mapping process. Both approaches are applied via the simulation configuration of the AUE simulation engine and use information from the goal description that is provided from the usability evaluation criteria (Figure 5.3). The second alternative also relies on mappings that are obtained using an integrated simulation approach that requires a specific goal description and tries to find a solution during interaction.

Two different kinds of interaction steps need to be distinguished in order to explain the applied mapping process:

**Manipulating interaction steps**

In general, these interaction steps require the user interaction model to enter information that gets processed by the application. On the level of the AUI model they are characterized by interaction with UI elements having the AUI types *freeInput* or *choice* that are used to manipulate variables of the application. Internal identifiers of these AUI elements can be used as attributes for describing the UI elements that the user interaction model needs to interact with. In that case required information can also be provided via the simulation configuration. Accordingly, all relevant attributes for the attribute-value pairs can be identified. In case of *choice* elements the specific assignments; e.g. information for captions of radio buttons are usually modeled in the domain or CUI model, can be used as values for completing each attribute-value pair (Figure 5.5). If applied in an automated way (e.g. when specific information is not needed or available), such an approach produces arbitrary information (i.e. information that is not meaningful in a specific task) for manipulating interaction steps. Thus, automatically filled manipulating interaction steps require manual validation, if that is essential to the evaluation goals.

**Navigational interaction steps**

These interaction steps describe actions by the user interaction model which let the dialogue proceed to a subsequent UI screen; e.g. when selecting "Next" or "Done" in order to proceed in the dialogue without providing new information relevant for the current task. Navigational steps are basically acts of confirming already selected information, but usually do not have a semantic relation to the goals of the task. If not modeled explicitly, these navigational interaction steps can also be identified using their AUI type via access through the simulation configuration. The AUI type *command* indicates such navigational input. Accordingly, all corresponding UI elements and their captions can be derived by using mappings from the AUI model elements to their corresponding CUI model elements (Figure 5.5).

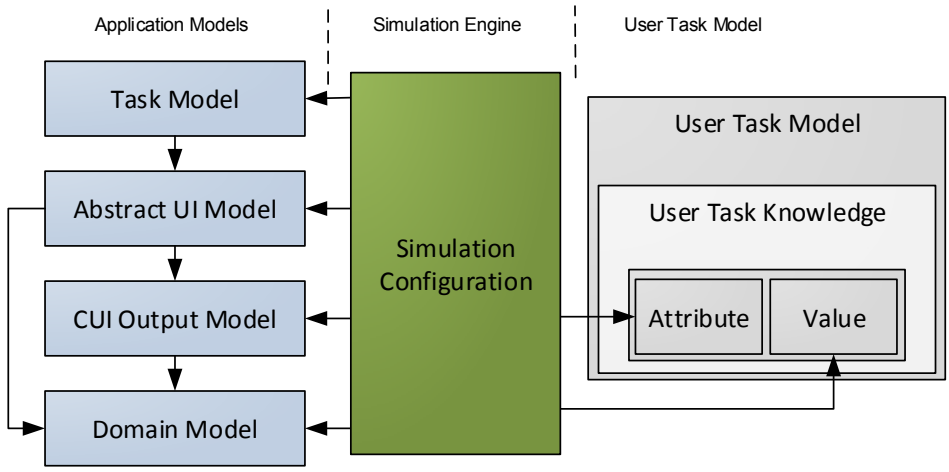


**Figure 5.5.:** The simulation configuration uses AUI and task model information for specifying attributes and domain and CUI model information for specifying values of the UTK.

In order to begin the simulated interaction process an initial set of information for the UTK is required.

1. **Predefined interactions**
   For this alternative the designer specifies each attribute-value pair required for the whole interaction process. This may require a lot of input by the designer, especially when evaluating multiple tasks with multiple adaptation variants. Nevertheless, there are certain use cases where this may be desired; e.g. when simulating a specific path, potentially including simulated (known) interaction errors, and inspecting the behavior of the application. Such an approach is beneficial when evaluating e.g. the error tolerance of an application (see Table 2.2) or error prevention functionalities (see Table 2.3).

   In the approach of this thesis, which makes use of the UI development models, this process can also be automated by calculating possible solution paths based on an existing task model of the application. The advantage of this option is that executable task and UI models conforming to CAMELEON inherit the required information and can be simulated. Consequently, the UTK can be initialized by deriving required information from the models of the application. Below, a generic description of this practice is summarized.

   If possible, different paths to a predefined goal state have to be generated, which present different versions of the UTK. Each resulting version of the UTK is called a *solution path*. Each solution path is being calculated by traversing the task model

of the application and by using mappings to the AUI and the CUI models. Due to the tree structure and the mappings between the models, the ETS is calculated based on the temporal operators between the tasks. This significantly lowers the complexity of the calculation process as the ETS is only a subset of the whole task model. More specifically, the following steps are required for determining a solution path based on the application's task model:

i. At first the goal description needs to be mapped to a specific end task or end ETS that needs to be reached during simulation.

ii. Then, starting with the initial ETS of the application's task model and by executing the active tasks, a graph of all reachable application states is being built, given the assumption that the graph is finite. Each node of the graph represents an ETS, while each edge represents a performed task. Thus, for any end task that conforms to the goal description, all potential solution paths which include all tasks the user interaction model has to accomplish, can be obtained. Consequently, as an intermediate result, each solution path contains all subtasks the user interaction model needs to pass through.

iii. As a next step these tasks need to be mapped to required interactions of the user. For this purpose, the mappings (see Figure 5.2) between task model via AUI model to the CUI model and domain model (see e.g. CAMELEON or UsiXML) are used.

iv. Finally, all attribute-value pairs from the navigational interaction steps are added together with the attribute-value pairs from the manipulating interaction steps to each solution path in a sorted sequence. In the end, this resulting solution path is transferred as UTK to the user task model ready for simulation.

If not altered by further concepts, this approach has the consequence that the simulated user behaves according to this predefined UTK; i.e. performs correct interactions in the given order. If this approach is combined with a further mechanism for simulating interaction errors (outlined in Section 8.3.2) the interaction process can also be adapted. As a consequence of such simulated interaction errors; i.e. the user leaves the calculated solution path; a new solution path beginning in the current state of the application's task model needs to be calculated. After that, this new solution path has to be transferred to the user interaction model and replaces the existing one in order to maintain the goal-directed interaction. In case no such interactions are found, before ultimately canceling the task, path

search can also be applied in order to look for interactions leading to a subgoal, or another state on the goal-directed path as proposed e.g. by Jameson et al. (2007).

2. **Predefined goal description**
   This alternative is mostly suitable for modeling novice users or exploratory behavior and the resulting interactions might not lead to the intended goal state or can be aborted. The designer only specifies an initial and the final state of the application that needs to be reached. This specification can be a set of attribute-value pairs. The underlying concept for retrieving information is the same as described above. But instead of a sorted sequence of attribute-value pairs, this set is not sorted and only contains manipulating interaction steps. By this means, the user interaction model still has access to the most relevant information of the task, but needs to match each attribute-value pair with information provided in the UI.

   After the simulation is started, the purpose of this approach lies in determining whether the user interaction model is able to achieve this goal state with the provided UTK (similar to the PUM approach and CogTool-Explorer described in Section 4.1.3).

In the next section the user interaction model and the process of simulating the user task is described.

## 5.4. User Interaction Model

The scope of this thesis and required information defined in Chapter 3 focus on a simulation-based approach for the AUE process. For this purpose a representation of the user is required that is able to simulate interaction based on users' expertise **(Req. Inf. 5 - Users' Expertise)** and abilities **(Req. Inf. 6 - Users' Abilities)**.

On the one hand, simulations of human performance can be done on the cognitive level with architectures like ACT-R or EPIC, which, however, require a detailed and task-specific model of the user and the cognitive procedures (see Section 4.1). On the other hand, a goal of the AUE approach is evaluating effects of different adaptations; i.e. combinations of different UIs, user-specific characteristics (expertise and abilities) and the context of use. For this purpose, a user interaction model on the intentional or behavioral level (see e.g. Eckert et al. (1997)) would be sufficient as there is not much additional information about the internal processes and procedures required. Such an approach allows transforming the simulated interaction paths into ACT-R for further refinement and analysis; e.g. using the ACT-Simple framework (see Section 4.1.2). If

such transformations to ACT-R are applied, the predictions of execution times are only valid for error-free interaction paths. Consequently, the simulated interaction paths can be evaluated according to e.g. predicted execution times or estimations of learnability. But, in order to predict these criteria, elements from cognitive theory need to be included or applied on top. Thus, a combined approach of behavioral and cognitive level is applied conforming to the defined required information.

Below, the applied concept of information exchange between the user interaction model and the provided information of the application is described.

### 5.4.1. Information Exchange between User Interaction Model and Application Models

The approach of exchanging information between user interaction model and application models makes use of the concept of *speech acts* (Searle, 1969). The underlying assumption is that a speech act can be referred to as a dialogue between user interaction model and application models on the basis of a communicative intention (*illocutionary acts)*. By this means, interaction can be simulated in a goal-directed way by sharing information between the models.

On the one hand, the dialogue can be described from the application's point of view. For this purpose, the UI of the application presents information in two ways:

- *Requesting* the user interaction model for input via UI elements that demand for an interaction; e.g. text fields, buttons and checkboxes; or
- *Informing* the user interaction model about the current state of the application and options for input via non-interactive elements; e.g. via describing text areas and labels.

On the other hand, the execution of an interaction by the user interaction model forms the opposite perspective. Actions of the user can be seen as:

- The response to a request of the application by *informing* the application about the result of the processing in the user interaction model via interacting with a UI element that transfers required information; e.g. selecting a radio button; or
- A *request* for information by the user interaction model via an interaction that leads to an updated UI; e.g. filtering or searching for information.

In both cases, the provided input and output capabilities of the application's UI have to be regarded because they add relevant information for the interaction process; e.g.

there can only be one radio button selected from a group while there can be multiple checkboxes selected. In this generic state, the proposed information exchange based on speech acts does not directly require any sensor systems for perception or motor systems of the user interaction model. This has the advantage that simulations on the behavioral level still can be conducted; e.g. solely on the basis of information from the user task models and all available information from the application models. Beyond that, the following section shows how to apply sensor and motor processes to reflect the nature of the combined approach of behavioral and cognitive simulations.

An interface of the converter engine (*UI element information API*) is used during simulation of the approach (Figure 5.6). On the one hand, the UI element information API is used for providing required information from both CUI models to the user interaction model. On the other hand, it is also used for triggering interactions via the converter engine on the application's concrete UI output model and thus acting with the application on the level of the UI development models (instead of the final UI).

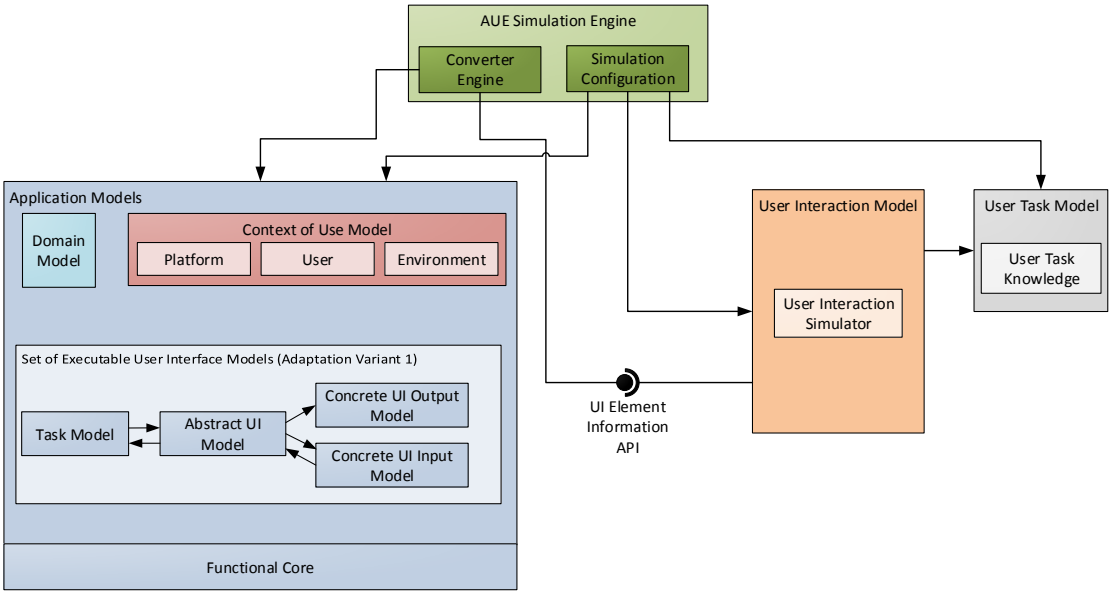


**Figure 5.6.:** Information from the CUI models is transferred by the converter engine to the UI element information API that is used for simulating interaction with the user interaction model.

For this purpose the UI element information API offers for each UI element of the application models that is currently present on screen:

- Type
- Caption

- Interaction types
- Size
- Position on screen
- A method for performing an interaction on the UI element

Next, the internal processes of the user interaction model are described. Furthermore, it is explained how required information for the UI element information API is derived from the application models and provided to the user interaction model.

### 5.4.2. Structure of Processes in the User Interaction Model

The user interaction simulator, as part of the user interaction model of this approach, conforms to the structure and concepts of the MHP (Newell, 1990) depicted in Figure 5.7.

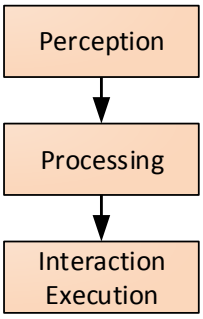


**Figure 5.7.:** Processes of the user interaction model adapted from Newell (1990).

In general, the MHP structures the internal processes into *perception* of information (perceptual processor), *processing* of information (cognitive processor) and *interaction execution* (motor processor). This generic structure is used by several cognitive architectures and thus allows for an easy interconnection of the described user interaction model with existing architectures or even exchanging the processes between different approaches. Below, each of these processes is described in more detail and a special focus is set on the required information exchange for allowing interaction between the models.

#### Perception

A simulated perception process comes first within each simulated interaction step. The perception is based on the modeled description of the UI in the CUI output model,

containing text from captions and layout information. For each UI element the converter engine retrieves the caption, the type, the size and the position on screen (see Section 4.4.3). In the same way, the converter engine retrieves possible capabilities for interaction (interaction types) with each UI element from the CUI input model. This information is then provided to the perception process of the user interaction model with the help of the UI element information API.

By providing this information from the CUI models, the user interaction model perceives relevant information from the current state of the application's UI. Thus, an implementation of this approach has to provide a routine that defines how this information is perceived. Such a *visual attention* (VA) defines how the modeled user is searching for relevant GUI elements with regard to their caption, size, position and further information.

A VA routine is usually part of cognitive architectures. The described user interaction model offers basic VA, capable of transferring all or a subset of the perceived UI elements to the subsequent processing step. Depending on the implemented VA routine this transfer can be in a specific order and amount. However, decoding of GUI elements such as icons that do not provide a caption is not supported by the perception of the described user interaction model, but could be added using specific algorithms from computer vision or image recognition.[20] Instead, information from the application models can be used to determine correct interaction with such elements (see previous Section 5.3.2). In the literature several implementations of the VA for representing the strategy for the perception process are described, while the most common approaches can be summed up as following:

- *Bottom-up* perception; i.e. influenced by what is perceived on the screen; or
- *Top-down* perception; i.e. influenced by cognitive processes from the processing module and the current goal of interaction.

Influencing factors for these approaches and further strategies are discussed e.g. by Halverson and Hornof (2007), Underwood (2009) and Betz et al. (2010).

#### Processing

In the subsequent processing step, interactions are chosen depending on whether they move the interaction process towards the current goal of the user interaction model. This

[20] E.g. OpenCV - http://opencv.org/ - Accessed in July 2015.

follows the assumption that human users interact rationally, as proposed in most *predictive simulation* approaches for user interaction; e.g. ACT-R and the PUM framework

Different strategies for matching perceived information from the application's UI with the simulated UTK, and thereby choosing the best fitting information, suit for this approach. Again, two ways of choosing an interaction are applicable for an implementation with the described user interaction model and depend on the way how the UTK gets provided:

1. Approaches based on a **predefined solution path** (e.g. Jameson et al. (2007)); aim at the simulation of ideal interaction behavior, which, if required, may also include specifically modeled interaction errors. This allows evaluating the application in a way how the designer envisions the flow of interaction or how a specific interaction sequence should look like. Such an approach does not consider any deviations and therefore only focuses on a narrow excerpt of the whole application. This, of course, minimizes the effort for modeling the application for evaluation needs, but it also limits the expected evaluation results. The approach requires the UTK to be specified as predefined interactions (see Section 5.3); i.e. as an ordered list of attribute-value pairs describing all required actions to be performed by the user interaction model.
2. Approaches using an integrated **on the fly interaction decision** mechanism; e.g. based on semantic similarities between the current simulated user task and the application's UI, usually suit best when evaluating navigational behavior. Thus, these approaches are mainly used when evaluating tasks where specific information needs to be searched. Rieman et al. (1996) propose the principle of *label following* to tackle this challenge; i.e. checking if captions of UI elements are semantically similar to a description of the (current) goal. This is achieved by providing a predefined goal description (see Section 5.3). For user tasks requiring longer interaction paths, the UTK can be structured by subgoals which the user interaction model needs to pass through. This can be modeled by using the hierarchical structure of the UTK. During the processing step, the semantic similarity between the current (sub-)goal and all provided captions from each application state is compared and the UI element with the highest similarity value is chosen for interaction.

In general, both strategies are applicable for implementing this approach by providing the required UTK and appropriate matching routines. Furthermore, having the option of choosing from both strategies allows for a high degree of freedom for the applied simulation-based AUE approach and consequently satisfies **(Req. Inf. 7 - Goal-State)**.

#### Interaction Execution

In the last of the three processes of the user interaction model, the execution of the selected interaction from the processing routine is simulated. In terms of interaction with GUIs this can be a simple click or touch on a button. As the possible interaction capabilities of each perceived UI element are known from the CUI input model, the user interaction model is enabled to perform an appropriate interaction via the UI element information API. Thus, the chosen interaction on the UI element is applied via the UI element information API and the converter engine as an input to the CUI input model.

The effect is that all application models can be updated accordingly and the next application state is generated by the executable application models so that a new cycle of interactions is started.

### 5.4.3. End of Simulation - AUE Analysis

After the simulation is finished; i.e. the goal of the user task is accomplished; the AUE simulation engine triggers the AUE analysis. A simulated interaction path is generated from all performed interactions together with information from the UI element information API and the user interaction model. Based on the specific AUE tool, relevant usability criteria are predicted on the simulated interaction path; e.g. task execution times.

## 5.5. Summary

In this chapter a conceptual environment for automated model-based usability evaluation of adaptive UIs based on simulated user interaction is derived. The focus is set on required information from Chapter 3 and derived required models, an information transfer between the models and processes to conduct the simulation.

More specifically, the approach allows for evaluating adaptive UIs conforming to CAMELEON by integrating the development UI models into an automated interaction simulation and usability evaluation. The approach also enables the evaluation of the application with regard to defined user tasks and a specific context of use. Furthermore, the approach allows the evaluation of simulated expert and non-expert behavior via different strategies that are conducted by the user interaction model.

In the following chapter a specific implementation of this conceptual approach is outlined.

# 6. Implementation

This chapter describes a specific implementation based on required information from Chapter 3, related work from Chapter 4 and the conceptual architecture from Chapter 5. The implementation presents a proof of concept that demonstrates different characteristics of the overall conceptual approach.

A first implementation decision for the proof of concept covers one of the basic claims of this thesis and the conceptual approach. An existing framework for model-based UIs is used to provide information about the application factors. The MASP (see Section 4.2.3) provides the application models and an execution runtime for these models for evaluation in this implementation. Required information is accessed by the implemented converter engine via an API. Section 6.1 explains how information from the application models is used by the implemented AUE simulation engine to simulate interaction. A specific focus of the description is set on how the implementation aims at preserving compatibility with the MeMo approach (see Section 4.1.4) and thus enables reuse for further evaluation purposes using the MeMo workbench as a specific AUE tool.

In Section 6.2 the implemented user task model is described. Subsequently, the implemented user interaction model, more specifically the implemented version of the semantic processing (Section 6.3), is explained in detail. Both implementations make explicit how information from the MASP UI development models is used and provide additional benefits to the implemented architecture.

Section 6.4 highlights the implementation of the AUE simulation engine UI, which allows maintaining simulations and exporting results for evaluation.

Finally, a further implementation decision is targeted in Section 6.5. The section describes the interconnection with an external AUE tool. For the purpose of predicting task execution times, the ACT-Simple framework (Salvucci and Lee, 2003) as part of CogTool (John et al., 2004) is applied with the benefit, that none of the required models for CogTool need to be created manually. Instead, they are provided in an automated way and thus save modeling effort and time for this specific evaluation based on predictive modeling.

## 6.1. Implementation of the Converter Engine

The converter engine of the approach is implemented for representing UI surface information and interaction logic of MASP UI models to the user interaction model and the user task model for interaction purposes. This section describes how these concepts are applied in the implementation.

The provided implementation of the MASP runtime framework does not feature a fully developed CUI model as described in the conceptual MASP framework and required for implementation of the approach of this thesis. Instead the layout is computed based on AUI model information (Feuerstack (2008, pp. 87-95)). Thus, all required information for the UI element information API cannot be directly accessed from the CUI models; e.g. the current position and size of the (final) UI elements are computed dynamically based on the current platform. Consequently, a solution was implemented that provides current information from the FUI and maps it to the AUI model. This way, all required information can be accessed. The implementation is comparable to the web browser interface of the UI management layer described by Biswas et al. (2013), but additionally adds mappings to the AUI model as required by the approach of this thesis.

### Mapping of FUI Elements to AUI Elements

In a first step, all UI elements from a given UI screen of a MASP application and their relevant attributes for an evaluation are identified. Conforming to the integrated approach, the UI element information API is used to provide information from each UI element on the UI screen to the simulation-based interaction process covered by the user interaction model. During this process, the task models that are being executed in the MASP, are used to filter relevant information. In particular, a subset of the ETS is queried from the MASP by filtering only tasks that are currently presented on the UI screen - the *presentation task set* (PTS). Each task of the PTS links to an AUI element that has a FUI representation on the current UI screen. Consequently, all required FUI elements are then collected by following the mappings from AUI elements to FUI representations in the MASP runtime platform.

For the proof of concept of this thesis a mechanism is applied that maps identifiers of HTML elements to expressions of AUI elements. These mappings between FUI representation and AUI model are implemented using descriptive paths to the location of the specific HTML element (representing the FUI) in predefined templates. Figure 6.1

depicts a part of the template from which the FUI is generated while Figure 6.2 shows an excerpt from the corresponding AUI model.

```
<td width="10%" align="center" valign="top" id="TVInfoPicture">
   <img id="#info_zacherl" src="../images/vcook/infobutton.png"/>
</td>
<td height="60" align="center" valign="center" id="startWithPersonalRecommendationButton">
   <div class="blackButton" align="center" id="#${selectedRecipe.name}" style="height:80%;width:98%">
      <table width="100%" height="100%">
         <tr align="center">
            <td class="ButtonNormal">Kochassistent starten</td>
         </tr>
      </table>
   </div>
</td>
```

**Figure 6.1.:** Identifiers in the HTML templates (in red frames) correspond to expressions in the XML description of the AUI model that is available in the runtime framework.

```
<elements
   xsi:type="OutputOnly"
   id="PersonalRecommendationReceiptInfo"
   label=""
   model="/">
  <templates
     templatePath="org/sercho/masp/demos/vcook/OptForTVRecipe.vm"
     expression="//td[@id='TVInfoPicture']"/>
</elements>
<elements
   xsi:type="Command"
   id="startWithPersonalRecommendation"
   label="pers&#xf6;nliche Empfehlung kochen"
   model="/">
  <templates
     templatePath="org/sercho/masp/demos/vcook/OptForTVRecipe.vm"
     expression="//td[@id='startWithPersonalRecommendationButton']"/>
</elements>
```

**Figure 6.2.:** Expressions (in red frames) in the XML document structure of the AUI model are added as mappings to the identifiers of specific elements in the templates used for generating the HTML (FUI representation).

By following these mappings, the required interconnection from AUI models, and thus task models, to FUI elements is accessed in the runtime framework. For this purpose, *Ajax Servlets* are implemented to retrieve required information such as width, height, position and captions of the HTML elements that are mapped to the AUI elements and provided via the UI element information API to the user interaction model.

Hence, this implementation provides up to date information of the UI screen (i.e. FUI), as it is at any given point of time on any available interaction device and for any adaptation variant of the UI provided by the MASP runtime framework.

#### Automatic on the fly Creation of a MeMo System Interaction Model

As stated above, a further implementation decision is to provide compatibility with the MeMo approach and its models in order to evaluate the simulated interaction path with this tool as well. For maintaining this compatibility and for providing an interaction trace of the simulation to CogTool, a part of the SIM that reflects all UI screens on the simulated interaction path, is automatically created during simulation. UI objects and UI states for the SIM are created from the UI element information API and transitions to subsequent UI states are created from the interaction that was selected by the user interaction model. Consequently, the SIM and the STM do not need to be modeled beforehand and are bypassed in the modeling phase of MeMo. Still, the reporting mechanism of the MeMo workbench can obtain the logging data from the simulated interaction path via its own models and provide reports.

In the following section the implemented user task model is described.

## 6.2. Implementation of the User Task Model

Using information from the MASP models, the user tasks can be implemented more freely compared with step-by-step predefined interactions and thus satisfy **(Req. Inf. 7 - Goal-State)**. More specifically, three additional features are implemented:

1. The **definition of a (MASP) PTS that describes the goal state** that needs to be reached during simulation.
2. An option for enabling **a routine of the user interaction model** that **processes** the UI state for **already filled in or wrongly submitted values** of e.g. text fields, radio buttons or checkboxes. In case such UI elements are perceived and their values do not match the provided UTK (including the case, that there is no such predefined information) specific task information is generated on the fly and added to the current UTK. Accordingly, the user interaction model is able to restore the UI and set values as required by the user task.
3. An option for defining if the order of specific information is irrelevant and thus allows for **non-deterministic interactions**. If enabled, this option gives the user interaction model the freedom to randomly select the order of interactions that correspond to the provided UTK if they can be performed in the current UI state. Such a behavior is useful to automatically simulate varying interaction traces; e.g. when checkboxes need to be selected without a specific order.

The approach is implemented using the simulation configuration that loads task information from a specific task file in an XML schema (Figure 6.3). All nodes, their values and usage during simulation are summarized in Table 6.1 on page 94.

```
<simulation name="Simulation_2_sameGoal" goalpts=
        "ConfirmCall;DecreasePersonsNumber;DisplayPersonsTip;IncreasePersonsNumber;
        InputPersonsNumber;RequestIngredients;Restart;SpecifyPersons"
        autocomplete="false">
    <usertask name="SelectChoice" deterministicorder="false">
        <knowledge>
            <information varname="Hauptgericht" description="no description"
                list="false">Hauptgericht</information>
            <assignment information="Hauptgericht">Hauptgericht</assignment>
        </knowledge>
        <knowledge>
            <information varname="Deutsch" description="no description"
                list="false">Deutsch</information>
            <assignment information="Deutsch">Deutsch</assignment>
        </knowledge>
    </usertask>
    <usertask name="SelectRecipe" deterministicorder="false">
        <knowledge>
            <information varname="AltdeutscherSauerbraten" description="no description"
                list="false">Altdeutscher Sauerbraten</information>
            <assignment information="AltdeutscherSauerbraten">Altdeutscher Sauerbraten</assignment>
        </knowledge>
    </usertask>
    <usertask name="IncreasePersonCount" deterministicorder="false">
        <knowledge executecount="2">
            <information varname="Plus" description="no description"
                list="false">+</information>
            <assignment information="Plus">+</assignment>
        </knowledge>
    </usertask>
</simulation>
```

**Figure 6.3.:** Example of a task file providing required user task knowledge encoded as information and assignments and a specific goal state encoded as a PTS.

The implemented schema allows directly providing a start state (simulation startpts) and a goal state (simulation goalpts), as well as specific interaction steps required to reach the goal (usertask) that can be grouped in subtasks. Consequently, the task file is used for specifying UTK that represents necessary manipulating interaction steps but can also provide UTK for additional navigational interaction steps. For example, this could be the captions of UI elements that have to be used to select and enter required values. In this case, interactions that are identified based on these captions are always preferred if they can be applied by the user interaction model in the current UI state. Finally, the designer can also determine in which UI state the user interaction model has to provide specific information from a subtask (usertask executepts). This is useful in case different UI screens allow entering identical information but in a different context. Thus, if required, the subtask can be annotated with the string of the PTS in which it needs to be executed.

**Table 6.1.:** XML nodes of the task file, their values and usage during simulation (compare with a real example depicted in Figure 6.3 on page 93).

| XML Nodes of the Task File | Values | Usage during Simulation |
|---|---|---|
| **Simulation name** | String representing the name of the simulation | Identifies the simulation. |
| **Simulation startpts** | Concatenated string of a MASP PTS | Identifies the PTS in which the simulation starts. |
| **Simulation goalpts** | Concatenated string of a MASP PTS | Identifies the PTS in which the simulation ends. |
| **Simulation autocomplete** | Boolean value | Sets a feature of the user interaction model to automatically perform interactions that undo erroneous input (see Section 6.4). |
| **Simulation loop** | (Optional) integer | Specifies the amount of iterations. |
| **Usertask name** | String representing the name of the user task | Identifies the specific user task. |
| **Usertask executepts** | (Optional) concatenated string of a MASP PTS | Identifies the PTS in which to transfer the associated UTK. |
| **Usertask deterministicorder** | Boolean value | Specifies whether the provided UTK (information assignments) is processed in sequence or random order. |
| **Knowledge executecount** | (Optional) integer | Represents the amount of repetitions for transferring the associated information assignment from the UTK. |
| **Knowledge information** | String representing the attribute of an information assignment | Identifies information assignments (see Section 5.3). |
| **Knowledge assignment** | String representing the value of an information assignment | Specifies a concrete value of the information assignment (see Section 5.3). |

## 6.3. Implementation of the Semantic Processing by Exploiting AUI and Task Models

The description in this section focuses on the implemented processing module. The preceding step of perceiving information with a VA routine from the UI screen is implemented in a basic bottom-up version. Thus, all UI elements from the current UI screen are forwarded to the processing step. In the same way, the implemented version of the interaction execution module is kept simple and performs an interaction via the provided method of the UI element information API.

In Section 5.3.2 two ways for providing UTK are explained, whether by a predefined goal description or by predefined interactions. Both approaches are applicable for implementation when using the MASP UI development models. In Section 6.5 an AUE tool is described that is applied to the implementation and that relies on predefined interactions. While that is trivial using optimally predefined UTK, this section describes a specific implementation of the *semantic processing module* of the user interaction model that requires a predefined goal description. During simulation the module automatically looks up further required information from the MASP development models; i.e. only manipulating interaction steps are provided by the designer.

### 6.3.1. Usage of Semantic and String Similarity Metrics

Steinnökel et al. (2011) describe an implementation for determining semantic relatedness of UTK with perceived information from the UI. In case the provided UTK does not exactly match any UI element information, synonyms are queried and semantically similar words from online sources are retrieved (OpenThesaurus[21], Wortschatz[22] and GermaNet[23]). Using the DISCO algorithm (Kolb, 2008) semantic relatedness is calculated with the result from these queries and the perceived information from the UI. In this proof of concept a similar mechanism is applied and enhanced.

An area of application for the semantic processing module is when the evaluator wants to simulate non-expert behavior; i.e. evaluating tasks for navigation and information search. Thus, the module does not guarantee finding the correct interactions depending on the provided UTK. Instead it depends on the outcome of the calculation of the semantic relatedness. Steinnökel et al. (2011) also present an evaluation of expected

[21] http://www.openthesaurus.de/ - Accessed in November 2014.
[22] http://wortschatz.uni-leipzig.de/ - Accessed in November 2014.
[23] http://www.sfs.uni-tuebingen.de/GermaNet/ - Accessed in November 2014.

results and compare the approach with existing tools that follow a similar approach; e.g. CogTool-Explorer (Teo and John, 2011).

When having the UI development models available, these models can be exploited to obtain more information and to enhance the semantic processing. The generic implementation can be separated into two different strategies that each perform a different analysis of the task and AUI models of the MASP to better support the decision for an interaction of the user interaction model:

1. An **analysis of the AUI model structure** to improve the selection of UI elements based on their (logical) grouping in the AUI model (Section 6.3.2).
2. An **analysis of the task model** to reason about the order of multiple interactions based on the modeled interaction logic (Section 6.3.3).

Figure 6.4 depicts the basic work flow of the implemented semantic processing module and the placement of the two implemented analysis methods of the task model and the AUI model that are described below. Internally, the module is working with information from the SIM (of MeMo) that is held in parallel. Consequently, the notations of the MeMo approach are used for the descriptions below.

### 6.3.2. Implementation of an Analysis of the AUI Model Structure

An analysis of the AUI model is performed in case there are multiple UI objects that can be matched by the semantic processing module with describing words from the same information assignment. In this case it is not conclusive which interaction needs to be performed. However, when the information assignment consists of multiple describing words, these can be used to disambiguate the selection between the matched UI objects (see Figure 6.4).

The underlying assumption of this implementation is that the logical structure of the AUI model is a valuable source of information. This logical structure is used to retrieve improved simulation results for specific simulation goals (see Section 4.4.3). In general the AUI model does not specify the layout of a GUI. But a semantic relation between AUI elements can be anticipated, if the designer chose to group several AUI elements using complexInteractors (see Table 4.2). For example when descriptive AUI elements are grouped with AUI elements that are used to provide interaction capabilities to describe their meaning; e.g. a label is used to ask for specific input from a group of radio buttons or checkboxes (see Figure A.2 on page 144 where the type of menu can be selected and describing text is presented using an associated label).

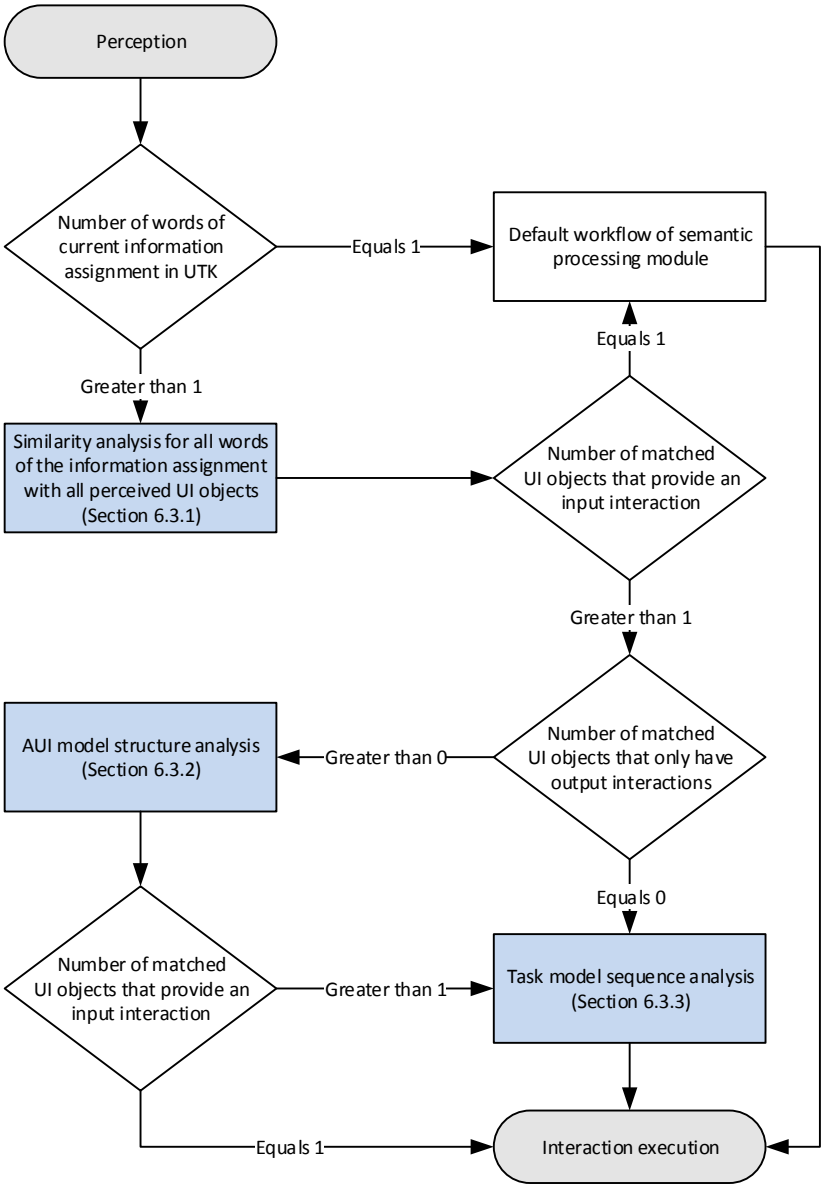


**Figure 6.4.:** Sequence of work flow within the enhanced semantic processing module.

This approach is helpful for the evaluation under the following preconditions:

- The UTK might not be defined using exact captions of UI objects that have to be interacted with, so that the user interaction model needs to match provided UTK with perceived information from UI objects; i.e. the UTK does not represent predefined interactions. Instead information assignments may even consist of several words that can be matched with different UI objects (compare precondition *number of matched UI objects that provide an input interaction* in Figure 6.4).
- A single information assignment of the UTK specifies interaction with a UI object based on semantics that are bound to surrounding information from the UI; e.g. a describing label or header that is not part of the UI object itself. This might be the case when two or more UI objects that have identical (or very similar) captions but belong to different categories that are distinguished by surrounding UI objects (compare additional precondition *number of matched UI objects that only have output interactions* in Figure 6.4).

In this implementation of the approach, the hierarchical structure of the AUI model is accessed via the converter engine using mappings (Figure 6.5) in case specific requirements are met in the described order:

1. The describing words of the same information assignment were matched to specific UI objects (upper part of Figure 6.5).
2. In case UI objects that have input interactions (i.e. are mapped to AUI elements of type command, input or choice) and further UI objects that only have output interactions (i.e. are mapped to AUI elements of type outputOnly) are matched by the processing module, it checks for finding a relation between these elements in the AUI model. For this purpose, the module establishes the connection to the MASP AUI model via the UI element information API of the converter engine to the FUI elements (middle part of Figure 6.5).
3. If the mapped AUI elements (outputOnly and input/choice/command) are on the same hierarchical level and are grouped by the same complexInteractor then the appropriate (input) interaction on this UI object is preferred over competing ones and, ultimately, the associated task is triggered via the mapping to the AUI element (lower part of Figure 6.5).

To sum up, the implemented analysis of the AUI model allows for the definition of the UTK to be enhanced and specified more freely. Information assignments can be stated that consist of multiple words and thus can be specified in more detail so that ambiguous

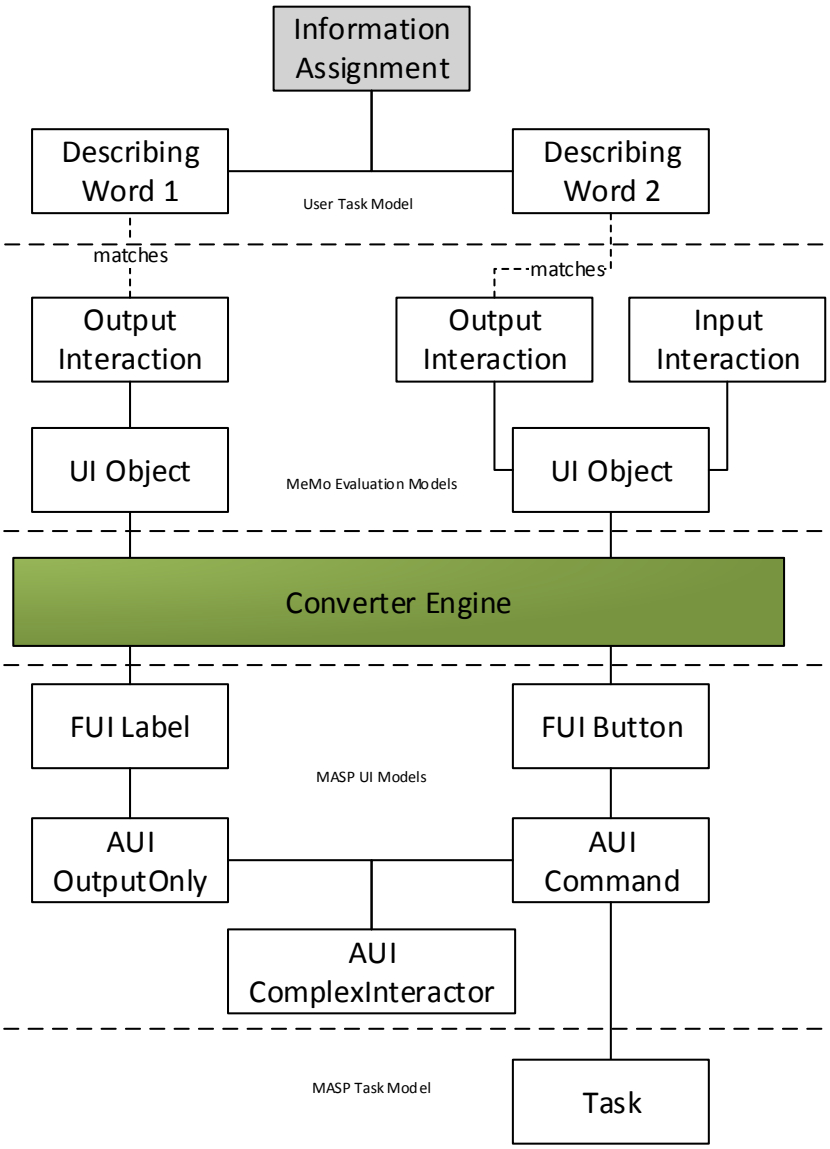


**Figure 6.5.:** Exemplary analysis of AUI model structure when two words of an information assignment can be matched with different UI objects that are grouped on the level of the AUI model.

selections can be distinguished. An example gives Figure A.1 on page 143 where the word "starten" can be mapped to two buttons of the same screen that are further specified by surrounding UI elements. As a side effect, this makes the definition of the UTK more comparable to task descriptions of real user tests by using descriptions that do not only use exact keywords from the UI (like a step-by-step guide). Instead it allows for indirect matches with the UI elements using paraphrasing words.

### 6.3.3. Implementation of an Analysis of Task Model Sequences

As depicted in Figure 6.4 an analysis of the task model of the application is required when there is more than one interaction derived from the current information assignment. In this case the order of interactions can be determined based on the task model of the application and is made accessible to the user interaction model. For this purpose the same mappings are established as in the analysis of the AUI model structure. The difference is that two UI objects that can be interacted with are matched with two different words from the information assignment; i.e. both UI objects have input and output interactions. Furthermore, not the relation on the level of the AUI is analyzed but the relation between the associated interaction tasks. Consequently, the CTT operators between the associated interaction tasks are checked for a specific dependency and order (Figure 6.6). A clear statement can be given if one of these tasks *enables* the other. The same applies to the relation *enabling with information passing*. In both cases the order of interactions is passed to the user interaction model, which then performs the interactions in the provided order.

An example for such a case is presented in Figure 7.1 on page 109 where the user needs to select the recipe from the list on the left side at first and then needs to select one of the buttons on the right side for declaring whether the specific ingredient is available or not. Both UI element types, the list and the buttons, provide input and output interactions. If the same information assignment indicates whether a specific ingredient is available or not, the task model analysis can provide the user interaction model with the required order of interactions that need to be performed for succeeding.

Both, the AUI model analysis and the task model analysis are part of the evaluation described in Chapter 7. In the subsequent section an implemented component is described that assists the designer during simulation and evaluation.

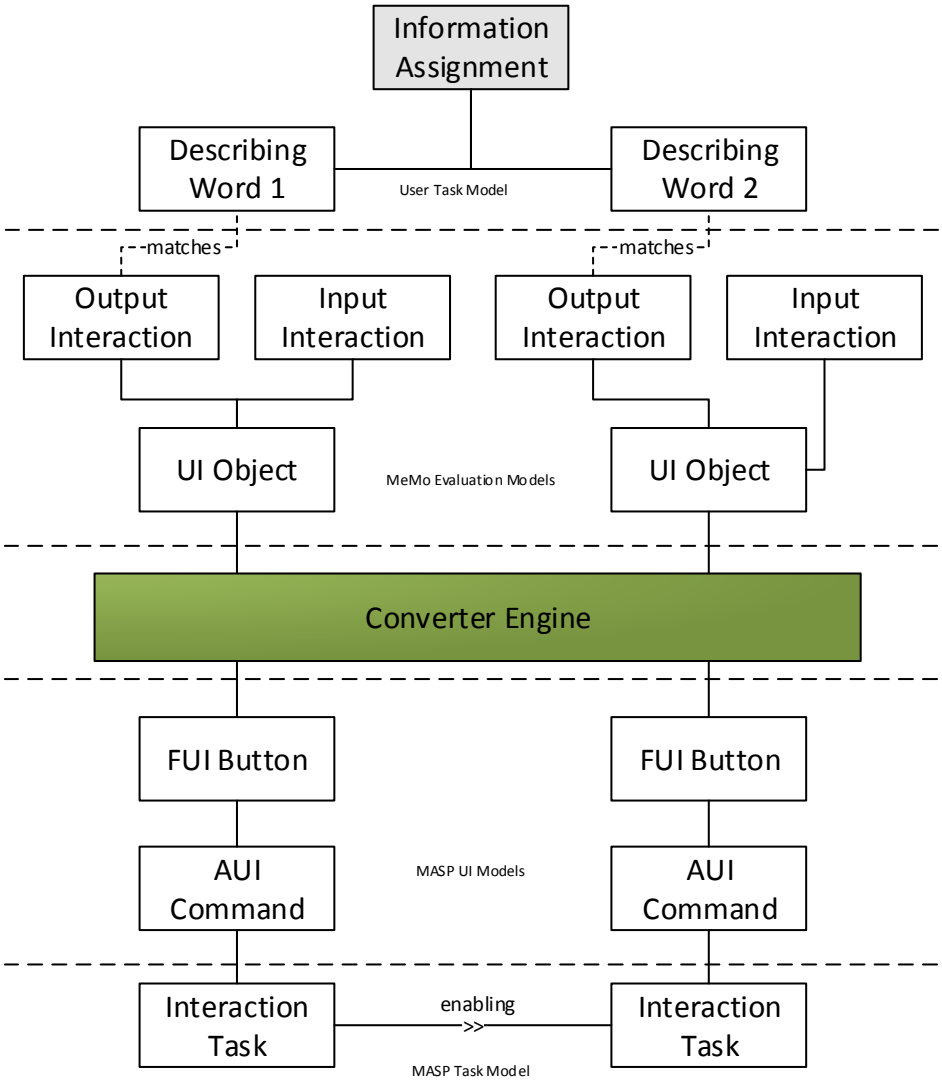


**Figure 6.6.:** Exemplary analysis of task model sequences in order to deduce the order of interactions for the user interaction model.

## 6.4. The AUE Simulation Engine UI

A further component for controlling the simulation process with an application of the MASP runtime framework and the user interaction model was implemented - the *AUE simulation engine UI* (Figure 6.7).

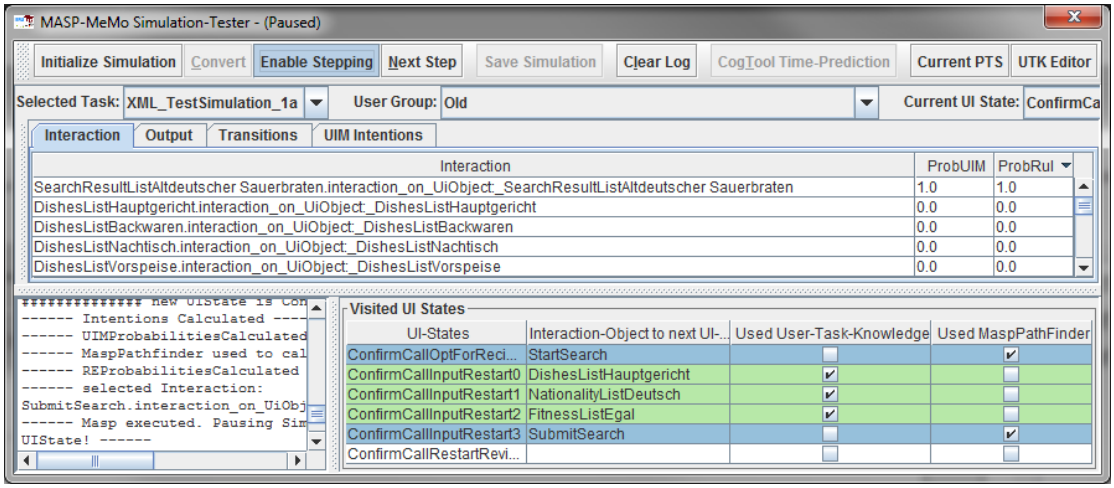


**Figure 6.7.:** The AUE simulation engine UI is a GUI for defining the UTK, monitoring the simulated interaction process, adjusting parameters to specific needs and exporting results of the simulation.

The AUE simulation engine UI is initialized during boot up of the MASP runtime framework and provides a GUI which allows defining and specifying parameters relevant for the simulation. As depicted in Figure 6.7 these parameters include:

- **Initializing the simulation** by choosing a specific task file which includes UTK and a goal state for the user task defined by a specific PTS of the MASP (see Table 6.1 for a full description).
- **Querying** the **current PTS** of the MASP required for specifying the start state or goal state of a specific user task.
- **Displaying** the **current user task** and the selected **user group**.
- **Editing** the **UTK** during simulation by adding or removing information assignments.
- **Stepping through the simulation** using breaks at all important points.
- **Displaying** all possible **interactions that the user interaction model can perform** in the current UI state (and their probabilities for interaction).

- **Modifying probabilities for interactions** and thus specifying a flow of interaction, if required.
- **Providing log data** for past interactions.
- **Displaying whether** information from the **UTK was used** or **navigational interaction steps** were derived from the MASP models.
- **Exporting** the simulated interaction path **to the MeMo workbench** for further evaluation using the internal mechanisms and tools.
- **Exporting** the simulated interaction path **to CogTool** and triggering the prediction of task execution times (see Section 6.5).

To sum up, with the help of the AUE simulation engine UI, the developer controls the simulated interaction process and is also able to adjust the simulation to required needs and stepping through the simulation; i.e. converting each UI screen of the MASP runtime framework and observing the processes of the user interaction model for perception, processing and interaction execution.

The next section describes an implementation for automatically predicting task execution times based on the combined architecture.

## 6.5. Applying CogTool for Predicting Task Execution Times

CogTool is applied as a proof of concept for an existing AUE tool for the implementation of the approach. The main reason for this decision is that CogTool is widely used and the underlying cognitive architecture (ACT-R) has been validated and applied in many domains. As a constraint of this approach, predicted task execution times are valid for expert interactions only. Consequently, a demonstrated interaction path cannot contain interaction errors or represent behavior of non-experts. Consequently, interactions of *adequately trained* users need to be simulated for e.g. routine tasks.

### 6.5.1. Requirements for Applying CogTool

In order to apply an evaluation with CogTool, representations of all *visited* UI elements from each UI screen that is on a specific interaction path are required. Using the original approach of CogTool, this is achieved by manual demonstration of interaction by the evaluator on the modeled UI. Accordingly, CogTool relies on the combination of

interaction logic (provided by the evaluator) with UI surface information that is also annotated manually (see Section 4.1.3).

For applying this in an automated way, the integrated approach of this thesis provides a simulated interaction path in combination with derived UI surface information (Section 6.5.2). This simulated interaction path then replaces a demonstration by hand while the derived UI surface information is transformed to models that are further processed by CogTool (Section 6.5.3). Finally, these interaction traces are then compiled to ACT-R using the integrated compiler of CogTool and then can be analyzed further by the evaluator.

### 6.5.2. Providing a Simulated Interaction Path to CogTool

Using the approach, a simulation for providing the interaction path to CogTool needs to be free of interaction errors. CogTool, i.e. the ACT-R model that is transformed from the KLM sequence, would also predict execution times for simulated interaction paths including errors. Yet, these predictions would e.g. not consider the (mental) processes and thus time required for noticing the errors. As a result these predicted times would not be as accurate. Thus, this special case is excluded from further descriptions. Instead, the user interaction model is provided with UTK that consists of a list of captions of UI elements that need to be interacted with in order to reach a specified goal state. Such predefined UTK matches the approach of CogTool to evaluate expert interaction tasks that are predefined by the developer.

In case the current UTK matches the caption of a UI element information, an interaction on that specific element is simulated. Finally, using mappings of the combined architecture as depicted in Figure 5.1, the corresponding AUI element in the MASP runtime framework is executed, which activates the next ETS of the application. By this means, the subsequent UI screen is triggered, which is evaluated accordingly until the end; i.e. the whole UTK is matched and the predefined goal state has been reached.

### 6.5.3. Transformation to CogTool

At this point, the simulated interaction path is complete and a partial MeMo SIM is built in parallel; i.e. the created SIM only reflects the simulated interaction path but not UI states that were not reached during simulation and therefore are not required for the ongoing evaluation. The partial SIM reflecting the simulated interaction path is

transformed to specific CogTool models - *widgets* are bound to *frames* that are interlinked by *transitions* that reflect *actions*.[24]

As can be seen in Figure 6.8, the basic concepts and elements of the UI models of MeMo and CogTool are similar, which eases the transformation of the models. Information for widgets is transformed from UI objects and UI states are transformed to frames. The simulated interactions of the user interaction model correspond to (manual) actions in CogTool. Additionally, information about the simulated interaction device for the CogTool project stems from the context model of the MASP and is then bound to the executed (input) interaction of the MeMo SIM. By this means, required information for a demonstration path for CogTool is created and evaluated automatically to predict the overall task execution time.

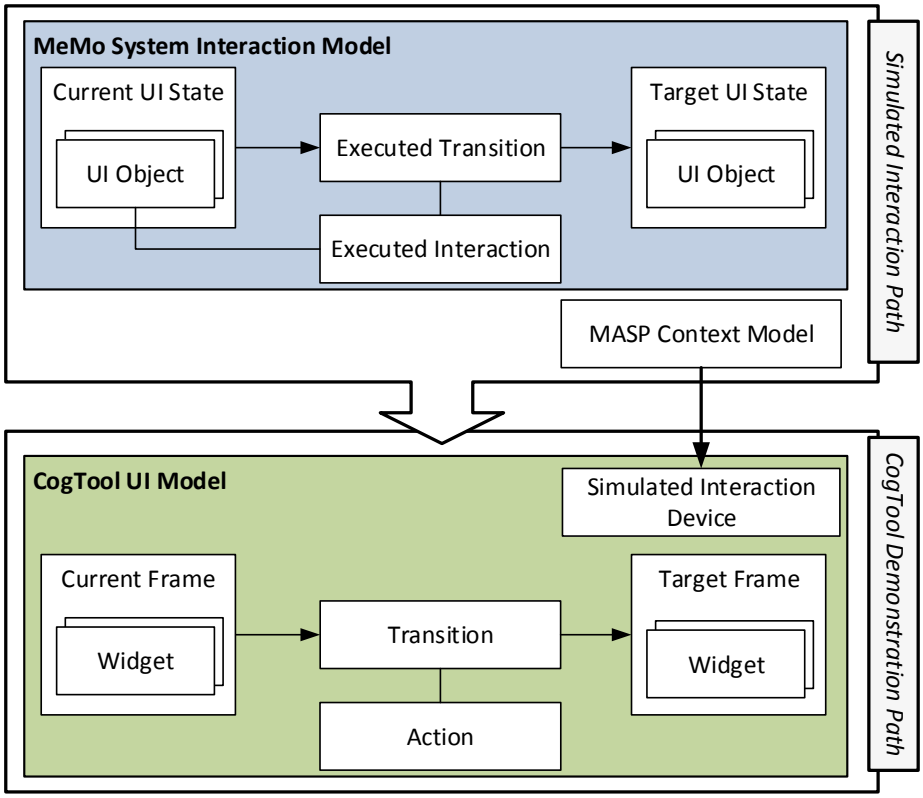


**Figure 6.8.:** Transformation of the simulated interaction path to a demonstration path in CogTool's models for predicting task execution times.

[24] This two-step approach was chosen to maintain compatibility of the implemented module with further projects created using the MeMo workbench, which in this way can be converted to CogTool as well.

The whole interaction process can be further validated using internal analysis methods and views of CogTool; e.g. checking for specific interaction steps and their associated prediction times or comparing different adaptation variants (see Figure B.3 on page 151).

## 6.6. Summary

This chapter highlights the specific implementation of the conceptual approach described in Chapter 5. The implemented architecture simulates interaction with application models of the MASP runtime platform, which was chosen in this specific proof of concept. In a first step, the implementation of the converter engine is highlighted. It accesses MASP application models during simulation. For this purpose a specific implementation is based on mappings between the AUI and FUI models of the MASP to gain required information for the simulation-based AUE approach.

Furthermore, the implementation of the user task model is targeted. This specific implementation explicitly enables additional functionalities based on access to application models of the MASP. This chapter also presents implemented routines of the user interaction model for an analysis of the MASP task model and an analysis of the grouping of elements in the AUI model. Both implemented routines make use of the converter engine and allow the user interaction model to determine the order of ambiguous interactions and to combine perceived information for better matches with the provided UTK.

With the help of the AUE simulation engine UI an implementation for maintaining the simulation-based interaction process is presented. The AUE simulation engine UI specifically provides adjusting simulation parameters and exporting functionalities.

Finally, with the help of CogTool, the cognitive architecture ACT-R is applied for predicting task execution times without requiring deeper insights into cognitive modeling. The described implementation reduces the modeling effort even more by using the development models of the MASP that are being executed in the runtime framework. Required information is transferred to CogTool together with the simulated interaction path according to the predefined goal state.

In Chapter 7 an evaluation of the implementation is described using a case study for highlighting further benefits from the combined approach.

# 7. Evaluation

This chapter presents evaluation results using the implemented architecture from Chapter 6. For the purpose of a running example a cooking assistant application is described in Section 7.1 that was developed using MASP development processes. The underlying models of the application and several adaptation variants are explained. Subsequently, case studies are presented that were conducted with the cooking assistant for validating the main concepts and the statement of this thesis proposed in Chapter 1.

Section 7.2 focuses on the savings in modeling time and decreased complexity of the modeling process when using the implementation described in Chapter 6. For this purpose, predictions are made for the expected time for modeling the application models using the MeMo workbench (Section 7.2.1). Furthermore, savings in the definition of required user task knowledge are outlined when using the described implementation of the processing module (Section 7.2.2).

Section 7.3 explains a case study and observations from user tests for showing how predictions of CogTool can be improved automatically when information from the development process is available. The work described in this section was conducted as part of the DFG project "Automatische Usability-Evaluierung modellbasierter Interaktionssysteme für Ambient Assisted Living".[25] At first, the experimental setup of an initial user test is described (Section 7.3.1) and then hypotheses for improvement using MASP application models are derived (Section 7.3.2). Finally, the improvements made to the prediction are validated within a second user test (Section 7.3.3).

This chapter concludes with a summary of the achievements and evaluation results in Section 7.4.

[25] The project is conducted together by DAI-Labor and the Quality&Usability Lab of TU Berlin. Preparation and execution of the user studies were conducted together. Analysis of the results from the user studies was conducted under the lead of Marc Halbrügge, who also provided the charts depicted in Figure 7.7, Figure 7.11 and Figure 7.12. Provision of the adaptation variants of the Cooking Assistant, the logging framework and the implementation of the improved CogTool version according to the three hypotheses described in Section 7.3 using information from the MASP application models were conducted and supervised by the author of this thesis.

## 7.1. The Cooking Assistant - A MASP Application

The interactive cooking assistant is an adaptive application that was developed using the model-based UI approach of the MASP. It provides the user with a cooking recipe database and assists in choosing and preparing meals during everyday life in a smart home environment. This section sums up the development goals and the underlying models for preparing the understanding of the case studies. A detailed description of the models and the development processes of an initial version of the cooking assistant is presented by Feuerstack (2008, pp. 148-180).

### Development: Goals and History

Initially developed during the SerCHo[26] project, the cooking assistant integrates an adaptive user interface and the capabilities of providing different interaction modalities. Subsequently, the cooking assistant was enhanced with additional models and an updated UI layout. It is capable of adapting its UI to different interaction devices that may have different sizes of the screen and different input devices; e.g. a touchscreen in the kitchen, a smartphone or a TV.

Furthermore, the cooking assistant is also able to adapt its UI to other parameters of the context of use; e.g. the observed distance of the user to the screen. Conforming to this observed distance, adaptations can be applied that enlarge relevant parts of the UI for a better perception and an additional voice output can be triggered. If required, the cooking assistant also allows controlling different kitchen appliances; e.g. the oven, the hood or a bread machine. This suits especially for users with disabilities; e.g. motor-impairments, as it allows the whole preparation process to take part with various interaction devices in an assistive way.

### Development: Task Model and UI Models

The cooking assistant consists of several interactive subtasks leading to the final goal of preparing a meal (see Figure A.1 - A.8 in the Appendix A starting on page 143). One of the interactive subtasks is the configuration of a shopping list that contains ingredients

[26] The Service Centric Home (SerCHo) project was a research project with the aim of developing services for smart homes that can be easily integrated into existing solutions and providing overarching interaction capabilities; e.g. support for multimodal and multi-device interactions. Further information can be found on: http://www.dai-labor.de/projekte/abgeschlossene_projekte/projekte/sercho/ - Accessed in December 2014.

needed for the recipe. The corresponding UI screen with English translations is shown in Figure 7.1.

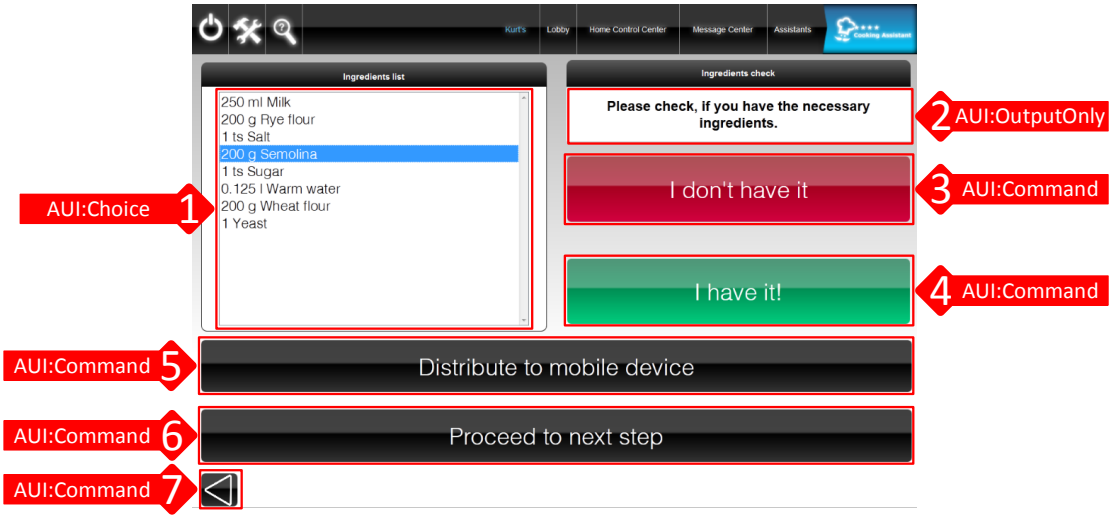


**Figure 7.1.:** UI screen for the recipe ingredients configuration task, annotated with AUI model information (the numbers correspond to those of the task model in Figure 7.2).

In the following, this interactive subtask of the cooking assistant is used as an excerpt for summarizing how the whole application is modeled using the MASP development approach and how information from these models is used for the AUE process in the case studies. The numbers of each AUI element correspond to the annotations of tasks from the task model of the cooking assistant that is shown in Figure 7.2 on page 110. By this means the connection from the task model to the AUI model is highlighted conforming to the overall architecture from Figure 5.1 on page 70.

The ingredients needed for the recipe (1) are listed on the top left of Figure 7.1 and the list is a representation of the interactive task *IngredientsList* (Figure 7.2). On the level of the AUI model the list is defined as a choice element, meaning that the user can select an entry from a set of alternatives. On the top right of Figure 7.1, the GUI provides a text hint (2), defined as a non-interactive element (outputOnly) in the AUI model that is displayed in parallel to the other tasks on the level of the task model. By tapping on the red (3) or green (4) button, the user informs the application about the availability of the currently selected ingredient in the household. Both buttons are modeled as command elements in the AUI model. Furthermore, interaction on these elements activates interaction tasks that enable associated application tasks in the task model. This association is depicted in the task model (bottom of Figure 7.2), where

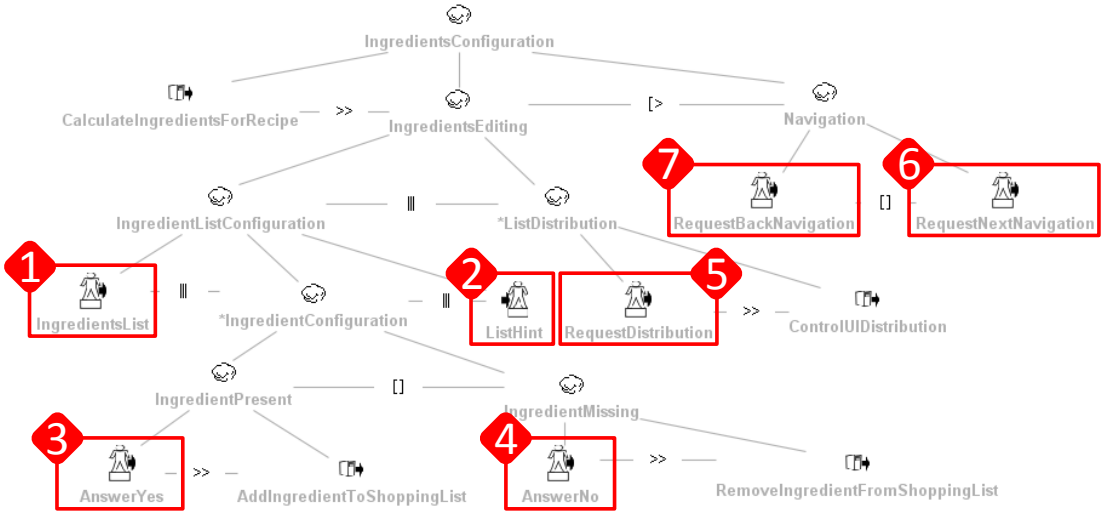


**Figure 7.2.:** Excerpt from the task model of the cooking assistant as modeled with the MTTE. Depicted is the subtree for the configuration of the recipe ingredients (the numbers correspond to annotated AUI model elements from Figure 7.1).

application tasks get enabled by both interaction tasks: *AnswerYes* enables *AddIngredientToShoppingList* and *AnswerNo* enables *RemoveIngredientFromShoppingList*. The same applies to the button for distributing the UI to a mobile device (5) with its corresponding interaction task *RequestDistribution* and the following *ControlUIDistribution* system task. The buttons (5), (6) and (7) navigate from the ingredients UI screen to further UI screens and are modeled as AUI elements of the type command.

The AUI model specification of these tasks is further refined in the CUI model as described in Section 4.4.3. By this means (1) is declared as a graphical list, (2) as a label, while tasks (3) - (7) are buttons. In addition to the output description, the CUI model also defines the mapping between the input provided by the user and the UI elements. The input definition for (3) - (7) is intuitive - any selection events (mouse clicks, touch taps) within the UI elements of these tasks trigger the execution of the underlying commands. In case of (1) the selection events are interpreted as the choice of the list element at which the input occurred. Finally, as (2) is a label, the CUI model does not define any input events.

### Adaptation Variants of the Cooking Assistant

Besides the standard version (*standard UI*), four different adaptation variants were focused for the evaluation described in this thesis for demonstrating the transferability of the implementation to specific adaptations that stem from the development process and were created by the designers (Figure 7.3). While the main goal of interaction remains the same for all variants - choosing and preparing a meal - all adaptation variants differ in specific criteria:

- *Adaptation 1* and *adaptation 2* have minor adaptations compared with the standard UI and are focused on reducing the amount of UI elements.
- *Adaptation 3* uses an adapted task model where two tasks are shown in parallel and the navigational step in between both tasks is removed to save an interaction step.
- *Adaptation 4* is presented on a tablet using a vertical layout instead of the horizontal mounted touch screen.

Thus, all variants ought to provide different interaction behavior that is reflected in criteria that can be measured using the integrated simulation approach.

Figure 7.3 depicts comparable UI screens from all adaptation variants to highlight the main differences, such as enlarged interaction elements in adaptation 2, parallel tasks for adaptation 3 and the vertical layout for the tablet in adaptation 4. The standard UI for this UI screen is presented for comparison in Figure A.5 on page 145.

## 7.2. Comparison of Model Complexity and Modeling Effort

This section illustrates in an exemplary way the complexity of the evaluation models of the application and user tasks and shows potential savings when using automation. For this purpose the standard UI and all different adaptation variants of the cooking assistant are simulated with three tasks, each leading to different goals of interaction. Required UTK for task 1 (choosing a specific recipe, checking available ingredients and preparing the meal) is presented in Table B.1 on page 152, for task 2 (choosing a specific recipe and checking available ingredients) in Table B.2 on page 153 and for task 3 (choosing a specific recipe) in Table B.3 on page 153.

While Section 7.2.2 focuses on the user tasks, a comparison of the models of the application and an analysis of the required time for manual modeling are presented below.

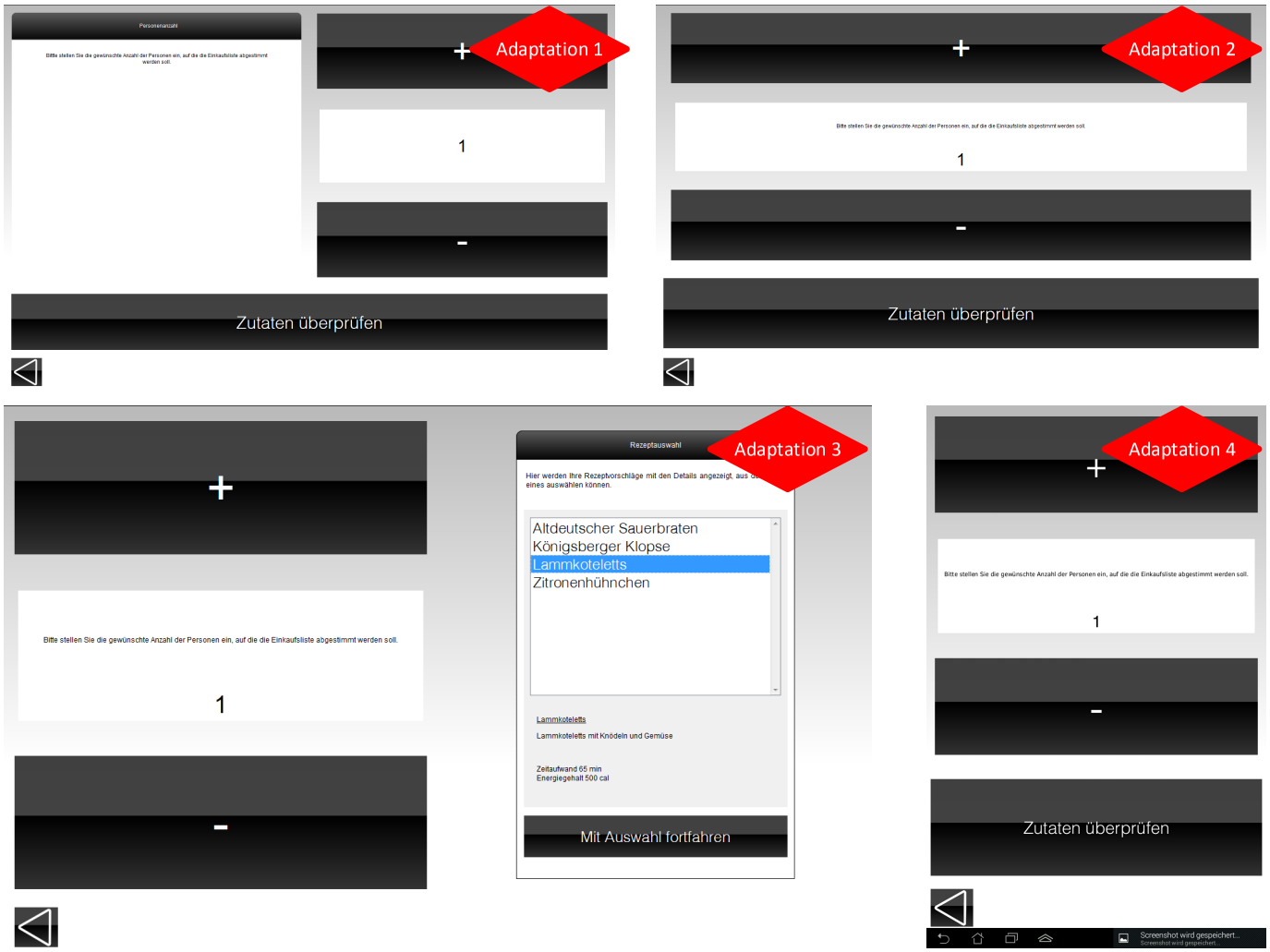


**Figure 7.3.:** Overview of four different adaptation variants of the cooking assistant.

### 7.2.1. Application Models - Case Study: Effort for Model Creation

Table 7.1 lists the numbers of UI states, interaction steps and UI objects when simulating the three different tasks with the different variants of the cooking assistant.

**Table 7.1.:** Numbers of converted UI states, simulated interaction steps and converted UI objects for three simulated tasks in five different UI variants.

| **Task 1** | **# of converted UI states** | **# of simulated interaction steps** | **# of converted UI objects** |
|---|---|---|---|
| Standard UI | 48 | 47 | 789 |
| Adaptation 1 | 48 | 47 | 717 |
| Adaptation 2 | 48 | 47 | 715 |
| Adaptation 3 | 47 | 46 | 725 |
| Adaptation 4 | 48 | 47 | 747 |
| **Task 2** | **# of converted UI states** | **# of simulated interaction steps** | **# of converted UI objects** |
| Standard UI | 22 | 21 | 311 |
| Adaptation 1 | 22 | 21 | 259 |
| Adaptation 2 | 22 | 21 | 256 |
| Adaptation 3 | 22 | 21 | 275 |
| Adaptation 4 | 22 | 21 | 251 |
| **Task 3** | **# of converted UI states** | **# of simulated interaction steps** | **# of converted UI objects** |
| Standard UI | 9 | 8 | 202 |
| Adaptation 1 | 9 | 8 | 162 |
| Adaptation 2 | 9 | 8 | 161 |
| Adaptation 3 | 9 | 8 | 173 |
| Adaptation 4 | 9 | 8 | 157 |

All of the 15 simulations were performed with identical UTK and thus give a proof of concept that the underlying approach of simulated interaction also adapts to different UI variants and tasks. While the total number of converted UI states and interaction steps remains mostly constant (apart from adaptation 3 in task 1), the number of converted UI objects does change significantly for all different adaptation variants (e.g. ~22% less UI objects in adaptation 4 than in the standard UI when simulating task 3).

#### Complexity of Application Models using MeMo

The MeMo workbench was chosen to demonstrate savings when using the automated approach in comparison to creating the models by hand with the MeMo workbench. On the one hand, this allows directly comparing the models because the implemented architecture is able to export the models conforming to MeMo and thus is able to serve

as a replacement of the modeling process in the considered scenario. On the other hand, the required modeling steps within the MeMo workbench for creating the SIM can be observed and the modeling effort and the required time can be recorded or predicted. For the sake of simplicity, in this case study the simulations of all three tasks with the standard UI are chosen, while the results can be applied to the other adaptation variants as well.

The MeMo workbench allows reusing already existing UI elements for different UI states. Thus only newly created or modified UI objects and transitions (further called *unique UI objects* and *unique transitions*) need to be counted in a best case scenario. By this means the modeling effort for similar UI states can be decreased. Furthermore, if interaction without errors is evaluated and the interaction steps are already known then only the appropriate transitions need to be modeled (*converted transitions* of the automated approach). Thus, Table 7.2 refines Table 7.1 by focusing on the standard UI only and presents the numbers of modeled UI elements for each of the three tasks.

**Table 7.2.:** Numbers of modeled elements for the standard UI and all three tasks specifying all UI states, all UI objects and all transitions as well as unique UI objects, unique transitions and converted transitions.

| | **Task 1** | **Task 2** | **Task 3** |
|---|---|---|---|
| **# of all UI states** | 48 | 22 | 9 |
| **# of all UI objects** | 789 | 311 | 202 |
| **# of unique UI objects** | 238 | 113 | 90 |
| **# of all transitions** | 486 | 205 | 113 |
| **# of unique transitions** | 486 | 205 | 96 |
| **# of converted transitions** | 48 | 22 | 7 |

### Time Prediction of Low-level Modeling Steps using MeMo

Tests with human users were discarded due to the relatively high modeling effort and the limited number of available skilled MeMo developers. Instead, predictions for the required modeling time of a SIM with the MeMo workbench were made with CogTool. This decision is backed by the benefit of better comparability using this objective approach. The assumption is that creating a SIM with MeMo consists of repetitive modeling steps for which the required time can be predicted with CogTool. Finally, the overall times can be calculated by adding up these times for the specific SIM and thus provide a founded estimate. For this purpose repetitive tasks such as the creation of a UI state, a UI object and a transition are targeted. The average number of letters for the captions of the UI objects was determined for each of the three tasks and a random

label using this number of letters was computed. This provides more realistic predictions. Furthermore, the time required for creating a transition between two UI states was predicted for the case that there are no conditions and consequences attached and for the case that a transition has one condition and one consequence. The latter case refers to a more common approach when modeling the SIM (see Figure 4.5). Figures B.1 and B.2 on page 150 depict examples of the CogTool models from the modeling process with the MeMo workbench. Table 7.3 lists execution times (and used random labels) for these low-level modeling steps that were predicted with CogTool.

**Table 7.3.:** Time predictions of CogTool for repetitive low-level modeling steps using the MeMo workbench.

| **Performed Task in MeMo Workbench** | **Used Random Label** | **Predicted Time in s** |
|---|---|---|
| Create a new UI state with background image | n/a | 12.4 |
| Create a UI object with 23 letters (avg. for all UI objects in task 1) | 3KS6lllt0RÖMvsqVß4s7pZ! | 23.9 |
| Create a UI object with 19 letters (avg. for unique UI objects in task 1) | OQwveMxdÜlZcdcRxXEO | 22.9 |
| Create a UI object with 22 letters (avg. for all UI objects in task 2) | I?rlYÜ3jJ48,0FOfl1fPXH | 23.3 |
| Create a UI object with 22 letters (avg. for unique UI objects in task 2) | ßZG+VQÜsÖeHeCü4üqDUzIT | 27.9 |
| Create a UI object with 21 letters (avg. for all UI objects in task 3) | VßüevÄZI8vidlzZY.aCwe | 24.3 |
| Create a UI object with 22 letters (avg. for unique UI objects in task 3) | 4YfüjE0JuDq9XD8eCtWy-, | 23.6 |
| Create a transition | n/a | 05.8 |
| Create a transition with conditions and consequences | n/a | 38.9 |

### Overall Estimated Time Predictions for Modeling the Application with MeMo

Estimations for the overall execution times for modeling the application with MeMo were computed based on the predictions for the low-level modeling steps in Table 7.3 and the number of UI elements from Table 7.2. These estimations are presented in Table 7.4 for several potential modeling scenarios that reflect the SIM as a combination of different types of UI elements:[27]

[27] The case where only those UI objects are modeled that are actively used during simulation; i.e. one UI object per UI state; was not considered as this is not a likely use case when using the MeMo workbench. Time predictions for creating such a SIM would be lower than the listed times and can be

- **All UI states, all UI objects and all transitions** relates to the maximum of UI elements that have to be modeled for the specific task. In this case no elements are reused and UI objects and transitions are modeled that are presented on the UI states of an interaction path, but that are not used during the simulation of a specific task in case no interaction errors occur.
- **All UI states, unique UI objects and unique transitions** relates to an average modeling process where already existing UI objects and transitions are reused (for the sake of simplicity, no further modeling time is added for the process of reusing elements).
- **All UI states, unique UI objects and converted transitions** relates to a simplified SIM, where existing UI objects are reused and only those transitions are modeled that are relevant for an expert simulation of the task for reaching the next UI state.

Finally, predictions for each of these three scenarios are made under the assumption that whether the transitions are modeled:

- **Without conditions and consequences** relates to the case that the modeled interaction logic is static and interaction with the same UI object always leads to the same subsequent UI state.
- **With conditions and consequences** relates to the case that the modeled interaction logic of the application is dynamic and thus depends on modified variables in the system information pool. Modeling of these conditions and consequences requires much more time than creating the transition itself (see Table 7.3).

The modeling of conditions and consequences in the SIM aims for restricting the interaction logic and for simulating effects of interactions to the modeled UI. For example, in case of the cooking assistant, consequences could be used for modifying available ingredients or the number of persons (see Figure 7.3). By this means an interaction with the associated button triggers a consequence that modifies an internal variable in the system information pool.

According to these distinctions, 18 different predictions of the modeling time were calculated that are presented in Table 7.4. A maximum and a minimum are derived out of the six predictions for each task, which are presented below.

calculated by using the same number of UI objects as converted transitions. However any side effects from other presented UI objects on the simulation process would be ignored in such simulations.

**Table 7.4.:** Time predictions of CogTool when modeling the part of the SIM of the standard UI required for simulating task 1, 2 and 3.

| | **Predicted Time in hh:mm:ss for Modeling** | |
|---|---|---|
| **Task 1** | **Without conditions and consequences** | **With conditions and consequences** |
| All UI states, all UI objects and all transitions | 06:11:11 | 10:39:18 |
| All UI states, unique UI objects and unique transitions | 02:27:44 | 06:55:51 |
| All UI states, unique UI objects and converted transitions | 01:45:24 | 02:11:53 |
| **Task 2** | **Without conditions and consequences** | **With conditions and consequences** |
| All UI states, all UI objects and all transitions | 02:25:08 | 04:18:14 |
| All UI states, unique UI objects and unique transitions | 01:16:55 | 03:10:00 |
| All UI states, unique UI objects and converted transitions | 00:59:13 | 01:11:21 |
| **Task 3** | **Without conditions and consequences** | **With conditions and consequences** |
| All UI states, all UI objects and all transitions | 01:34:36 | 02:36:56 |
| All UI states, unique UI objects and unique transitions | 00:46:32 | 01:39:30 |
| All UI states, unique UI objects and converted transitions | 00:37:56 | 00:41:48 |

### Interpretation of the Results

Given the statement that predictions using the underlying KLM theory of CogTool are within 20% of the execution times of skilled users (John and Salvucci, 2005) the results from Table 7.4 can be interpreted as following.

#### Task 1

The relatively long task 1 consists of 48 UI states with 789 UI objects and 486 different transitions that need to be modeled. The time for modeling all of these elements including conditions and consequences is predicted to be 10 hours, 39 minutes and 19 seconds. Under the assumption that the prediction is 20% too low the modeling can take up to 12 hours, 47 minutes and 9 seconds, which is the longest predicted modeling time for all considered cases. In contrary, when modeling only the UI states, the 238 unique UI objects and 48 transitions without conditions and consequences (i.e. the simulated interaction path) then the

predicted modeling time is only 1 hour, 45 minutes and 24 seconds. Under the assumption, that this prediction is 20% too high, the time drops to 1 hour, 24 minutes and 19 seconds.

**Task 2**

The SIM of task 2 consists of 22 UI states with 311 UI objects and 205 transitions. In case all elements are modeled by hand and all transitions have conditions and consequences the predicted time is 4 hours, 18 minutes and 14 seconds (+ 20%: 5 hours, 9 minutes and 52 seconds). When only unique UI objects and converted transitions without conditions and consequences are modeled this time is reduced to 59 minutes and 13 seconds (-20%: 47 minutes and 22 seconds).

**Task 3**

The SIM of task 3 is the smallest of the considered cases with only 7 UI states, 202 UI objects and 113 transitions. When all of these elements are created by hand and the transitions have conditions and consequences then the predicted time is 2 hours, 36 minutes and 56 seconds (+20%: 3 hours, 8 minutes and 19 seconds). In case only unique UI objects and converted transitions without conditions and consequences are modeled by hand the predicted time is 37 minutes and 56 seconds (-20%: 30 minutes and 21 seconds).

### Discussion

The predicted modeling times give an overview of how long it takes a skilled expert to provide (a part of) the model of an application merely for evaluation purposes with the MeMo workbench. In contrast, the implemented approach does not require any additional time for providing a comparable model for the evaluation besides the time it takes for modeling the application using the model-based development process itself.

The predicted times from above only present an excerpt from the overall modeling process but already illustrate the potentially saved time and reduction of complexity. Usually, when modeling an application with MeMo (and other simulation-based approaches such as CogTool-Explorer) the SIM contains more functionality than just the simulated interaction path for one specific task. Thus, the SIM in most cases is more complex than it is in the considered scenario where it only reflects the simulated paths of the three tasks. A SIM as complex as the model for task 1 in the scenario where all UI states, all unique UI objects and all transitions are modeled is a more common scenario and so is the predicted time for modeling of several hours. Furthermore, some side constraints were

not considered in this analysis; e.g. modeling the system information pool and aligning the evaluation models with the development models.

One can argue that import functionalities for different AUE tools exist or could be tailor-made, which ease the process of creating a model of the application by automating parts of the process. For example CogTool-Helper (Swearngin et al., 2012) allows importing UIs written in Java and the OpenOffice interface to CogTool, which is a big achievement and reduces required modeling steps. Still, the interaction logic needs to be provided by the designer, whether by demonstrating a specific task by hand or by describing test cases in the GUITAR format. When evaluating different iterative designs or adaptation variants this process needs to be repeated. Similarly, MeMo has an import functionality for web-based interfaces that produces UI states with UI objects. But the interaction logic also needs to be provided by the evaluator; i.e. the transitions with conditions and consequences need to be modeled and attached to the imported UI objects in the appropriate UI states. Even more important in the scope of this thesis is that none of these import functionalities suffice for providing information stemming from the development process and thus lack the additional benefits (see Section 7.2.2 and 7.3).

### 7.2.2. User Task Model - Case Study: Effort for Providing the UTK

A further benefit of the approach is that not all interaction steps need to be predefined in the UTK for creating goal-directed interaction behavior. Instead, several implementations of the user task model and the user interaction model can be achieved with the help of the concepts described in Section 5.4 and described for implementation in Section 6.2 and Section 6.3.

Examples of these combinations of implemented strategies are depicted in Figure 7.4. It displays a simulation of task 2 (see Table B.2 on page 153) and lists the UI object and its caption that need to be interacted with in each interaction step.

Additionally, the further columns of Figure 7.4 display:

- Information that was directly used without any further modification from the UTK (indicated by white background color),
- Repeated interaction steps that are annotated with the flag *executecount* in the task file (see Table 6.1, indicated by yellow background color),
- Navigational steps that can be retrieved from the task model of the MASP during simulation (see Section 5.3, indicated by green background color), and

| Interaction Step | Type of UI Object | Caption of UI Object | All Interactions Predefined | Manipulating Interactions Steps only | Enh. Semantic Processing (Task & AUI Analysis) |
|---|---|---|---|---|---|
| 1 | Button | Rezeptsuche starten | Rezeptsuche starten | *MASP Task Model* | *MASP Task Model* |
| 2 | Checkbox | Italienisch | Italienisch | Italienisch | Italienisch |
| 3 | Checkbox | Französisch | Französisch | Französisch | Französisch |
| 4 | Checkbox | Nachtisch | Nachtisch | Nachtisch | Nachtisch |
| 5 | Button | Suche starten | Suche starten | *MASP Task Model* | *MASP Task Model* |
| 6 | List Entry | Panna Cotta | Panna Cotta | Panna Cotta | Panna Cotta |
| 7 | Button | Mit Auswahl fortfahren | Mit Auswahl fortfahren | *MASP Task Model* | *MASP Task Model* |
| 8 | Button | + | + (executecount = 2) | + (executecount = 2) | + (executecount = 2) |
| 9 | Button | + | | | |
| 10 | Button | Zutaten überprüfen | Zutaten überprüfen | *MASP Task Model* | *MASP Task Model* |
| 11 | List Entry | 3 Blattgelatine | 3 Blattgelatine | 3 Blattgelatine | Blattgelatine Habe ich |
| 12 | Button | Habe ich | Habe ich | Habe ich | |
| 13 | List Entry | 75 ml Caramelsirup | 75 ml Caramelsirup | 75 ml Caramelsirup | Caramelsirup Habe ich |
| 14 | Button | Habe ich | Habe ich | Habe ich | |
| 15 | List Entry | 375 g Sahne | 375 g Sahne | 375 g Sahne | Sahne Habe ich |
| 16 | Button | Habe ich | Habe ich | Habe ich | |
| 17 | List Entry | 3 Vanilleschote | 3 Vanilleschote | 3 Vanilleschote | Vanilleschote Habe ich |
| 18 | Button | Habe ich | Habe ich | Habe ich | |
| 19 | List Entry | 39 g Zucker | 39 g Zucker | 39 g Zucker | Zucker Habe ich |
| 20 | Button | Habe ich | Habe ich | Habe ich | |
| 21 | Button | Einkaufsliste erzeugen | Einkaufsliste erzeugen | *MASP Task Model* | *MASP Task Model* |

**Figure 7.4.:** Comparison of required UTK when simulating task 2 without any information from development models (all interactions predefined), with using the application's task model (manipulating interactions steps only) and when using the application's task model in conjunction with the AUI model (enh. semantic processing).

- Composed information from the UTK that is decomposed and mapped to different UI objects during simulation using the task and AUI analysis (see Section 6.3, indicated by purple background color).

### Discussion

Using the example of task 2 that is displayed in Figure 7.4, several interaction steps are derived implicitly during simulation. Each time a navigational step needs to be performed and no appropriate UTK is provided, the task model of the application is queried and the associated UI object is determined so that the text of the caption is added to the UTK. As a restriction, this only applies to cases when expert interactions are evaluated. Still, for task 2 this saves five interaction steps for which no UTK needs to be provided. Additionally, using the enhanced semantic processing module capable of analyzing the task and AUI structure, another five interaction steps can be saved. One can argue that this is a composition of information that can otherwise be provided in two different information assignments. Yet, the module also determines the correct order and, using the semantic similarity measures, is also moderately capable of dealing with renamed UI objects (e.g. in adaptations or iterated UI variants). Thus, this allows simulating a variety of UI versions with the same UTK without the need of forwarding minor changes of captions from UI elements for manipulating interaction steps to the task files. Table 7.5 lists the results and savings for task 1, 2 and 3 for all of the discussed variants.

**Table 7.5.:** Numbers of provided information assignments in the UTK for task 1, 2 and 3 using different implementations of the UIM.

| **Task** | **Simulated Interaction Steps** | **All Interactions Predefined** | **Manipulating Interactions Only** | **Enh. Sem. Processing (Task & AUI analysis)** |
|---|---|---|---|---|
| Task 1 | 41 | 40 | 34 | 20 |
| Task 2 | 21 | 20 | 15 | 10 |
| Task 3 | 8 | 8 | 4 | 4 |

To sum up, when using the integrated approach, information from the development models not only saves time when creating the models required for representing the application (see Section 7.2.1) but also saves time and effort when providing the user tasks. Furthermore, especially in cases when navigational steps are renamed in adaptations or sequential iterations of the same UI there is no need to modify the UTK because required information is added on the fly during simulation.

Finally, it needs to be remarked that for the considered cases at least the manipulating interaction steps have to be predefined, which is why a complete automation of providing all information assignments of the UTK was disregarded. As described in Section 5.3 such an approach would include random behavior for task relevant parts; e.g. choosing an arbitrary meal for an arbitrary number of people and a random list of available ingredients. While such behavior might be sufficient for testing the robustness of the UI and task models, it does not contribute to the evaluation of a goal-directed interaction behavior and specific scenarios; e.g. comparing the same user input for different adaptation variants.

## 7.3. Improvements to Cognitive User Models and Predicted Execution Times

In this section a case study is highlighted that was conducted with the interactive cooking assistant from Section 7.1 and the implemented architecture. In particular, cognitive user models are presented that are derived automatically using the implementation based on MASP UI development models, the user interaction model and the integration of CogTool as described in Section 6.5. This is supported with results from two user studies described in more detail by Quade et al. (2014). Based on observations from the initial user study and a comparison of predicted interaction times by CogTool, three hypotheses for improving the predictions are provided. Subsequently, it is shown how required information for applying these hypotheses automatically can be derived from development models of the MASP. Finally, the hypotheses are evaluated by using the implementation and comparing the predictions with observations from a second user study.

Below, a short summary of the initial user study is given and subsequently observed findings and their impact on the implementation of the architecture are presented.

### 7.3.1. Initial User Study: Experimental Setup

The experimental setup was divided into two user tests. An initial user test with the cooking assistant was designed for observing user interaction, deriving hypotheses from the observations and predicting task execution times using the architecture in combination with CogTool. A validation user study was conducted later (see Section 7.3.3) to validate the hypotheses and modifications made to the implementation with different users, adapted versions of the cooking assistant and different interaction devices.

### Preparation and Execution

The initial user study was conducted in May 2013. 4 female and 6 male participants with an average age of 29 years took part. As depicted in Figure 7.5 the participants acted in a kitchen environment of a living lab at the TU Berlin while the cooking assistant was presented on a 19" touch screen that is mounted to a cupboard above the sink.

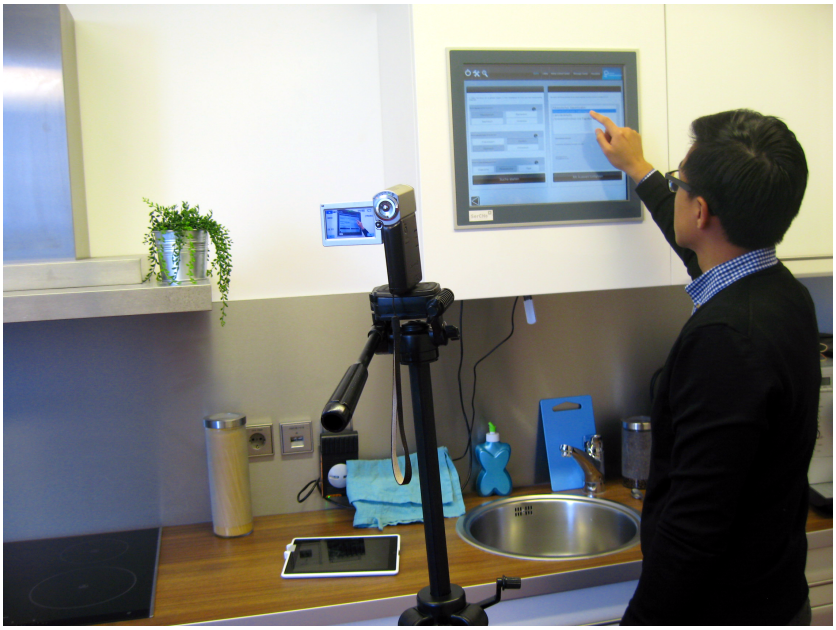

**Figure 7.5.:** Experimental setup depicting the cooking assistant on the mounted touch screen for the user study conducted in May 2013.

The evaluation described in this thesis was the last part of the experiment (after about 40 minutes of interaction) that consisted of multiple parts of interaction with the cooking assistant. Hence, all participants had worked several times with the recipe finder and the touch screen so that they were accustomed to the interaction flow and peculiarities of the whole interactive system. Thus, they were considered to be trained with the application and consequently matched the requirements for a time prediction using cognitive user models representing expert interaction.

### Logging and Annotation

All actions of the participants were recorded on video (see placement of camera in Figure 7.5) and logged by the MASP using an extended logging framework that was imple-

mented for the specific needs of this study. Table 7.6 gives an overview of the logged information and their use for evaluation.

**Table 7.6.:** Information logged during user tests for evaluation.

| Logged Information | Values | Use for Evaluation |
|---|---|---|
| **Event types of the UI** | ◦ Client rendered UI<br>◦ Focused UI element<br>◦ Clicked button | Is used to distinguish different types of UI input and output events and to monitor the interaction process between user and application. |
| **Time when rendering of the UI is finished** | ◦ Time in ms | Is used to derive system response times (and potential rendering lags). |
| **Time of a user interaction** | ◦ Time in ms | Is used to derive the times for specific interactions and the overall task execution times. |
| **X/Y coordinates of user interaction (click or touch)** | ◦ Numerical value of coordinates | Is used to check where users touched the screen and if they hit the UI element or to identify clicks with errors. |
| **Performed user task** | ◦ String value of task id | Is used for tracing MASP task model performance and the interaction flow. |
| **Performed AUI element** | ◦ String value of AUI element id | Is used for tracing interactions on MASP AUI elements. |
| **Performed FUI element** | ◦ String value of FUI element id | Is used for tracing interactions on FUI elements of the client and to derive UTK for the simulation. |
| **Selected elements from lists** | ◦ String value of caption | Is used for tracing selected elements from lists and to derive UTK for the simulation. |

With the help of the tool ELAN (Wittenburg et al., 2006) the logging data and the recordings from the video camera were synchronized to derive and annotate user interaction errors, such as wrong and unrecognized clicks in addition to the logged values from Table 7.6.

Logged interactions with errors were discarded because valid predictions can only be made for interaction traces free of errors according to the requirements of time predictions for expert interaction. However, this left too few complete tasks for statistical analysis of complete task execution time predictions from this study. So, times between pairs of

clicks were focused instead. Thus, 78 out of 447 logged user interaction steps had to be removed from analysis. The remaining clicks formed 218 valid pairs of clicks.

### 7.3.2. Initial User Study: Results, Hypotheses and Improvement of Predictions

As described above, main goal of this study was measuring times in between user interaction steps (clicks) and comparing these times with predictions of CogTool. For the evaluation interactions were distinguished into four different groups that are annotated exemplarily in Figure 7.6.

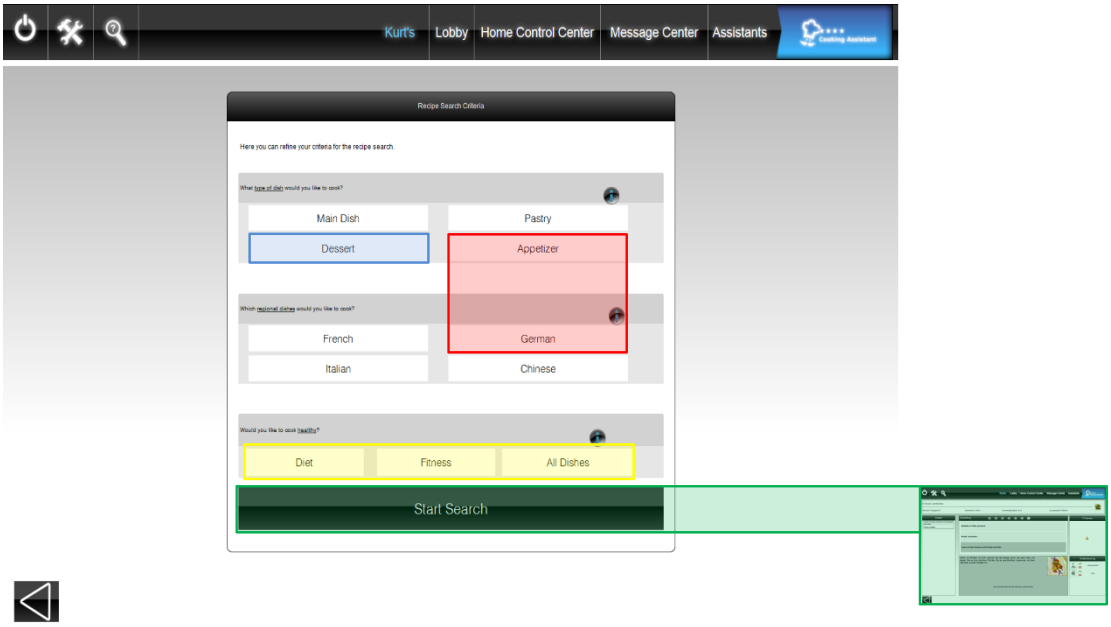


**Figure 7.6.:** Graphical user interface for selecting the recipe search, annotated with examples for types of clicks - blue refers to same button, yellow refers to same group, red refers to other group and green refers to new screen.

The four groups can be derived from the MASP models and are distinguished as follows:

- *Same button* refers to consecutive interaction steps with the same button.
- *Same group* refers to consecutive interaction steps with UI elements from the same group of UI elements (e.g. list entries, radio buttons or check boxes).
- *Other group* refers to consecutive interaction steps with UI elements from different groups of UI elements (e.g. at first selecting an entry from a list and then clicking on a button that is not part of that list, but already visible on the same UI screen).

- *New screen* refers to consecutive interaction steps with UI elements from one UI screen and the next following UI screen.

### Observed Results and Predicted Execution Times of CogTool

As a next step, a time prediction using the implemented architecture in combination with a standard version of CogTool[28] (Out-of-the-box - Ootb) was applied and compared with the observed interaction steps (see average times and predicted values for Ootb in Figure 7.7).

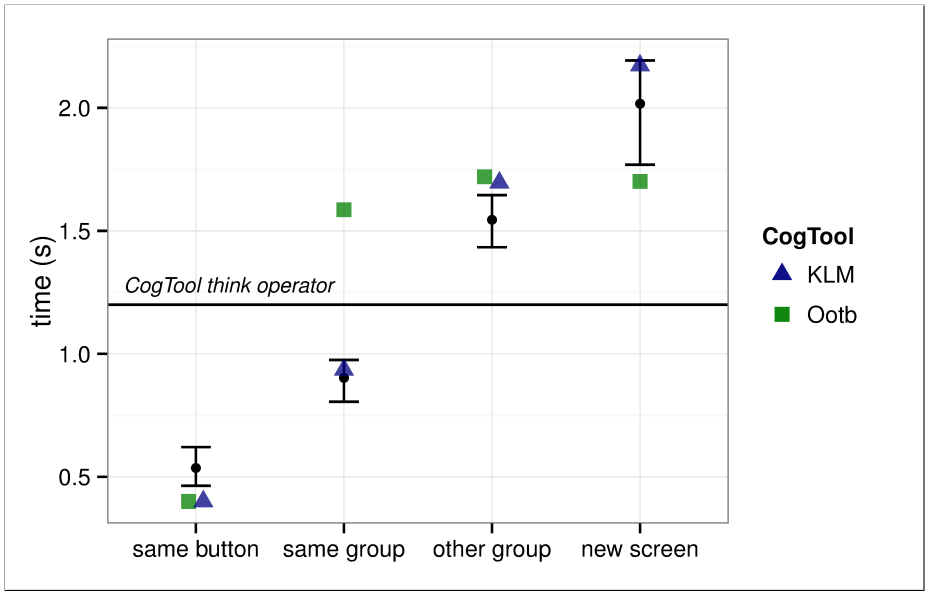


**Figure 7.7.:** Average times between clicks on different UI elements for the initial user study - taken from Quade et al. (2014): KLM refers to the CogTool model based on the three hypotheses; Ootb refers to the basic CogTool.

Based on the observations and predictions, the coefficient of determination ($R^2$) was computed for the Ootb version of CogTool for predictions of pairs of clicks ($R^2 = .597$). As a next step, three hypotheses from the observed interaction data were derived to improve the automatically created cognitive models.

[28] CogTool version 1.2.2 available for download from https://github.com/cogtool/cogtool/releases - Accessed in December 2014.

#### Hypothesis I: Units of Mental Processing

The first hypothesis targets interaction steps of users performed on UI elements that are grouped by their semantics.

**Observation from user study**

Consecutive clicks on UI elements within the *same group* of UI elements are performed faster than clicks between UI elements from *other group* of UI elements.

**CogTool (Ootb) time prediction**

CogTool (Ootb) does not sufficiently match the observed interaction execution times. The predicted times for clicks within the *same group* are too high. Observed times are even below the automatically added generic time for the think operator (1.2 s) and thus cannot be predicted correctly by CogTool (Ootb).

**Hypothesis**

CogTool (Ootb) uses a specific preset of the KLM heuristics. According to the KLM theory the placement of think operators needs to be modified under specific circumstances. Only one think operator needs to be placed in front of actions belonging to the same *cognitive unit* (same button and same group); e.g. when a user enters a name there would not be think operators before each entered letter of the name but only in front of the whole sequence. But, the placement of these operators requires human interpretation of the UI to identify these cognitive units.

**Implementation using UI development models**

Using the integrated approach, such cognitive units can be automatically extracted from the UI development models. Similar to the analysis of the AUI structure described in Section 6.2 the mappings between the UI objects of MeMo and the runtime FUI elements and the AUI model are established to reason about the semantic structure of the UI. In this case the assumption is that FUI elements that are related semantically are modeled using the same AUI element (of type *choice*). Thus, they are also grouped by their semantics on the level of the AUI model and can be used to identify cognitive units in an automated way using the UI development models. Consequently, the specific AUI element of each FUI element is queried during conversion to CogTool. In case FUI elements share the same choice AUI element, they are considered to be a cognitive unit and the generation of the CogTool model is altered by removing all but the first think operator prior to consecutive interactions on these FUI elements from the same AUI element (Figure 7.8). In a simplified way clicks on the *same button* are queried

by comparing consecutive interactions on the same FUI element and treated in the same way.

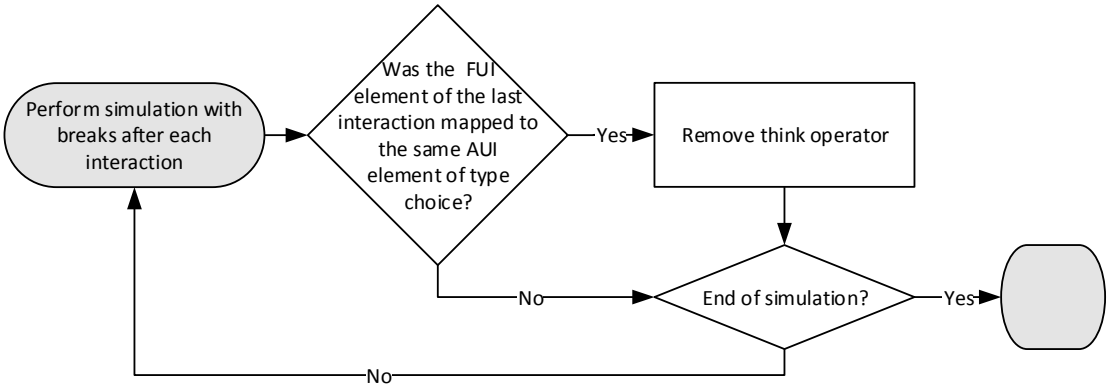


**Figure 7.8.:** Schematic implementation of hypothesis I.

#### Hypothesis II: System Response Times when Loading New Screen

The second hypothesis refers to the automatic detection of new UI screens and associated system response times.

##### Observation from user study

As the test setup was using a web-based client-server architecture the system response times required for calculating and rendering the UI of new screens for the specific client took about 500 ms before the screen was rendered and could be perceived as such by the participants (cf. page load time for web pages (Egger et al., 2012)).

##### CogTool (Ootb) time prediction

CogTool (Ootb) predicts interaction times that are too low compared with the observations for interactions that fall into the category of *new screen* when compared to interactions of the category *other group*.

##### Hypothesis

The underlying KLM does not consider that system response times used to render UI screens need to be added to the think operator instead of being shadowed, in case they are shorter than the think operator, because new information can only be perceived and processed by users when the UI is rendered.

##### Implementation using UI development models

As system response times to user interactions (e.g. influenced by activated system

tasks) may vary between different platforms and devices, these need to be measured or estimated once. However, the approach benefits from the fact that the evaluation of the models is directly combined with testing the real application; i.e. the FUI. This specifically allows measuring and including system response times into the conversion process to CogTool in an automated way. After each simulated user interaction step the PTS is checked for changes. If so, it can be assumed that a new UI screen was rendered. In this case the system response time is automatically added in sequence with the think operator because users cannot perceive information from the following UI screen until the rendering is finished. If the PTS did not change, the system response time is added conforming to the CogTool implementation and might be shadowed by a (parallel) think operator (Figure 7.9).

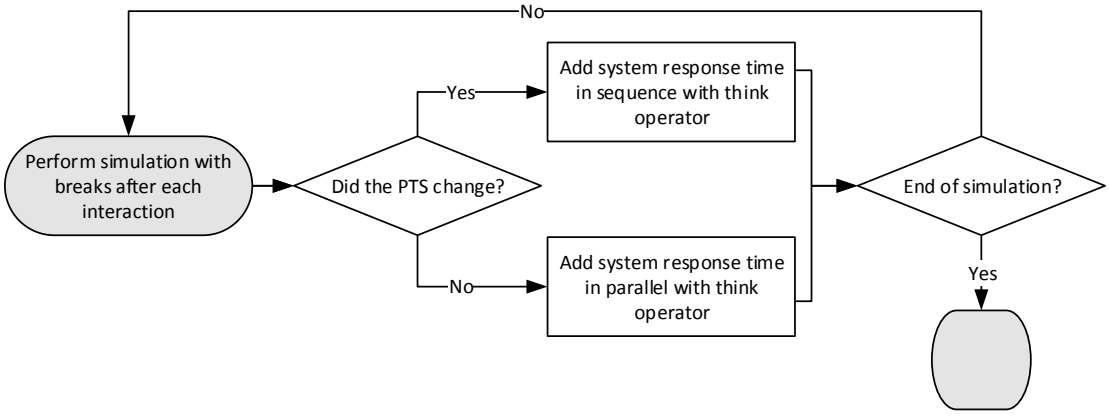


**Figure 7.9.:** Schematic implementation of hypothesis II.

#### Hypothesis III: Monitoring

The third hypothesis targets users that are monitoring if their actions were correctly recognized by the system.

#### Observation from user study

Interactions within the *same group* took about 360 ms longer than interactions on the *same button*.

#### CogTool (Ootb) time prediction

The predictions of CogTool (Ootb) are too high. After applying hypothesis I and removing the think operator for interactions within the same group the prediction is too low and does not account for the additional 360 ms.

#### Hypothesis

The touch screen used in the initial user study had moderate reliability, so users were checking (waiting) if actions were recognized; e.g. a list entry was marked as selected. Consequently, an additional monitoring step consisting of the required time to display the change on the screen (300 ms) and the time for the user to notice the change has to be added.

#### Implementation using UI development models

During conversion to CogTool the system response time and an additional *look-at* operator are added automatically to model a user monitoring if the performed action is reflected in the GUI. For this purpose, the FUI type of the UI element is queried and, in case it is an interaction on a UI element that can be toggled; e.g. radio buttons and checkboxes, the cognitive model is altered accordingly (Figure 7.10).

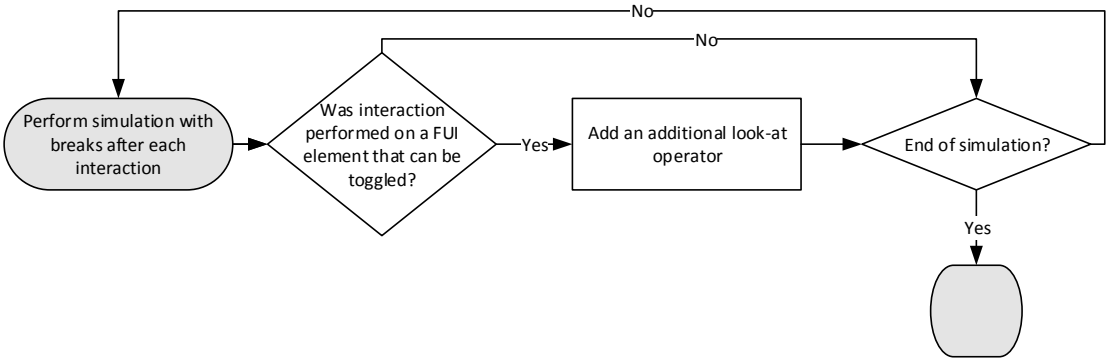


**Figure 7.10.:** Schematic implementation of hypothesis III.

#### Predicted Execution Times using the three Hypotheses

With $R^2 = .995$, the predictions of the adapted CogTool (KLM) model using the three hypotheses were improved significantly for the initial user study (compare predictions for KLM and Ootb in Figure 7.7 on page 126). The modifications to the cognitive models were target of a validation user study that is described next.

### 7.3.3. Validation User Study: Results & Discussion

The modifications made to the cognitive user model (Quade et al., 2014, pp. 7-8), based on the three hypotheses from the initial user study, were validated with observations from a second user study that was conducted in November 2013. For this validation

study the task domain was kept constant, but the user group (12 participants, 2 female and 10 male) and physical device (27" touch screen with better reliability) were varied in order to test the generalization of the hypotheses to a modified context of use (a bigger touch screen with different aspect ratio and minor adaptations of the UI). 1930 pairs of clicks were built for the evaluation and separated according to the classification into different groups described above.

### Results

The predictions based on the enhanced cognitive model (KLM) using the three hypotheses accounted for time differences between the new tasks of the validation user study very well (see Figure 7.7 for the initial study, Figure 7.11 for the validation study).

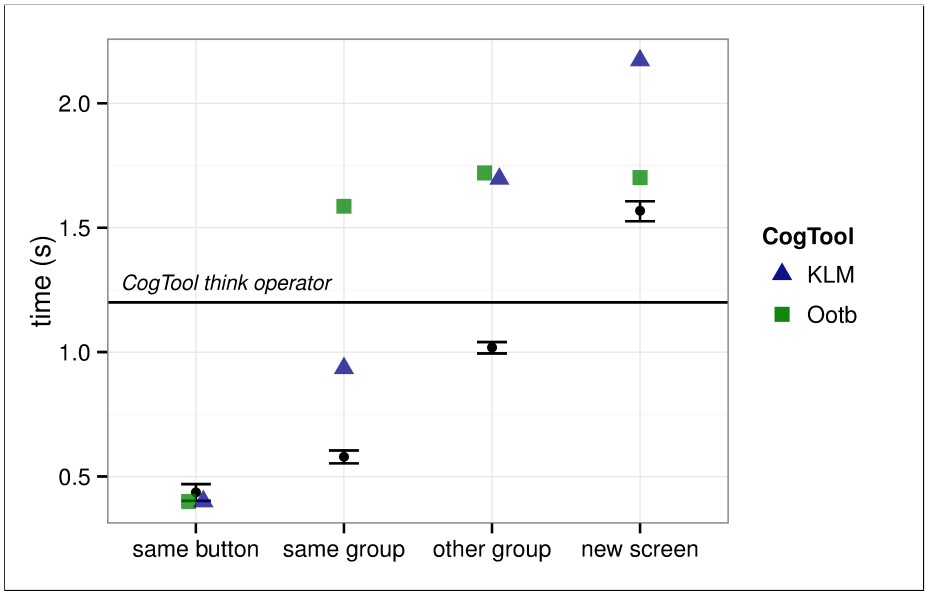


**Figure 7.11.:** Average times between clicks on different UI elements for the validation user study - taken from Quade et al. (2014): KLM refers to the CogTool model based on the three hypotheses; Ootb refers to the basic CogTool.

An overall shift from 1.41 s to 1.04 s in execution times for consecutive clicks was not predicted correctly by the model (compare drops in average execution times between Figure 7.7 and Figure 7.11). Still, the improved results are reflected in an $R^2 = .927$ for the KLM version compared to $R^2 = .425$ for the predictions of CogTool Ootb.

The higher amount of observed interactions and less frequent interaction errors of the validation study also allowed for predictions for the overall task completion times. Figure 7.12 displays these predictions in seven categories indicated by the number of inter-

action steps (total clicks) and the number of interaction steps within the same group (same group clicks). Again, predictions made by the KLM version of CogTool using the results from the three hypotheses showed better results, reflected in $R^2 = .965$ compared to CogTool Ootb with $R^2 = .735$.

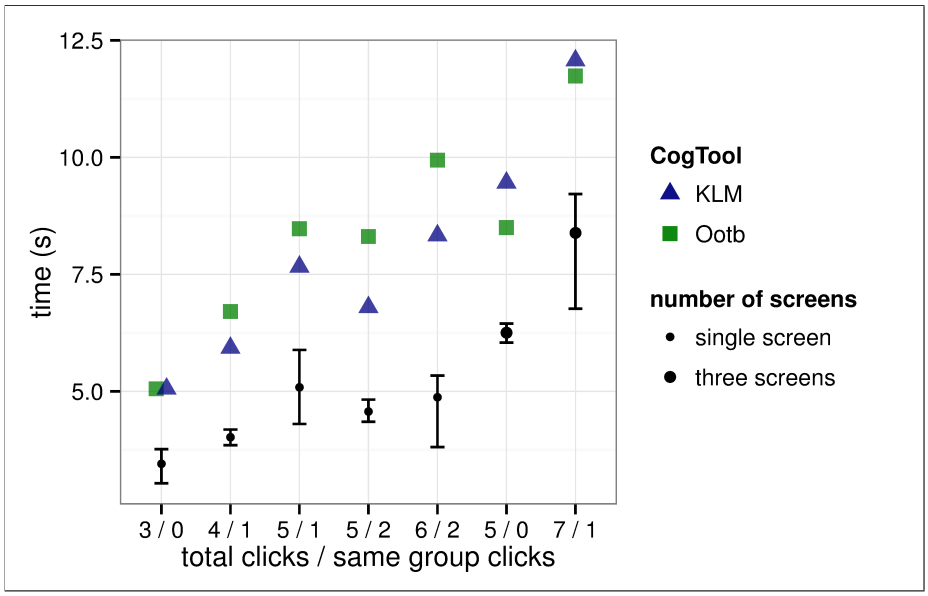


**Figure 7.12.:** Average task completion times for the validation user study - taken from Quade et al. (2014): KLM refers to the CogTool model based on the three hypotheses; Ootb refers to the basic CogTool.

### Discussion

The validation user study shows that information from the development models of the MASP significantly improves the goodness-of-fit of the results. Especially the implementation of the first hypothesis highly benefits from using the UI development models and would not be possible to this extent without the integrated approach. This is founded on the fact that the improved placement of think operators requires case-specific expert knowledge. Due to the automated approach, which accesses and analyzes the AUI model, designers do not need such extensive knowledge.

The implementations of the other two hypotheses also benefit from the development models and the fact that the models are executed in a runtime framework. One can argue that these two cases are simpler and thus require less expertise than the first hypothesis. Furthermore, the implementation of the third hypothesis only relies on information from the level of the FUI and thus could also be provided using different approaches; e.g.

on basis of the MeMo UI models or via manual inspection. Still, both implementations provide complete automation to the evaluation process and thus decrease the chance that human evaluators overlook these specific cases.

## 7.4. Summary

In the evaluation three scenarios are considered for demonstrating how the implemented approach addresses the main challenges of a complex and time consuming modeling process as well as improving evaluation results based on development models. The results demonstrated that the approach lowers the barriers for adopting AUE to the development process to improve the application models.

For reasons of comparison it is shown how much effort is involved when creating (an excerpt of) a model of an application with the MeMo workbench by hand as an example for a typical AUE modeling process. Several scenarios were targeted and the results and implications presented. Most important is the fact that with the help of the approach there is no further need to provide specific models for the applied evaluation method as the development models are directly used and interaction is simulated with the real application. This leads to savings of several hours of modeling work that can be spend for improving the application.

In a similar way the effects of the described enhancements to the processing of the UIM are shown when automatically inferring navigational interaction steps and analyzing the AUI model. On the one hand, this saves time when providing the UTK required for simulating a specific task and, on the other hand, allows simulating the same UTK with different adaptation variants.

Finally, improved cognitive user models were created without requiring further intervention by usability experts. For this purpose three hypotheses are presented that are based on observations from an initial user study and then were implemented and checked in a validation user study. UI development models and characteristics of model-based UI development were exploited for this implementation for automating the process with the goal of decreasing the level of expertise for evaluation. Even if usability experts are available, they can base their work on the automatically created CogTool models, sparing them from the laborious creation of UI mockups and storyboards.

Furthermore, with the help of the approach different UI adaptation variants can be compared in an objective way using less effort than modeling the application in CogTool. An example for application is predicting the efficiency of a UI for frequently vs. rarely

used functions. Especially in the domain of smart home environments, specific recurring functions are used quite often and thus profit from an efficient interaction.

Compared with further model-based evaluation approaches that are focusing on evaluations of task models (e.g. by Paternò (2005)), the integrated approach and the made improvements use AUI and FUI information and thus allow creating richer cognitive models. Especially, exact button positions (and their labels for matching purposes) can be used for optimized predictions using Fitts' Law (Fitts, 1954) without requiring data from real user tests. Finally, compared to approaches that focus on exhaustive task model evaluations, the degree of simulation for the described evaluation was limited to predefined interactions. By this means, the presented focus of the approach enables a complementary evaluation of detailed execution time predictions for expert interactions. But specifically this focus provides benefits when evaluating adaptive UIs because task execution times are a quantifiable criterion of usability that can be predicted automatically.

# 8. Conclusion

This conclusion presents an overview of the work described in this thesis. Below, the main contributions are summarized that were made in order to tackle the problems and shortcomings explained in Chapter 1. Subsequently, limitations of the presented approach are discussed and areas of future work are pointed out. In the last section a general conclusion is drawn.

## 8.1. Contributions of this Thesis

In Chapter 1 the underlying problems are described that arise when evaluating adaptive UIs on the one hand and when applying automated usability evaluation approaches on the other hand. While providing various benefits, such as objective comparisons and a reduction of required expertise, specifically automated usability evaluation approaches suffer from a costly and time-consuming modeling process when evaluating adaptive user interfaces due to their high complexity. Based on this problem statement the work in this thesis is motivated by combining both areas on the foundation of model-based UI development. On the one hand, this allows AUE processes to use information provided in development models and, on the other hand, designers can obtain usability evaluation results during early phases of development on the basis of their models. For this purpose Chapter 2 of this thesis gives an overview of fundamentals in human-computer-interaction, engineering of adaptive UIs and usability engineering. A specific focus is set on evaluation criteria and measures that can be obtained objectively and thus serve for comparing different variants of an application.

### Definition of Requirements

Required information is defined in Chapter 3 that allows for a generic approach of evaluating adaptive user interfaces in an automated way. The first four define the necessity to access information of the application and the context of use by a simulation-based automated evaluation approach. The last three state information that needs to be defined

for simulating interaction of specific users for a specific goal. In addition, the necessity for each required information and expected results of appropriate evaluations are given.

### Concept and Design of an Environment

Based on required information the approach of this thesis is described in Chapter 5 that makes use of information from model-based user interface development. The approach features core components and explains required information transfer using specific processes and models. It is aligned with a common reference framework for the development of model-based adaptive UIs and describes ways for introducing specific behavior of the user model and obtaining information for simulating tasks.

### Implementation of the Environment

An implementation is highlighted in Chapter 6 as a proof of concept of the approach. For this purpose a simulation engine for automated usability evaluation is described that accesses required models and components of an existing runtime architecture for model-based user interfaces as well as a user model capable of simulating interaction with the models. Furthermore, with the help of a widely used tool for automated usability evaluation a cognitive architecture is applied that uses the simulation-based interaction process for predicting task execution times as criteria for comparing different design and adaptation variants of an application.

### Validation of Concepts of the Implemented Environment

In Chapter 7 an evaluation of the implemented architecture is presented that focuses on two main aspects. The potential savings in modeling the application for evaluation purposes are highlighted in a case study that analyzes the modeling time and effort for different adaptation variants and different tasks. Beyond that the evaluation demonstrates the ability of the approach to accomplish the goals of simulation with different levels of provided information. This is achieved by showing that implemented components analyze encoded information from the AUI and task models in order to provide navigational interaction steps and disambiguation of user tasks. Even more, the evaluation presents a further case study showing that automatically derived cognitive user models can be enriched using information from the development models during simulation with the result of more accurate execution time predictions.

## 8.2. Limitations

The scope of this thesis is narrowed to approaches of model-based UI development that provide means for executable models. The advantage of these approaches is that they implicitly satisfy required information of representing the application to the approach. Required information is encapsulated in the development models; e.g. the task model, the AUI and the CUI models. Potential follow-up states of the application are reached by interaction with the final UI or directly on the models (e.g. executing the task model or the AUI elements). The implementation makes use of the MASP, which was developed with a special focus on executing the models and deriving the final UI at runtime. Yet, other approaches related to UsiXML might not provide the same means and are focused on providing executability on the level of the final UI only. In a worst case, further models might not be available anymore or cannot be processed automatically, due to their informal nature. While there are approaches for reverse engineering more abstract models from the final UI for reasons of retargeting an application to a different platform, this limits the applicability of the implementation. However, if the development models are still available, Sottet et al. (2008) describe a possible solution based on mappings between the models. These mappings provide means for linking modeled information between all development models in a similar way as required by the approach.

A further limitation of the implementation is the evaluation of adaptations to the application that are not predefined on the task and UI models, but occur dynamically due to changes in the context of use during simulation. While the MASP provides means to such adaptations, they were not focused in the implementation and evaluation of this thesis. For this purpose a specific module for the simulation engine that manipulates the context of use during simulation needs to be integrated into the AUE engine.

## 8.3. Future Work

The following proposed areas of future work sketch ideas, questions, problems or solutions that are linked to this thesis but are beyond the scope of this work.

### 8.3.1. Supporting Developers of Model-based Adaptive UIs

In general, the integration of the approach into the development cycle could be increased. Two potential cases are sketched below.

The concept described in Section 6.3.1 for determining semantic and string similarities could also be applied for suggesting alternative captions for UI elements that might yield better fits with the expectations of users for specific tasks. A proof of concept has been implemented that looks up fitting synonyms for selected captions from different sources and displays these to the developer in the order of their predicted similarity values. While this approach is promising, more work needs to be done in order to better integrate it into the development cycle and to validate the benefits.

An improved approach for visual attention of the MeMo workbench (Ruß, 2011) could also be included directly into the development process. The resulting heat maps could be used for highlighting how information is perceived when performing specific tasks. This way the developers could be informed better and gain more insights into providing different adaptations or UI variants by directly seeing how users will perceive the user interface.

Finally, approaches that suggest solutions or improvements for found usability problems could be integrated. This has the advantage that the improvements could be applied to the development models for providing further adaptation versions of the UI, which could be analyzed subsequently. Based on the results, a cycle of prediction and improvement could be achieved for further support of developers.

### 8.3.2. Extensions to the Approach

Several possible extensions to the conceptual approach are outlined below.

#### Evaluations of Simulated Interaction Errors

Beyond predictions of execution times further criteria of usability and measurements for determining the quality of applications and adaptive user interfaces need to be implemented. One of these is the simulation of user interaction errors and possible correction strategies, which were excluded from the scope of the implementation of this thesis. However, the overall architecture allows for integrating these concepts into the process of simulation. First steps towards this integration are described by Halbrügge et al. (2015a,b). Especially for identifying reasons for interaction errors, such as omissions (forgotten interaction steps) and intrusions (unnecessary additional interaction steps), the development models can serve as a valuable source of information. On the one hand, this could be made available to the developer when defining the models during early development steps in order to highlight problematic design decisions. On the other

hand, this can help to improve specific instances of the user interaction model that focus on simulations of interaction errors and error recovery strategies for testing side effects of such interactions; e.g. in order to test the robustness and error tolerance of the application.

When evaluating adaptive UIs the complexity for providing ideal interaction paths for each version of the adapted application becomes increasingly high. Furthermore, when extending these approaches with regard to different user groups; e.g. users with special and specific needs; or to the domain of self-adaptive applications, it becomes more and more important to take interaction errors into account as well. These errors might occur due to various reasons; such as:

- A wrong understanding of the task; or
- The user might not be able to decode relevant information from the UI; or
- The user is not able to interact in the right way with the provided UI or the input device due to physical limitations, missing domain expertise or bad interaction capabilities of the device; or
- The adaptive application does not provide relevant information for accomplishing the user task due to misconceptions that are hard to foresee.

There exist more reasons and their effects on the interaction differ, but this short list of examples illustrates the need for covering interaction errors.

The conceptual approach is open for using e.g. a probabilistic approach in order to simulate user interaction variance. Possible reasons for errors are combinations of user characteristics, application attributes and dimensions of the context of use. This requires that probabilities need to be edited to the needs of the AUE approach and the goals of evaluation; e.g. testing many interaction paths or the error tolerance of the application. Such an approach for formalizing knowledge about the effect of combinations from user, application and context of use features on the interaction can be based on rules (Ruß et al., 2012). In the user interaction model, these rules can be applied to all three phases of simulated interaction: perception, processing and interaction execution. The probabilities are used to define the likelihood of correct actions, according to the provided UTK during each of these phases; i.e. if the correct UI element can be perceived, processed and executed as required by the user task. As a result, such an approach relies on input that is provided as information about:

- The user; i.e. characteristics that have an influence on the interaction process during these three phases. This information is provided in a static part of the user

interaction model and referred to as user characteristics, which contain persistent information that is modeled using approaches from *user modeling* for adaptive UIs and AUE.

- The application, obtained via the set of application models.
- The context of use that is provided in the context of use model.

For the needs of this approach, the rules could follow the structure of a simple *IF-THEN* schema, providing a precondition (IF) and a consequence (THEN):

- The precondition of these rules describes trigger points for rules to fire; e.g. the user has a tremor (stored as an attribute in the static information from the user interaction model) and there is a small button on a touch display (derived from information from the application's CUI model).
- The consequence describes the effects of the preconditions on the simulated interaction process; e.g. reducing the probability that the user performs the interaction correctly (during the interaction execution phase of the user interaction model).

The source for information for the preconditions of rules can be: context of use model, CUI output model, CUI input model and user characteristics. The consequences of the rules could be applied to the three internal phases of the user interaction model. Several such rules can be defined based on usability expert knowledge or experimental data; e.g. as described by Bastien and Scapin (1993) and Vanderdonckt (1994). The rules can also be based on tests with former prototypes of the UI or to provide specific intended behavior of the modeled users (compare e.g. the approach for modeling users with automata by Bezold (2009) described in Section 4.3.2). This way, non-expert interactions can be simulated and the effects of interaction errors can be evaluated when simulating different iterations with the probabilistic approach and thereby producing many different interaction paths (in the style of a *Monte Carlo* method).

Yet, the described approaches for simulating interaction variance or errors are not complete. For example, errors in the user's understanding of the task cannot be simulated this way. However, simulating interaction problems as described above is a first step towards a simulation of users with special and specific needs and the effects of applying adaptations to the application's UI layout. When evaluating plastic UIs this can provide results related to the application's robustness when adapting to a specific context of use and unexpected user behavior.

A further discussion point is the definition of rules and the value of their influence on the interaction process. The literature from above is a starting point for defining these

rules, but further problems arise when different rules interfere with each other or some aspects become too complex to be expressed in the *IF-THEN* schema. Nevertheless, such an approach is promising for limited problems, which can be described reasonably. The described problems might have big influences on the interaction process and deserve a further validation.

### Evaluations at Runtime

An evaluation at runtime of the application (Quade et al., 2011; Quade, 2012) provides a further scenario for an extension of this approach. For example, when adapting the UI to a different context, all potentially applicable adaptation variants of the UI could be simulated for the current task and predictions can be computed. Based on these automatically predicted execution times, whether annotated during development or predicted during runtime, the best fitting adaptation variant can be chosen; e.g. predicted times can serve as a hard constraint for excluding adaptations or for preferring a specific variant over others. The basic requirements for such an approach are similar to the ones described in this thesis. However, this also comes with side effects that need to be weighed against the benefits. The simulations and predictions should not increase the perceived system response times and users should not be confused by incomprehensible or unpredictable adaptations at runtime. One way to address this is to make underlying reasoning and causes for adaptations transparent to the user; e.g. by using concepts of a meta-UI.

### Evaluations of Multimodal Interaction

Finally, the domain of multimodal interaction is interesting for the approach as well. Both, the presented required information and the generic approach can be applied to multimodal interaction as well. Description languages such as UsiXML are specifically designed for multimodal interaction and thus provide required information that can be exploited for the integrated approach. Also, the MASP framework and the MeMo workbench that were applied in the implementation, aim for providing and evaluating multimodal interaction. Interesting aspects of such simulations are the choice of modalities for specific tasks and in a specific context of use. Criteria, such as execution time predictions might yield insights into the underlying decisions as the modality that can be performed faster might be preferred in specific cases. The same applies to the modeling of potential user interaction errors that are bound to (the availability of) specific modalities.

## 8.4. General Conclusion

This thesis describes an automated usability evaluation based on simulated user interactions with adaptive user interfaces. For this purpose, two research areas are used as inspiration and are combined for the main approach: automated usability evaluation and model-based user interface development. Especially the area of model-based UI development aims for creating adaptive UIs by reusing existing formal patterns and design decisions. In this thesis it is shown how this information can be exploited for providing even more automation to an automated usability evaluation approach.

Required information is summed up and an integrated approach is described that is capable of simulating interaction with UI and task models stemming from a model-based UI development process. Using a specific implementation, a proof of concept is demonstrated and potential savings in the modeling of the application and the task for evaluation purposes are described. Specifically for the domain of (automated) model-based evaluation, the approach presents a benefit that is highly important. Modeling the system usually requires a significant part of the modeling effort (Kieras, 2003). Thus, connecting the user model to the real system and still using underlying information from the development models simplifies model-based evaluation greatly. Beyond that, a case study is described where the automatically generated cognitive user models are improved by directly using information from the UI development models for improving usability predictions.

Possible areas of application arise, when taking a look at current challenges in model-based UI development. On the one hand, the presented approach provides a tool that can predict efficiency across different contexts of use and thus can be included in developing and maintaining plastic UIs. By this means, especially different design decisions, user interface layouts or adaptation variants can be compared with regard to task execution times. On the other hand, usability regression testing can be included into the development cycle after each iteration step, starting with early prototypes. The integrated approach can also be used to add automation to existing solutions (Abrahão et al., 2008). Finally, the applied methods for gaining information from UI templates in combination with dynamic CSS and JavaScript even allows automated extraction of UI information in the current trend of *responsive design*.

This thesis concludes with the statement that the presented approach, as well as others that are based on automation, aims for reducing the time and effort for early usability evaluations but they cannot fully replace user testing. Thus, both approaches ought to be applied complementary.

# Appendix A.

# Cooking Assistant Application - UI Screens and Export to MeMo

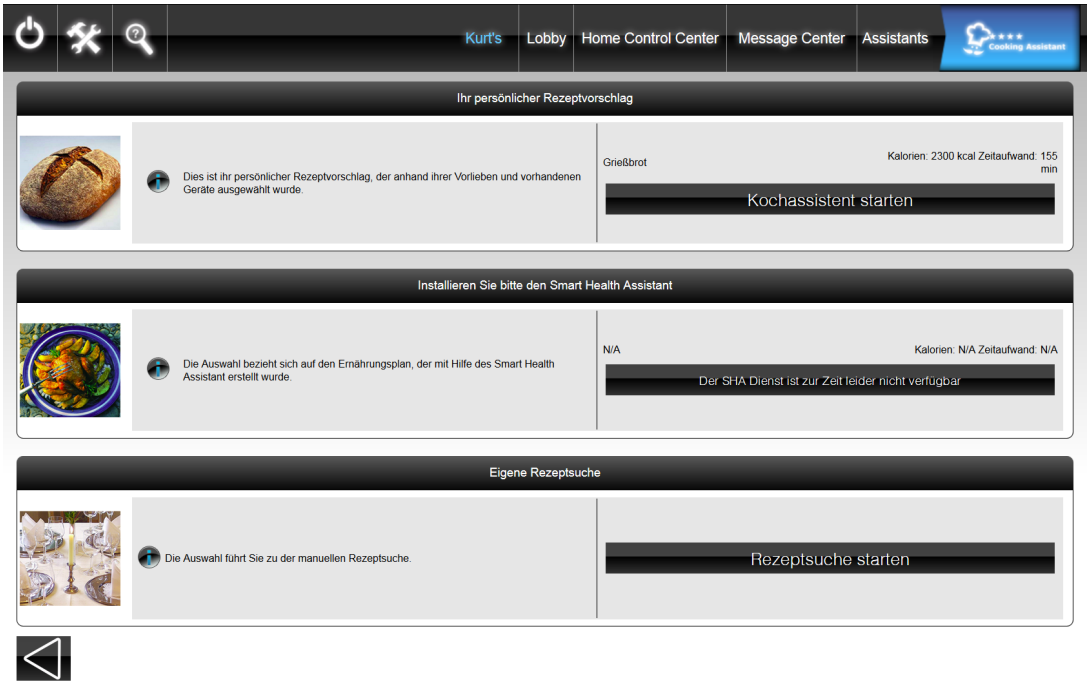


**Figure A.1.:** Initial dialog of the cooking assistant allowing to choose between recipe recommendations by a health assistant or from personalized preferences (upper and middle section) and entering a guided recipe search dialog (lower section).

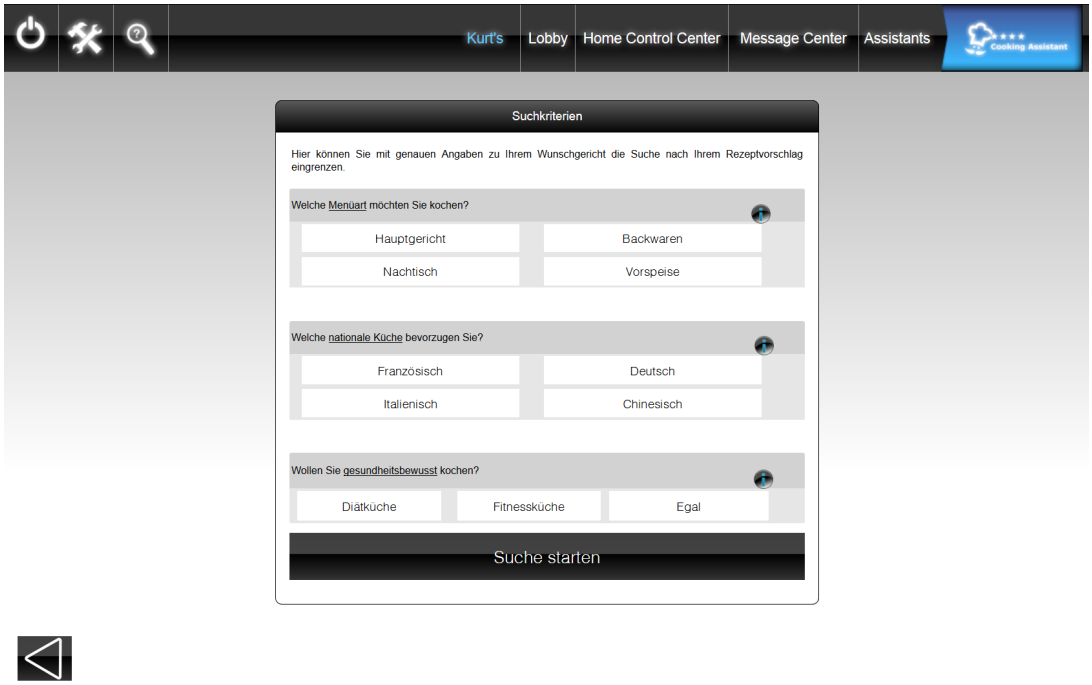


**Figure A.2.:** Dialog which allows defining search criteria such as the type of menu and the nationality of cuisine.

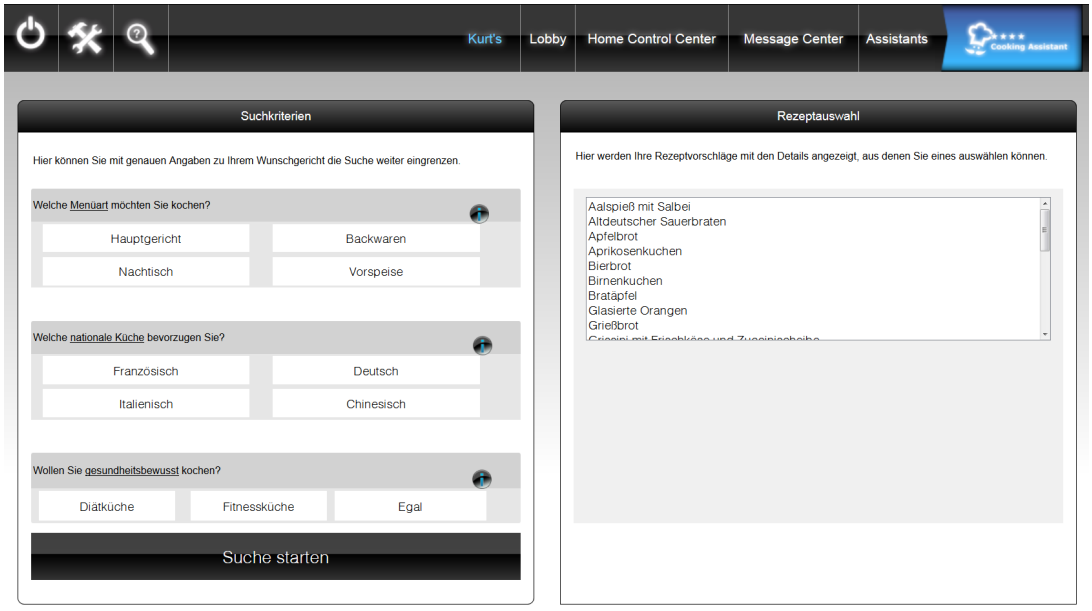


**Figure A.3.:** Enhanced search dialog with parallel display of results.

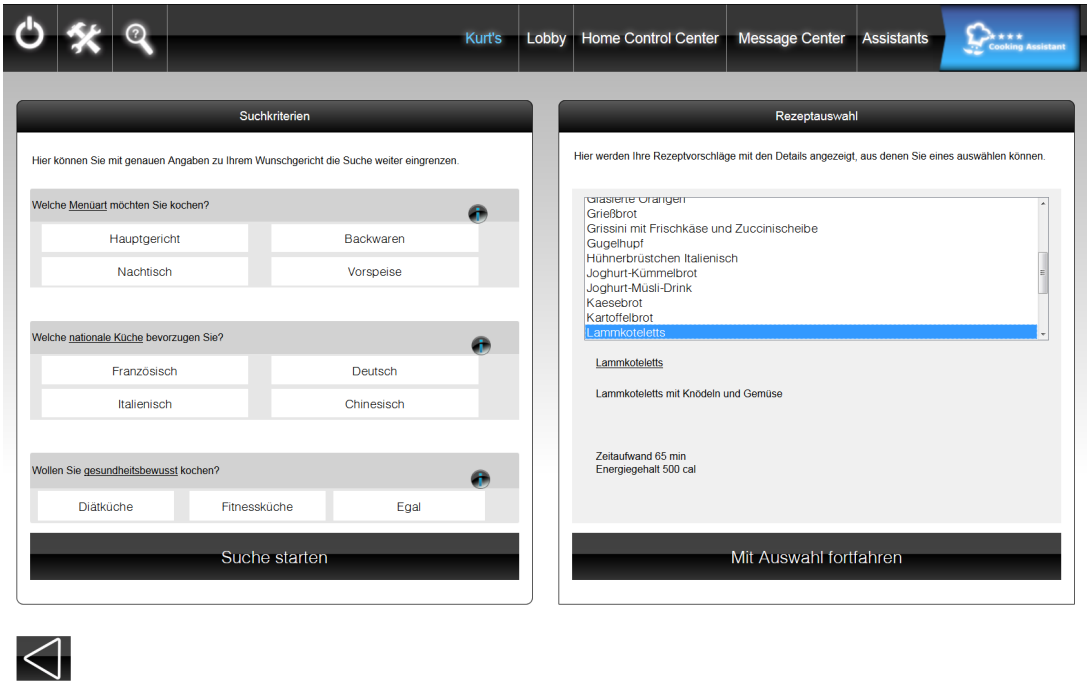


**Figure A.4.:** Same dialog as in Figure A.3 but with an enabled recipe and an extra button which allows proceeding to the next step.

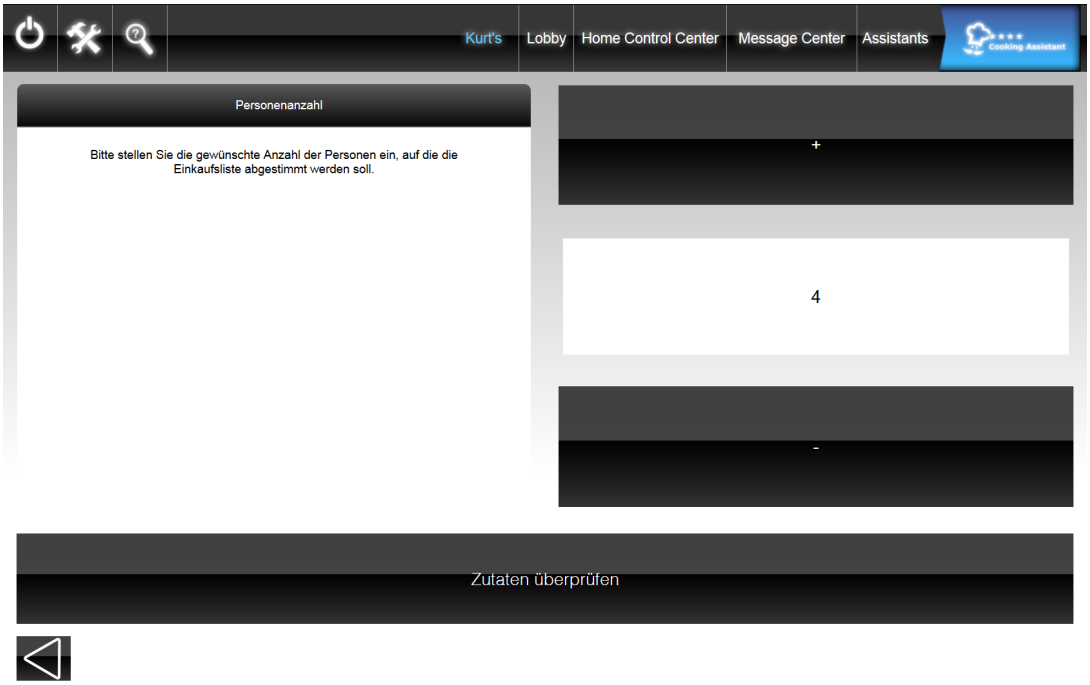


**Figure A.5.:** Dialog for detailing the amount of people for whom to cook the meal.

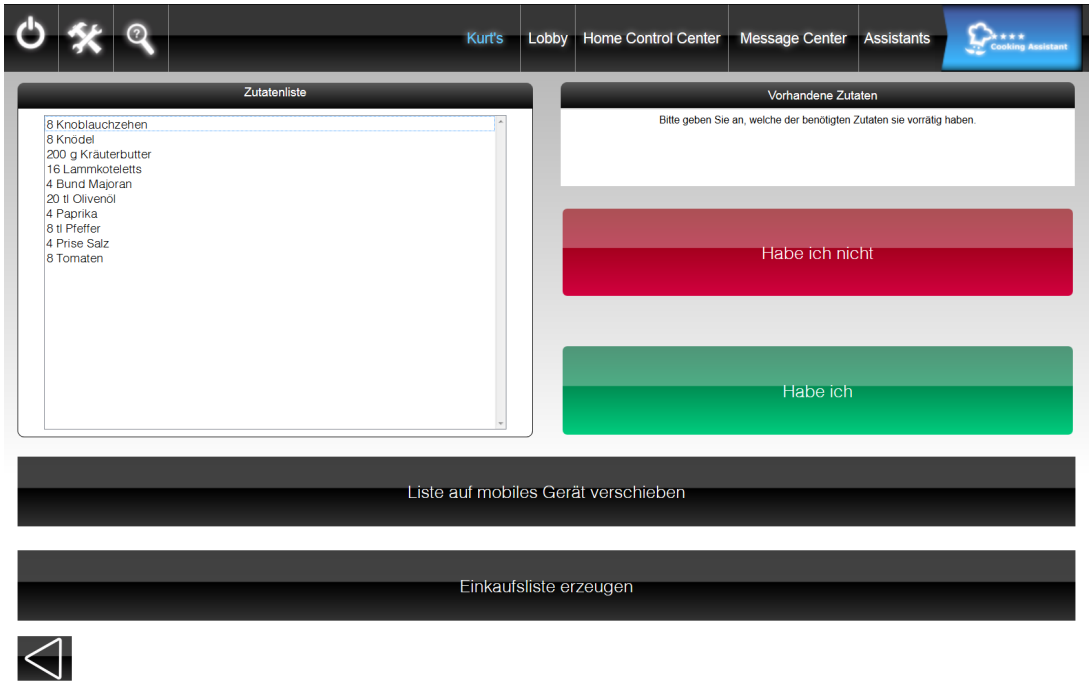


**Figure A.6.:** In this dialog the user is asked to specify which ingredients are already available in the household.

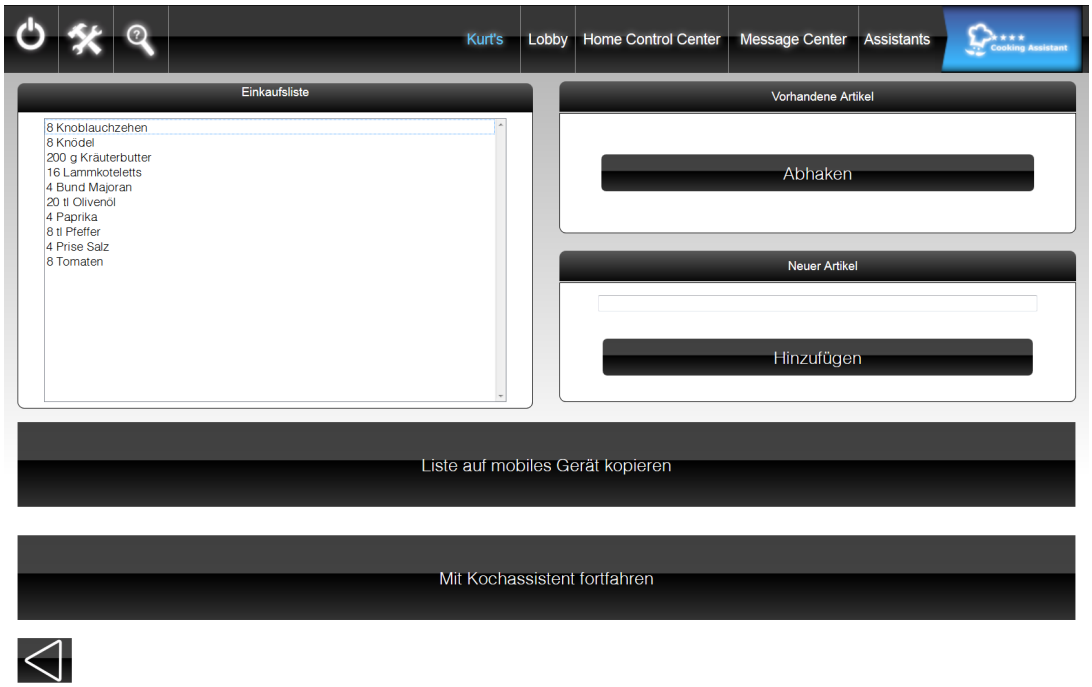


**Figure A.7.:** This dialog presents the missing ingredients and allows generating a shopping list or to proceed with the next step of the cooking assistant.

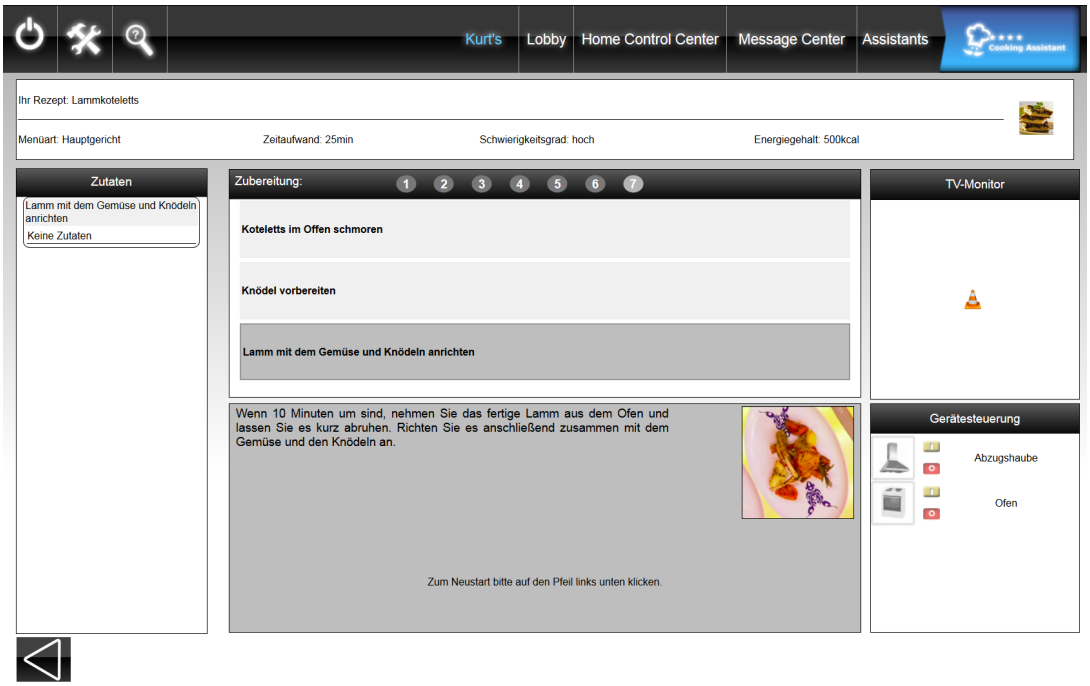


**Figure A.8.:** The last dialog guides through each preparation step.

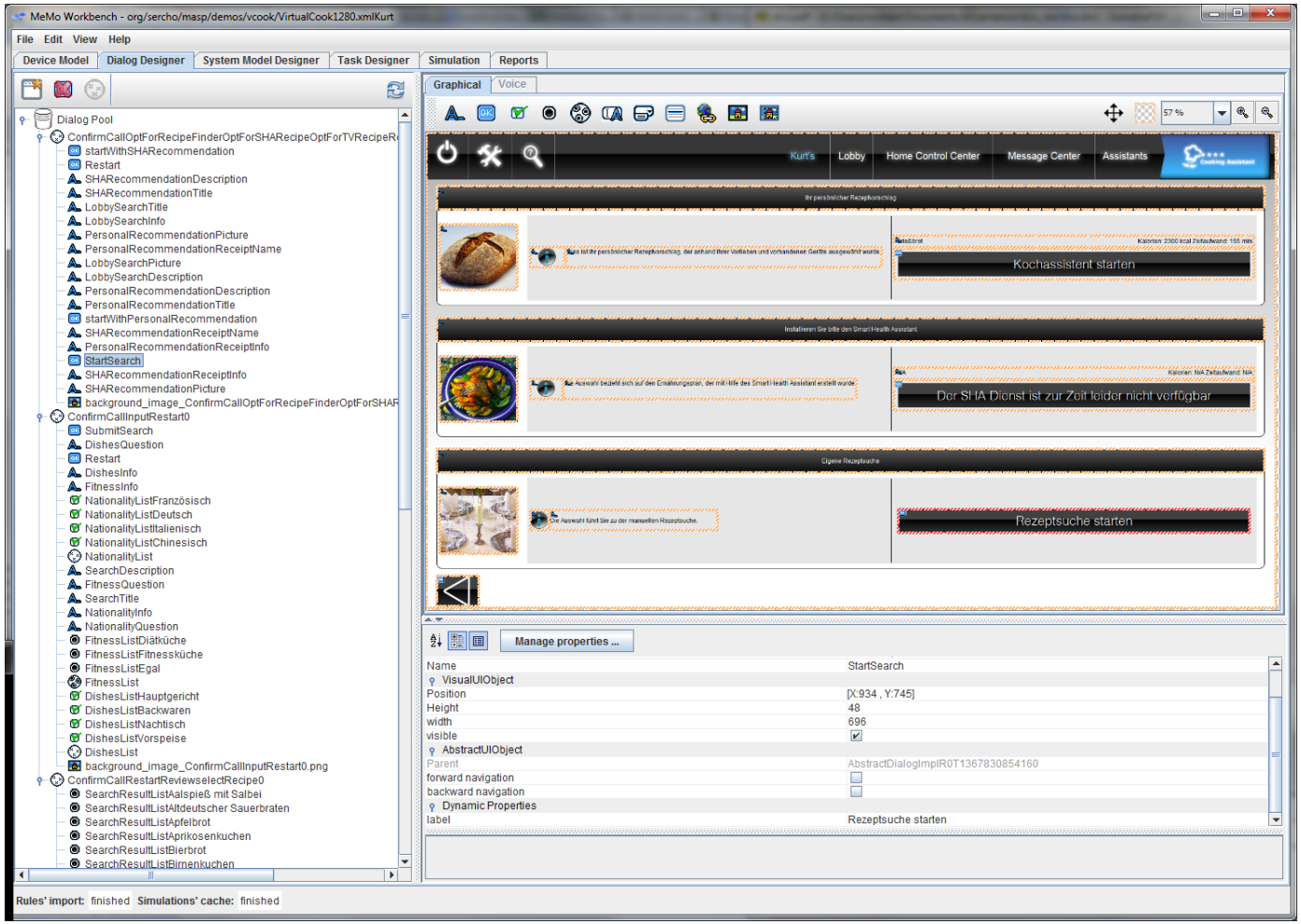


**Figure A.9.:** Screenshot of the MeMo workbench (Dialog Designer) with an imported model from a dialog of the cooking assistant showing the types of the UI elements, their sizes, positions and captions (compare with Figure A.1 on page 143).

# Appendix B.

# Evaluation - Task Models, Modeling Time Prediction with CogTool

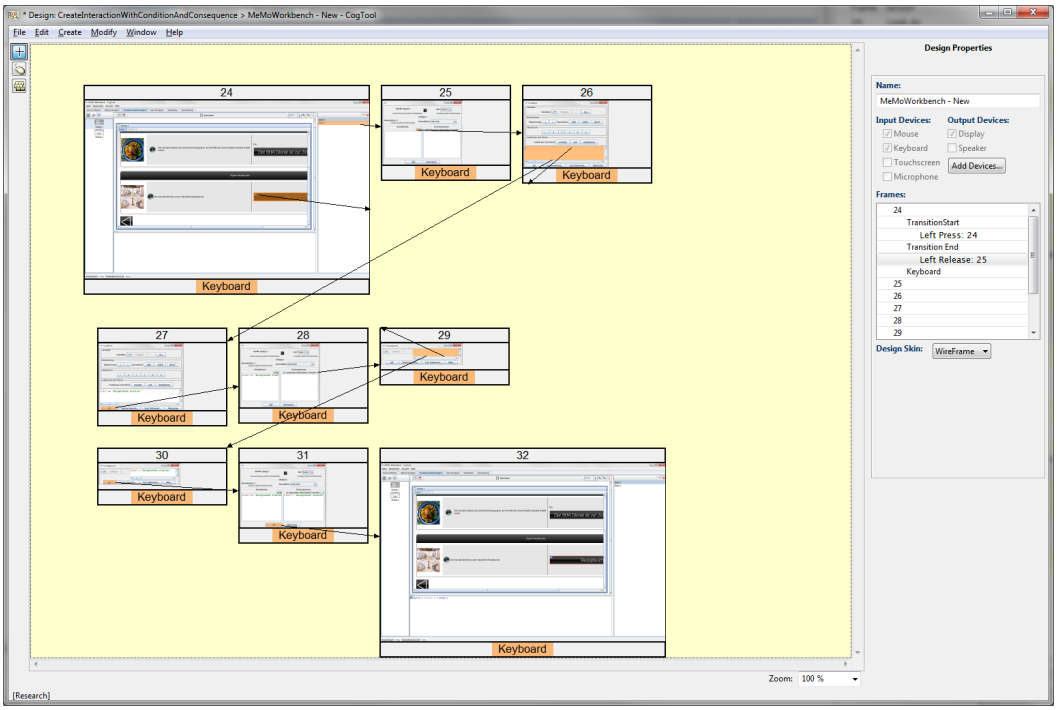


**Figure B.1.:** Design of the CogTool model of the MeMo workbench when modeling a transition using conditions and consequences.

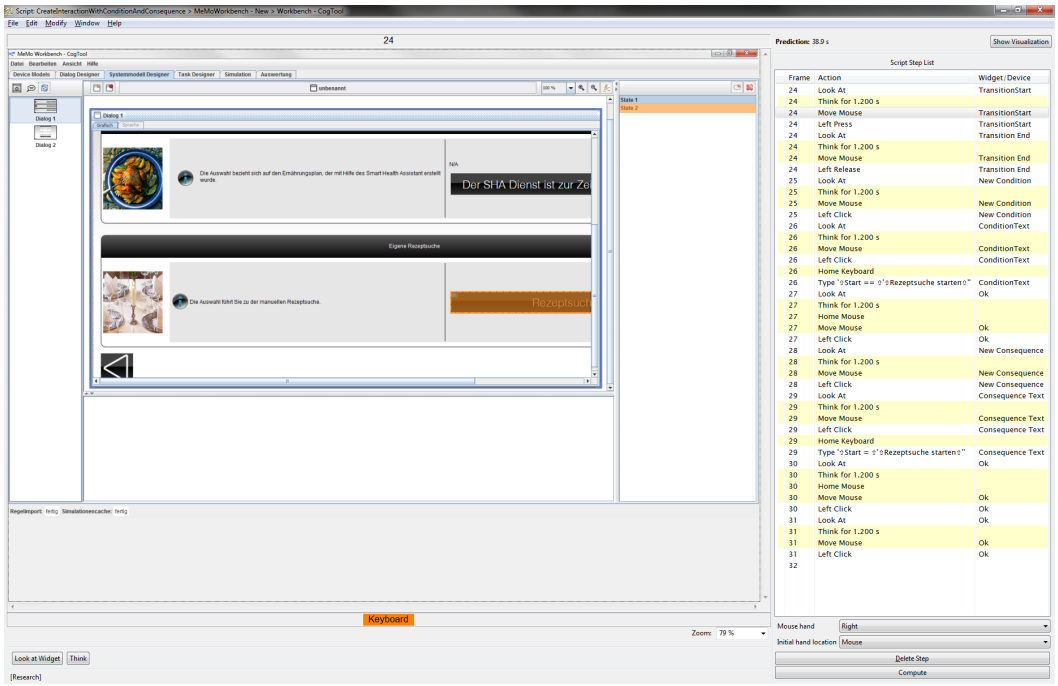


**Figure B.2.:** Actual CogTool model of the MeMo workbench when modeling a transition using conditions and consequences.


**Figure B.3.:** Screenshot of visualization of the CogTool model of the MeMo workbench depicting the time prediction and underlying steps.

**Table B.1.:** Task 1: Complete walkthrough of the cooking assistant.

| # UTK | All Interactions Predefined | Manipulating Interactions Only | Enh. Sem. Processing (Task & AUI Analysis) |
|---|---|---|---|
| **01** | Rezeptsuche starten | Hauptgericht | Deutsches Hauptgericht |
| **02** | Hauptgericht | Deutsch | Egal |
| **03** | Deutsch | Egal | Lammkoteletts |
| **04** | Egal | Lammkoteletts | + (executecount=3) |
| **05** | Suche starten | + (executecount=3) | Knoblauchzehen Nicht Vorhanden |
| **06** | Lammkoteletts | 4 Knoblauchzehen | Knödel Nicht Vorhanden |
| **07** | Mit Auswahl fortfahren | Nicht Vorhanden | Kräuterbutter Habe ich |
| **08** | + (executecount=3) | 4 Knödel | Lammkoteletts Nicht Vorhanden |
| **09** | Zutaten überprüfen | Nicht Vorhanden | Majoran Nicht Vorhanden |
| **10** | 4 Knoblauchzehen | 52g Kräuterbutter | Olivenöl Habe ich |
| **11** | Nicht Vorhanden | Habe ich | Paprika Habe ich |
| **12** | 4 Knödel | 4 Lammkoteletts | Pfeffer Habe ich |
| **13** | Nicht Vorhanden | Nicht Vorhanden | Salz Habe ich |
| **14** | 52g Kräuterbutter | 4 Bund Majoran | Tomaten Habe ich |
| **15** | Habe ich | Nicht Vorhanden | Einkaufsliste erzeugen |
| **16** | 4 Lammkoteletts | 8 tl Olivenöl | Koblauchzehen Abhaken |
| **17** | Nicht Vorhanden | Habe ich | Knödel Abhaken |
| **18** | 4 Bund Majoran | 4 Paprika | Lammkoteletts Abhaken |
| **19** | Nicht Vorhanden | Habe ich | Majoran Abhaken |
| **20** | 8 tl Olivenöl | 4 tl Pfeffer | Fertig (executecount=6) |
| **21** | Habe ich | Habe ich | |
| **22** | 4 Paprika | 4 Prise Salz | |
| **23** | Habe ich | Habe ich | |
| **24** | 4 tl Pfeffer | 4 Tomaten | |
| **25** | Habe ich | Habe ich | |
| **26** | 4 Prise Salz | 4 Knoblauchzehen | |
| **27** | Habe ich | Abhaken | |
| **28** | 4 Tomaten | 4 Knödel | |
| **29** | Habe ich | Abhaken | |
| **30** | Einkaufsliste erzeugen | 4 Lammkoteletts | |
| **31** | 4 Knoblauchzehen | Abhaken | |
| **32** | Abhaken | 4 Bund Majoran | |
| **33** | 4 Knödel | Abhaken | |
| **34** | Abhaken | Fertig (executecount=6) | |
| **35** | 4 Lammkoteletts | | |
| **36** | Abhaken | | |
| **37** | 4 Bund Majoran | | |
| **38** | Abhaken | | |
| **39** | Mit Kochassistent fortfahren | | |
| **40** | Fertig (executecount=6) | | |

**Table B.2.:** Task 2: Choosing a meal, checking the available ingredients and creating a shopping list.

| # UTK | All Interactions Predefined | Manipulating Interactions Only | Enh. Sem. Processing (Task & AUI Analysis) |
|---|---|---|---|
| **01** | Rezeptsuche starten | Italienisch | Italienisch |
| **02** | Italienisch | Französisch | Französisch |
| **03** | Französisch | Nachtisch | Nachtisch |
| **04** | Nachtisch | Panna Cotta | Panna Cotta |
| **05** | Suche starten | + (executecount=2) | + (executecount=2) |
| **06** | Panna Cotta | 3 Blattgelatine | Blattgelatine Habe ich |
| **07** | Mit Auswahl fortfahren | Habe ich | Caramelsirup Habe ich |
| **08** | + (executecount=2) | 75 ml Caramelsirup | Sahne Habe ich |
| **09** | Zutaten überprüfen | Habe ich | Vanilleschote Habe ich |
| **10** | 3 Blattgelatine | 375 g Sahne | Zucker Habe ich |
| **11** | Habe ich | Habe ich | |
| **12** | 75 ml Caramelsirup | 3 Vanilleschote | |
| **13** | Habe ich | Habe ich | |
| **14** | 375 g Sahne | 39 g Zucker | |
| **15** | Habe ich | Habe ich | |
| **16** | 3 Vanilleschote | | |
| **17** | Habe ich | | |
| **18** | 39 g Zucker | | |
| **19** | Habe ich | | |
| **20** | Einkaufsliste erzeugen | | |

**Table B.3.:** Task 3: Choosing a meal, then restart and choose a different meal.

| # UTK | All Interactions Predefined | Manipulating Interactions Only | Enh. Sem. Processing (Task & AUI Analysis) |
|---|---|---|---|
| **01** | Rezeptsuche starten | Hauptgericht | Hauptgericht |
| **02** | Hauptgericht | Neustart | Neustart |
| **03** | Neustart | Diätküche | Diätküche |
| **04** | Rezeptsuche starten | Thunfischcreme | Thunfischcreme |
| **05** | Diätküche | | |
| **06** | Suche starten | | |
| **07** | Thunfischcreme | | |
| **08** | Mit Auswahl fortfahren | | |

# Bibliography

[Aarts et al. 2002] Aarts, E. ; Harwig, R. ; Schuurmans, M.: Ambient intelligence. In: Denning, P. J. (Ed.): *The invisible future*. New York, NY, USA : McGraw-Hill, Inc., 2002. – ISBN 0–07–138224–0, 235–250

[Abowd et al. 1999] Abowd, G. D. ; Dey, A. K. ; Brown, P. J. ; Davies, N. ; Smith, M. ; Steggles, P.: Towards a Better Understanding of Context and Context-Awareness. In: *HUC '99: Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing*. London, UK : Springer-Verlag, 1999. – ISBN 3–540–66550–1, pp. 304–307

[Abowd and Mynatt 2000] Abowd, G. D. ; Mynatt, E. D.: Charting Past, Present, and Future Research in Ubiquitous Computing. In: *ACM Transactions Computer-Human Interaction* 7 (2000), No. 1, pp. 29–58. – ISSN 1073–0516

[Abrahão et al. 2008] Abrahão, S. ; Iborra, E. ; Vanderdonckt, J.: Usability Evaluation of User Interfaces Generated with a Model-Driven Architecture Tool. In: *Maturing Usability*, Springer London, 2008 (Human-Computer Interaction Series). – ISBN 978–1–84628–940–8 (Print) 978–1–84628–941–5 (Online), 3–32

[Amant et al. 2005] Amant, R. S. ; Freed, A. R. ; Ritter, F. E.: Specifying ACT-R Models of User Interaction with a GOMS Language. In: *Cognitive Systems Research* 6 (2005), No. 1, pp. 71 – 88. – ISSN 1389–0417

[Anderson et al. 2004] Anderson, J. R. ; Bothell, D. ; Byrne, M. D. ; Douglass, S. ; Lebiere, C. ; Qin, Y.: An Integrated Theory of the Mind. In: *Psychological Review* 111 (2004), October, No. 4, pp. 1036–1060. – ISSN 0033–295X

[Balme et al. 2004] Balme, L. ; Demeure, A. ; Barralon, N. ; Coutaz, J. ; Calvary, G.: CAMELEON-RT: A Software Architecture Reference Model for Distributed, Migratable, and Plastic User Interfaces. In: Markopoulos, P. (Ed.) ; Eggen, B. (Ed.) ; Aarts, E. (Ed.) ; Crowley, J. (Ed.): *Ambient Intelligence* Vol. 3295. Berlin, Heidelberg : Springer, 2004. – ISBN 978–3–540–23721–1, 291-302

[Bass et al. 1992] Bass, L. ; Faneuf, R. ; Little, R. ; Mayer, N. ; Pellegrino, B. ; Reed, S. ; Seacord, R. ; Sheppard, S. ; Szczur, M. R.: A Metamodel

for the Runtime Architecture of an Interactive System: The UIMS Tool Developers Workshop. In: *SIGCHI Bull.* 24 (1992), No. 1, pp. 32–37. – ISSN 0736–6906

[Bastien and Scapin 1993] Bastien, J. M. C. ; Scapin, D. L.: Ergonomic Criteria for the Evaluation of Human-Computer Interfaces / Inria, Institut National de Recherche en Informatique et en Automatique. 1993 (RT-0156)

[Baumeister et al. 2000] Baumeister, L. K. ; John, B. E. ; Byrne, M. D.: A Comparison of Tools for Building GOMS Models. In: *CHI '00: Proceedings of the SIGCHI conference on Human factors in computing systems.* New York, USA : ACM, 2000. – ISBN 1–58113–216–6, pp. 502–509

[Betz et al. 2010] Betz, T. ; Kietzmann, T. C. ; Wilming, N. ; König, P.: Investigating task-dependent top-down effects on overt visual attention. In: *Journal of Vision* 10 (2010), No. 3

[Bezold 2009] Bezold, M.: Describing User Interactions in Adaptive Interactive Systems. In: Houben, G.-J. (Ed.) ; McCalla, G. (Ed.) ; Pianesi, F. (Ed.) ; Zancanaro, M. (Ed.): *User Modeling, Adaptation, and Personalization* Vol. 5535. Springer Berlin Heidelberg, 2009. – ISBN 978–3–642–02246–3, 150-161

[Biswas et al. 2013] Biswas, P. ; Langdon, P. ; Duarte, C. ; Coelho, J. ; Guerreiro, T. ; Jung, C.: An accessible, adaptive and multimodal digital TV framework and corresponding development tool. In: *Science and Information Conference (SAI), 2013*, 2013, pp. 28–37

[Biswas et al. 2012] Biswas, P. ; Robinson, P. ; Langdon, P.: Designing Inclusive Interfaces Through User Modeling and Simulation. In: *Int. J. Hum. Comput. Interaction* 28 (2012), No. 1, 1–33

[Blackmon et al. 2005] Blackmon, M. H. ; Kitajima, M. ; Polson, P. G.: Tool for Accurately Predicting Website Navigation Problems, Non-Problems, Problem Severity, and Effectiveness of Repairs. In: *CHI '05: Proceedings of the SIGCHI conference on Human factors in computing systems.* New York, USA : ACM, 2005. – ISBN 1–58113–998–5, pp. 31–40

[Blandford et al. 2004] Blandford, A. ; Butterworth, R. ; Curzon, P.: Models of Interactive Systems: A Case Study on Programmable User Modelling. In: *Int. J. Hum.-Comput. Stud.* 60 (2004), No. 2, pp. 149–200. – ISSN 1071–5819

[Blumendorf 2009] Blumendorf, M.: *Multimodal Interaction in Smart Environments A Model-based Runtime System for Ubiquitous User Interfaces*, Technische Universität Berlin, Diss., 2009

[Blumendorf et al. 2008] Blumendorf, M. ; Lehmann, G. ; Feuerstack, S. ; Albayrak, S.: Executable Models for Human-Computer Interaction. In: Graham, T. (Ed.) ; Palanque, P. (Ed.): *Interactive Systems. Design, Specification, and Verification* Vol. 5136. Springer Berlin Heidelberg, 2008. – ISBN 978–3–540–70568–0, 238-251

[Bodart et al. 1995] Bodart, F. ; Hennebert, A.-M. ; Leheureux, J.-M. ; Provot, I. ; Sacré, B. ; Vanderdonckt, J.: Towards a Systematic Building of Software Architecture: The TRIDENT Methodological Guide. In: Palanque, P. (Ed.) ; Bastide, R. (Ed.): *Design, Specification and Verification of Interactive Systems '95*. Wien : Springer-Verlag, 1995, 262–278

[Buchholz and Propp 2008] Buchholz, G. ; Propp, S.: Towards Usability Evaluation for Smart Appliance Ensembles. In: Graham, T. (Ed.) ; Palanque, P. (Ed.): *Interactive Systems. Design, Specification, and Verification* Vol. 5136. Springer Berlin / Heidelberg, 2008, 294-299

[Byrne et al. 1994] Byrne, M. D. ; Wood, D. ; Sukaviriya, P. N. ; Foley, J. D. ; Kieras, D. E.: Automating Interface Evaluation. In: Plaisant, C. (Ed.): *CHI Conference Companion*, ACM, 1994. – ISBN 0–89791–651–4, pp. 216

[Calvary et al. 2003] Calvary, G. ; Coutaz, J. ; Thevenin, D. ; Limbourg, Q. ; Bouillon, L. ; Vanderdonckt, J.: A Unifying Reference Framework for Multi-Target User Interfaces. In: *Interacting with Computers* 15 (2003), No. 3, 289–308. – ISSN 0953–5438

[Calvary et al. 2002] Calvary, G. ; Coutaz, J. ; Thevenin, D. ; Limbourg, Q. ; Souchon, N. ; Bouillon, L. ; Florins, M. ; Vanderdonckt, J.: Plasticity of User Interfaces: A Revised Reference Framework. In: *TAMODIA '02: Proceedings of the First International Workshop on Task Models and Diagrams for User Interface Design*, INFOREC Publishing House Bucharest, 2002. – ISBN 973–8360–01–3, pp. 127–134

[Card et al. 1983] Card, S. K. ; Moran, T. P. ; Newell, A.: *The Psychology of Human-Computer Interaction*. Hillsdale : Lawrence Erlbaum Ass. Publ., 1983. – ISBN 0898592437

[Carmichael et al. 2005] Carmichael, D. J. ; Kay, J. ; Kummerfeld, B.: Consistent Modelling of Users, Devices and Sensors in a Ubiquitous Computing Environment. In: *User Modeling and User-Adapted Interaction* 15 (2005), No. 3-4, pp. 197–234. – ISSN 0924–1868

[Casas et al. 2008] Casas, R. ; Blasco Marín, R. ; Robinet, A. ; Delgado, A. R. ; Yarza, A. R. ; Mcginn, J. ; Picking, R. ; Grout, V.: User Modelling in Ambient Intelligence for Elderly and Disabled People. In: *ICCHP '08: Proceedings of the 11th international conference on Computers Helping People with Special Needs*. Berlin, Heidelberg : Springer-Verlag, 2008. – ISBN 978–3–540–70539–0, pp. 114–122

[Chen and Kotz 2000] Chen, G. ; Kotz, D.: A Survey of Context-Aware Mobile Computing Research / Dept. of Computer Science, Dartmouth College. 2000 (TR2000-381)

[Clerckx et al. 2004] Clerckx, T. ; Luyten, K. ; Coninx, K.: DynaMo-AID: A Design Process and a Runtime Architecture for Dynamic Model-Based User Interface Development. In: *Engineering Human Computer Interaction and Interactive Systems*, 2004, pp. 77–95

[Collignon et al. 2008] Collignon, B. ; Vanderdonckt, J. ; Calvary, G.: An Intelligent Editor for Multi-Presentation User Interfaces. In: *Proc. of 23rd Annual ACM Symposium on Applied Computing SAC 2008*, 2008

[Cooper 1999] Cooper, A.: *The Inmates Are Running the Asylum*. Indianapolis, IN, USA : Macmillan Publishing Co., Inc., 1999. – ISBN 0672316498

[Crowley et al. 2002] Crowley, J. L. ; Coutaz, J. ; Rey, G. ; Reignier, P.: Perceptual Components for Context Aware Computing. In: *UbiComp '02: Proceedings of the 4th international conference on Ubiquitous Computing*. London, UK : Springer-Verlag, 2002. – ISBN 3–540–44267–7, pp. 117–134

[Demeure et al. 2008] Demeure, A. ; Calvary, G. ; Coninx, K.: COMET(s), A Software Architecture Style and an Interactors Toolkit for Plastic User Interfaces. In: *Design, Specification, and Verification*, 2008, pp. 225–237

[DIN EN ISO 9241-110 2008] Norm DIN EN ISO 9241-110 09 2008. *Ergonomics of Human-System Interaction: Part 110 Dialogue Principles*

[Dix et al. 2003] Dix, A. ; Finlay, J. ; Abowd, G. D. ; Beale, R.: *Human-Computer Interaction*. Third. Pearson Prentice Hall, 2003

[Eason 1984] Eason, K. D.: Towards the experimental study of usability. In: *Behaviour and Information Technology, 3*, 1984, pp. 133–143

[Eckert et al. 1997] Eckert, W. ; Levin, E. ; Pieraccini, R.: User Modeling For Spoken Dialogue System Evaluation. In: *Proc. IEEE ASR Workshop*, 1997, pp. 80–87

[Egger et al. 2012] Egger, S. ; Reichl, P. ; Hosfeld, T. ; Schatz, R.: "Time is bandwidth"? Narrowing the gap between subjective time perception and Quality of

Experience. In: *Communications (ICC), 2012 IEEE International Conference on*, 2012. – ISSN 1550–3607, pp. 1325–1330

[Farenc et al. 1995] Farenc, C. ; Palanque, P. ; Vanderdonckt, J.: User Interface Evaluation: Is it Ever Usable? In: Y. Anzai, K. O. (Ed.) ; Mori, H. (Ed.): *Proceedings of 6th International Conference on Human-Computer Interaction HCI International'95* Vol. Vol. 20B. Amsterdam : Elsevier Science B.V, 1995 (Advances in Human Factors/Ergonomics Series), pp. 329–334

[Feuerstack 2008] Feuerstack, S.: *A Method for the User-centered and Model-based Development of Interactive Applications*, Technische Universität Berlin, Diss., 2008

[Feuerstack et al. 2008] Feuerstack, S. ; Blumendorf, M. ; Kern, M. ; Kruppa, M. ; Quade, M. ; Runge, M. ; Albayrak, S.: Automated Usability Evaluation during Model-Based Interactive System Development. In: *HCSE-TAMODIA '08: Proceedings of the 2nd Conference on Human-Centered Software Engineering and 7th International Workshop on Task Models and Diagrams*. Berlin, Heidelberg : Springer-Verlag, 2008. – ISBN 978–3–540–85991–8, pp. 134–141

[Finin and Drager 1986] Finin, T. ; Drager, D.: GUMS: A General User Modeling System. In: *Proceedings of the workshop on Strategic computing natural language of the Human Language Technology Conference*, Association for Computational Linguistics, May 1986, pp. 224–230

[Fischer 2001] Fischer, G.: User Modeling in Human-Computer Interaction. In: *User Modeling and User-Adapted Interaction* 11 (2001), No. 1-2, 65-86. – ISSN 0924–1868

[Fitts 1954] Fitts, P. M.: The information capacity of the human motor system in controlling the amplitude of movement. In: *Journal of Experimental Psychology* 47 (1954), June, No. 6, 381-391

[Fu and Pirolli 2007] Fu, W.-t. ; Pirolli, P.: SNIF-ACT: A Cognitive Model of User Navigation on the World Wide Web. In: *Human-Computer Interaction* 22 (2007), pp. 355–412

[García Frey et al. 2010] García Frey, A. ; Calvary, G. ; Dupuy-Chesa, S.: Xplain: An Editor for Building Self-Explanatory User Interfaces by Model-driven Engineering. In: *Proceedings of the 2nd ACM SIGCHI symposium on Engineering interactive computing systems*. New York, USA : ACM, 2010 (EICS '10). – ISBN 978–1–4503–0083–4, pp. 41–46

[González-Calleros et al. 2013] González-Calleros, J. M. ; Osterloh, J. P. ; Feil, R. ; Lüdtke, A.: Automated UI evaluation based on a cognitive architecture and UsiXML. In: *Science of Computer Programming Journal* In Press (2013), 05

[Gram and Cockton 1997] Gram, C. (Ed.) ; Cockton, G. (Ed.): *Design principles for interactive software*. London, UK : Chapman & Hall, Ltd., 1997. – ISBN 0–412–72470–7

[Halbrügge et al. 2015a] Halbrügge, M. ; Quade, M. ; Engelbrecht, K.-P.: How Can Cognitive Modeling Benefit from Ontologies? Evidence from the HCI Domain. In: Bieger, J. (Ed.) ; Goertzel, B. (Ed.) ; Potapov, A. (Ed.): *Artificial General Intelligence* Vol. 9205. Springer International Publishing, 2015. – ISBN 978–3–319–21364–4, 261-271

[Halbrügge et al. 2015b] Halbrügge, M. ; Quade, M. ; Engelbrecht, K.-P.: A Predictive Model of Human Error based on User Interface Development Models and a Cognitive Architecture. In: Taatgen, N. A. (Ed.) ; Vugt, M. K. (Ed.) ; Borst, J. P. (Ed.) ; Mehlhorn, K. (Ed.): *Proceedings of the 13th International Conference on Cognitive Modeling*, 2015, pp. 238–243

[Halverson and Hornof 2007] Halverson, T. ; Hornof, A. J.: A minimal model for predicting visual search in human-computer interaction. In: *Proceedings of the SIGCHI conference on Human factors in computing systems*. New York, NY, USA : ACM, 2007 (CHI '07). – ISBN 978–1–59593–593–9, 431–434

[Harris et al. 2010] Harris, B. N. ; John, B. E. ; Brezin, J.: Human Performance Modeling for All: Importing UI Prototypes into CogTool. In: *Proceedings of the 28th of the international conference extended abstracts on Human factors in computing systems*. New York, USA : ACM, 2010 (CHI EA '10). – ISBN 978–1–60558–930–5, pp. 3481–3486

[Hassenzahl 2004] Hassenzahl, M.: The interplay of beauty, goodness, and usability in interactive products. In: *Human-Computer Interaction* 19 (2004), December, No. 4, 319–349. – ISSN 0737–0024

[Hassenzahl et al. 2008] Hassenzahl, M. ; Schöbel, M. ; Trautmann, T.: How motivational orientation influences the evaluation and choice of hedonic and pragmatic interactive products: The role of regulatory focus. In: *Interacting with Computers* 20 (2008), September, No. 4-5, 473–479. – ISSN 0953–5438

[Heckmann 2006] *Chapter* Situation Modeling and Smart Context Retrieval with Semantic Web Technology and Conflict Resolution. In: Heckmann, D.: *Lecture Notes in*

*Computer Science*. Vol. 3946/2006: *Modeling and Retrieval of Context*. Springer Berlin / Heidelberg, 2006, pp. 34–47

[Heckmann et al. 2005] Heckmann, D. ; Schwartz, T. ; Brandherm, B. ; Schmitz, M. ; Wilamowitz-Moellendorff, M. von: GUMO - The General User Model Ontology. In: *User Modeling 2005* Vol. 3538, Springer Berlin / Heidelberg, 2005 (Lecture Notes in Computer Science). – ISBN 978–3–540–27885–6, 428–432

[Heinath et al. 2007] Heinath, M. ; Dzaack, J. ; Wiesner, A. ; Urbas, L.: Applications for Cognitive User Modeling. In: *UM '07: Proceedings of the 11th international conference on User Modeling*. Berlin, Heidelberg : Springer-Verlag, 2007. – ISBN 978–3–540–73077–4, pp. 127–136

[Ivory and Hearst 2001] Ivory, M. Y. ; Hearst, M. A.: The State of the Art in Automating Usability Evaluation of User Interfaces. In: *ACM Comput. Surv.* 33 (2001), No. 4, pp. 470–516. – ISSN 0360–0300

[Jameson et al. 2007] Jameson, A. ; Mahr, A. ; Kruppa, M. ; Rieger, A. ; Schleicher, R.: Looking for unexpected consequences of interface design decisions: the MeMo workbench. In: *Proceedings of the 6th international conference on Task models and diagrams for user interface design*. Berlin, Heidelberg : Springer-Verlag, 2007 (TAMODIA'07). – ISBN 3–540–77221–9, 978–3–540–77221–7, 279–286

[John and Suzuki 2009] John, B. ; Suzuki, S.: Toward Cognitive Modeling for Predicting Usability. In: Jacko, J. (Ed.): *Human-Computer Interaction. New Trends* Vol. 5610. Springer Berlin / Heidelberg, 2009, pp. 267–276

[John and Jastrzembski 2010] John, B. E. ; Jastrzembski, T.: Exploration of Costs and Benefits of Predictive Human Performance Modeling for Design. In: Salvucci, D. (Ed.) ; Gunzelmann, G. (Ed.): *Proceedings of the 10th International Conference on Cognitive Modeling*. Philadelphia, PA, 2010, pp. 115–120

[John and Kieras 1996] John, B. E. ; Kieras, D. E.: The GOMS Family of User Interface Analysis Techniques: Comparison and Contrast. In: *ACM Trans. Comput.-Hum. Interact.* 3 (1996), No. 4, pp. 320–351. – ISSN 1073–0516

[John et al. 2004] John, B. E. ; Prevas, K. ; Salvucci, D. D. ; Koedinger, K.: Predictive Human Performance Modeling Made Easy. In: *CHI '04: Proceedings of the SIGCHI conference on Human factors in computing systems*. New York, USA : ACM Press, 2004. – ISBN 1–58113–702–8, pp. 455–462

[John and Salvucci 2005] John, B. E. ; Salvucci, D. D.: Multipurpose Prototypes for Assessing User Interfaces in Pervasive Computing Systems. In: *IEEE Pervasive Computing* 4 (2005), No. 4, pp. 27–34. – ISSN 1536–1268

[Kay et al. 2002] Kay, J. ; Kummerfeld, B. ; Lauder, P.: Personis: A Server for User Models. In: Bra, P. (Ed.) ; Brusilovsky, P. (Ed.) ; Conejo, R. (Ed.): *Adaptive Hypermedia and Adaptive Web-Based Systems* Vol. 2347. Springer Berlin Heidelberg, 2002. – ISBN 978–3–540–43737–6, 203-212

[Keates et al. 2000] Keates, S. ; Clarkson, J. ; Robinson, P.: Investigating the applicability of user models for motion-impaired users. In: *Proceedings of the fourth international ACM conference on Assistive technologies*. New York, NY, USA : ACM, 2000 (Assets '00). – ISBN 1–58113–313–8, 129–136

[Kieras 2003] Kieras, D.: Model-based Evaluation. In: Jacko, J. A. (Ed.) ; Sears, A. (Ed.): *The Human-computer Interaction Handbook*. Hillsdale, NJ, USA : L. Erlbaum Associates Inc., 2003. – ISBN 0–8058–3838–4, 1139–1151

[Kieras 2006] Kieras, D.: *A Guide to GOMS Model Usability Evaluation using GOMSL and GLEAN4. Unpublished manuscript.* `ftp://ftp.eecs.umich.edu/people/kieras/GOMS/GOMSL_Guide.pdf`, 31. March 2006

[Kieras et al. 2001] Kieras, D. ; Meyer, D. ; Ballas, J.: Towards Demystification of Direct Manipulation: Cognitive Modeling Charts the Gulf of Execution. In: *CHI '01: Proceedings of the SIGCHI conference on Human factors in computing systems*. New York, USA : ACM, 2001. – ISBN 1–58113–327–8, pp. 128–135

[Kieras and Meyer 1997] Kieras, D. E. ; Meyer, D. E.: An Overview of the EPIC Architecture for Cognition and Performance with Application to Human-Computer interaction. In: *Hum.-Comput. Interact.* 12 (1997), No. 4, pp. 391–438. – ISSN 0737–0024

[Kieras and Santoro 2004] Kieras, D. E. ; Santoro, T. P.: Computational GOMS modeling of a complex team task: lessons learned. In: *CHI '04: Proceedings of the SIGCHI conference on Human factors in computing systems*. New York, USA : ACM, 2004. – ISBN 1–58113–702–8, pp. 97–104

[Kieras et al. 1995] Kieras, D. E. ; Wood, S. D. ; Abotel, K. ; Hornof, A.: GLEAN: a computer-based tool for rapid GOMS model usability evaluation of user interface designs. In: *UIST '95: Proceedings of the 8th annual ACM symposium on User interface and software technology*. New York, NY, USA : ACM, 1995. – ISBN 0–89791–709–X, pp. 91–100

[Kirisci et al. 2011] Kirisci, P. ; Thoben, K.-D. ; Klein, P. ; Modzelewski, M.: Supporting inclusive product design with virtual user models at the early stages of product development. In: *Proceedings of the 18th International Conference on Engineering Design (ICED11), Vol. 9*, 2011, pp. 80–90

[Kobsa 2001] Kobsa, A.: Generic User Modeling Systems. In: *User Modeling and User-Adapted Interaction* 11 (2001), No. 1-2, pp. 49–63. – ISSN 0924–1868

[Kolb 2008] Kolb, P.: DISCO: A Multilingual Database of Distributionally Similar Words. In: Storrer, A. (Ed.) ; Geyken, A. (Ed.) ; Siebert, A. (Ed.) ; Würzner, K.-M. (Ed.): *KONVENS 2008 – Ergänzungsband: Textressourcen und lexikalisches Wissen*, 2008, 37–44

[Larsen et al. 2008] Larsen, K. ; Nevo, D. ; Rich, E.: Exploring the Semantic Validity of Questionnaire Scales. In: *Hawaii International Conference on System Sciences, Proceedings of the 41st Annual*, 2008. – ISSN 1530–1605, pp. 440

[Lehmann et al. 2011] Lehmann, G. ; Blumendorf, M. ; Trollmann, F. ; Albayrak, S.: Meta-modeling Runtime Models. In: Dingel, J. (Ed.) ; Solberg, A. (Ed.): *Models in Software Engineering* Vol. 6627. Springer Berlin / Heidelberg, 2011. – ISBN 978–3–642–21209–3, 209-223

[Limbourg et al. 2005] Limbourg, Q. ; Vanderdonckt, J. ; Michotte, B. ; Bouillon, L. ; López-Jaquero, V.: USIXML: A Language Supporting Multi-path Development of User Interfaces. In: Bastide, R. (Ed.) ; Palanque, P. (Ed.) ; Roth, J. (Ed.): *Engineering Human Computer Interaction and Interactive Systems* Vol. 3425. Berlin, Heidelberg : Springer-Verlag, 2005. – ISBN 978–3–540–26097–4, 200-220

[Matiouk et al. 2013] Matiouk, S. ; Modzelewski, M. ; Mohamad, Y. ; Lawo, M. ; Kirisci, P. ; Klein, P. ; Fennell, A.: Prototype of a Virtual User Modeling Software Framework for Inclusive Design of Consumer Products and User Interfaces. In: Stephanidis, C. (Ed.) ; Antona, M. (Ed.): *Universal Access in Human-Computer Interaction. Design Methods, Tools, and Interaction Techniques for eInclusion* Vol. 8009. Springer Berlin Heidelberg, 2013. – ISBN 978–3–642–39187–3, 59-66

[Michotte and Vanderdonckt 2008] Michotte, B. ; Vanderdonckt, J.: GrafiXML, A Multi-Target User Interface Builder based on UsiXML. In: *Proc. of 4th International Conference on Autonomic and Autonomous Systems ICAS 2008*, IEEE Computer Society Press, 16-21 March 2008

[Modzelewski et al. 2012] Modzelewski, M. ; Lawo, M. ; Kirisci, P. ; Connor, J. ; Fennell, A. ; Mohamad, Y. ; Matiouk, S. ; Valle-Klann, M. ; Gökmen,

H.: Creative Design for Inclusion Using Virtual User Models. In: Miesenberger, K. (Ed.) ; Karshmer, A. (Ed.) ; Penaz, P. (Ed.) ; Zagler, W. (Ed.): *Computers Helping People with Special Needs* Vol. 7382. Springer Berlin Heidelberg, 2012. – ISBN 978–3–642–31521–3, 288-294

[Mori et al. 2004] Mori, G. ; Paternò, F. ; Santoro, C.: Design and Development of Multidevice User Interfaces through Multiple Logical Descriptions. In: *IEEE Trans. Softw. Eng.* 30 (2004), No. 8, pp. 507–520

[Morin et al. 2009] Morin, B. ; Barais, O. ; Jezequel, J.-M. ; Fleurey, F. ; Solberg, A.: Models@ Run.time to Support Dynamic Adaptation. In: *Computer* 42 (2009), October, No. 10, 44–51. – ISSN 0018–9162

[Newell 1990] Newell, A.: *Unified Theories of Cognition.* Cambridge, MA, USA : Harvard University Press, 1990. – ISBN 0–674–92099–6

[Nielsen et al. 2005] Nielsen, C. M. ; Overgaard, M. ; Pedersen, M. B. ; Stage, J.: Feedback from Usability Evaluation to User Interface Design: Are Usability Reports Any Good? In: *Human-Computer Interaction - INTERACT 2005* Vol. 3585, Springer Berlin / Heidelberg, 2005 (Lecture Notes in Computer Science). – ISBN 978–3–540–28943–2, pp. 391–404

[Nielsen 1992] Nielsen, J.: Finding Usability Problems Through Heuristic Evaluation. In: *CHI '92: Proceedings of the SIGCHI conference on Human factors in computing systems.* New York, USA : ACM Press, 1992. – ISBN 0–89791–513–5, pp. 373–380

[Nielsen 1993a] Nielsen, J.: Noncommand User Interfaces. In: *Commun. ACM* 36 (1993), No. 4, pp. 83–99. – ISSN 0001–0782

[Nielsen 1993b] Nielsen, J.: *Usability Engineering.* San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 1993. – ISBN 0125184050

[Nielsen and Molich 1990] Nielsen, J. ; Molich, R.: Heuristic Evaluation of User Interfaces. In: *CHI '90: Proceedings of the SIGCHI conference on Human factors in computing systems.* New York, USA : ACM Press, 1990. – ISBN 0–201–50932–6, pp. 249–256

[Norman 1983] *Chapter* Some Observations on Mental Models. In: Norman, D. A.: *Mental Models.* Hillsdale, NJ, USA : Erlbaum, 1983, pp. 7–14

[Norman 1988] Norman, D. A.: *The Psychology Of Everyday Things.* Basic Books, 1988. – ISBN 0465067093

[Palanque et al. 2011] Palanque, P. ; Barboni, E. ; Martinie, C. ; Navarre, D. ; Winckler, M.: A Model-based Approach for Supporting Engineering Usability Evaluation of Interaction Techniques. In: *Proceedings of the 3rd ACM SIGCHI symposium on Engineering interactive computing systems*. New York, USA : ACM, 2011 (EICS '11). – ISBN 978–1–4503–0670–6, pp. 21–30

[Paternò 1999] Paternò, F.: *Model-Based Design and Evaluation of Interactive Applications*. Springer-Verlag, 1999 (Applied Computing). – 208 P. – ISBN 1–85233–155–0

[Paternò 2005] Paternò, F.: Model-based Tools for Pervasive Usability. In: *Interacting with Computers* 17 (2005), No. 3, 291–315

[Peissner and Edlin-White 2013] Peissner, M. ; Edlin-White, R.: User Control in Adaptive User Interfaces for Accessibility. In: Kotzé, P. (Ed.) ; Marsden, G. (Ed.) ; Lindgaard, G. (Ed.) ; Wesson, J. (Ed.) ; Winckler, M. (Ed.): *Human-Computer Interaction - INTERACT 2013* Vol. 8117. Springer Berlin Heidelberg, 2013. – ISBN 978–3–642–40482–5, 623-640

[Peissner et al. 2012] Peissner, M. ; Häbe, D. ; Janssen, D. ; Sellner, T.: MyUI: Generating Accessible User Interfaces from Multimodal Design Patterns. In: *Proceedings of the 4th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*. New York, NY, USA : ACM, 2012 (EICS '12). – ISBN 978–1–4503–1168–7, 81–90

[Polson et al. 1992] Polson, P. G. ; Lewis, C. ; Rieman, J. ; Wharton, C.: Cognitive walkthroughs: a method for theory-based evaluation of user interfaces. In: *International Journal of Man-Machine Studies* 36 (1992), No. 5, 741 - 773. – ISSN 0020–7373

[Quade 2012] Quade, M.: Model-Based Evaluation of Adaptive User Interfaces. In: Wichert, R. (Ed.) ; Van Laerhoven, K. (Ed.) ; Gelissen, J. (Ed.): *Constructing Ambient Intelligence* Vol. 277. Springer Berlin Heidelberg, 2012. – ISBN 978–3–642–31478–0, 318-322

[Quade et al. 2011] Quade, M. ; Blumendorf, M. ; Lehmann, G. ; Roscher, D. ; Albayrak, S.: Evaluating User Interface Adaptations at Runtime by Simulating User Interaction. In: *25th BCS Conference on Human Computer Interaction - HCI2011*, 2011

[Quade et al. 2014] Quade, M. ; Halbrügge, M. ; Engelbrecht, K.-P. ; Albayrak, S. ; Möller, S.: Predicting Task Execution Times by Deriving Enhanced Cognitive Models from User Interface Development Models. In: *Proceedings of the 2014 ACM SIGCHI Symposium on Engineering Interactive Computing Systems*. New York, NY, USA : ACM, 2014 (EICS '14). – ISBN 978–1–4503–2725–1, 139–148

[Rich 1979] Rich, E.: User modeling via stereotypes. In: *Cognitive Science* 3 (1979), No. 4, 329–354

[Rieman et al. 1996] Rieman, J. ; Young, R. M. ; Howes, A.: A Dual-Space Model of Iteratively Deepening Exploratory Learning. In: *International Journal of Human-Computer Studies* 44 (1996), No. 6, pp. 743–775. – ISSN 1071–5819

[Ritter et al. 2006] Ritter, F. E. ; Haynes, S. ; Cohen, M. ; Howes, A. ; John, B. ; Best, B. ; Lebiere, C. ; Jones, R. ; Crossman, J. ; Lewis, R. ; Amant, R. S. ; McBride, S. ; Urbas, L. ; Leuchter, S. ; Vera., A.: High-level behavior representation languages revisited. In: *ICCM 2006 Seventh International Conference on Cognitive Modeling*, 2006, pp. 404–407

[Rossi et al. 2005] Rossi, G. ; Gordillo, S. ; Lyardet, F.: Design Patterns for Context-Aware Adaptation. In: *SAINT-W '05: Proceedings of the 2005 Symposium on Applications and the Internet Workshops (SAINT 2005 Workshops)*. Washington, DC, USA : IEEE Computer Society, 2005. – ISBN 0–7695–2263–7, pp. 170–173

[Ruß 2011] Russ, A.: Modeling Visual Attention for Rule-Based Usability Simulations of Elderly Citizen. In: *Engineering Psychology and Cognitive Ergonomics - 9th International Conference, EPCE 2011, Held as Part of HCI International 2011, Orlando, FL, USA, July 9-14, 2011. Proceedings*, 2011, 72–81

[Ruß et al. 2012] Russ, A. ; Quade, M. ; Kruppa, M. ; Runge, M.: Rule-Based Approach for Simulating Age-Related Usability Problems. In: Wichert, R. (Ed.) ; Eberhardt, B. (Ed.) ; VDE (Org.): *Ambient Assisted Living* Vol. 5. AAL-Kongress 2012. Berlin, Germany : Springer, January 25-25 2012 (Advanced Technologies and Societal Change). – ISBN 978–3–642–27490–9, pp. 149–166

[Salvucci 2009] Salvucci, D. D.: Rapid Prototyping and Evaluation of In-vehicle Interfaces. In: *ACM Trans. Comput.-Hum. Interact.* 16 (2009), June, pp. 9:1–9:33. – ISSN 1073–0516

[Salvucci and Lee 2003] Salvucci, D. D. ; Lee, F. J.: Simple Cognitive Modeling in a Complex Cognitive Architecture. In: *CHI '03: Proceedings of the SIGCHI conference on Human factors in computing systems*. New York, USA : ACM Press, 2003. – ISBN 1–58113–630–7, pp. 265–272

[Sanchez et al. 2008] Sanchez, M. ; Barrero, I. ; Villalobos, J. ; Deridder, D.: An execution platform for extensible runtime models. In: *Technical Report COMP COMP-005-2008 Lancaster University* (2008), pp. 107

[Schilit et al. 1994] Schilit, B. ; Adams, N. ; Want, R.: Context-Aware Computing Applications. In: *Proceedings of the 1994 First Workshop on Mobile Computing Systems and Applications*. Washington, DC, USA : IEEE Computer Society, 1994 (WMCSA '94). – ISBN 978–0–7695–3451–0, 85–90

[Schmidt 2000] Schmidt, A.: Implicit human computer interaction through context. In: *Personal Technologies* 4 (2000), No. 2-3, 191-199. – ISSN 0949–2054

[Searle 1969] Searle, J.: *Speech Acts*. Cambridge : Cambridge University Press, 1969. – ISBN 0–521–09626–X

[Sears 1995] Sears, A.: AIDE: a step toward metric-based interface development tools. In: *UIST '95: Proceedings of the 8th annual ACM symposium on User interface and software technology*. New York, NY, USA : ACM Press, 1995. – ISBN 0–89791–709–X, pp. 101–110

[Shneiderman and Plaisant 2004] Shneiderman, B. ; Plaisant, C.: *Designing the User Interface: Strategies for Effective Human-Computer Interaction (4th Edition)*. Pearson Addison Wesley, 2004. – ISBN 0321197860

[Sottet et al. 2008] Sottet, J.-S. ; Calvary, G. ; Coutaz, J. ; Favre, J.-M.: A Model-Driven Engineering Approach for the Usability of Plastic User Interfaces. In: Gulliksen, J. (Ed.) ; Harning, M. (Ed.) ; Palanque, P. (Ed.) ; Veer, G. van d. (Ed.) ; Wesson, J. (Ed.): *Engineering Interactive Systems* Vol. 4940. Springer Berlin Heidelberg, 2008. – ISBN 978–3–540–92697–9, 140-157

[Sottet et al. 2007] Sottet, J.-S. ; Ganneau, V. ; Calvary, G. ; Coutaz, J. ; Demeure, A. ; Favre, J.-M. ; Demumieux, R.: Model-Driven Adaptation for Plastic User Interfaces. In: Baranauskas, M. C. C. (Ed.) ; Palanque, P. A. (Ed.) ; Abascal, J. (Ed.) ; Barbosa, S. D. J. (Ed.): *Human-Computer Interaction - INTERACT 2007, 11th IFIP TC 13 International Conference, Rio de Janeiro, Brazil, September 10-14, 2007, Proceedings, Part I* Vol. 4662, Springer, 2007 (Lecture Notes in Computer Science). – ISBN 978–3–540–74794–9, 397-410

[Steinnökel et al. 2011] Steinnökel, P. ; Scheel, C. ; Quade, M. ; Albayrak, S.: Towards An Enhanced Semantic Approach For Automatic Usability Evaluation. In: *Proceedings of the Computational Linguistics-Applications Conference*, 2011. – ISBN N 978–83–60810–47–7, pp. 85–91

[Sukaviriya et al. 1993] Sukaviriya, P. ; Foley, J. D. ; Griffith, T.: A second generation user interface design environment: the model and the runtime architecture. In: *CHI '93: Proceedings of the SIGCHI conference on Human factors in computing systems*. New York, NY, USA : ACM Press, 1993. – ISBN 0–89791–575–5, pp. 375–382

[Swearngin et al. 2012] Swearngin, A. ; Cohen, M. ; John, B. ; Bellamy, R.: Easing the Generation of Predictive Human Performance Models from Legacy Systems. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. New York, NY, USA : ACM, 2012 (CHI '12). – ISBN 978–1–4503–1015–4, 2489–2498

[Teo and John 2011] Teo, L. ; John, B. E.: The Evolution of a Goal-Directed Exploration Model: Effects of Information Scent and GoBack Utility on Successful Exploration. In: *Topics in Cognitive Science* 3 (2011), No. 1, pp. 154–165

[Totterdell and Boyle 1990] *Chapter* The Evaluation of Adaptive Systems. In: Totterdell, P. ; Boyle, E.: *Adaptive User Interfaces*. Academic Press, 1990, pp. 161–194

[Trewin and Pain 1999] Trewin, S. ; Pain, H.: Keyboard and mouse errors due to motor disabilities. In: *Int. J. Hum.-Comput. Stud.* 50 (1999), February, No. 2, 109–144. – ISSN 1071–5819

[Underwood 2009] Underwood, G.: Cognitive Processes in Eye Guidance: Algorithms for Attention in Image Processing. In: *Cognitive Computation* 1 (2009), 64-76. – ISSN 1866–9956

[Vanderdonckt 1994] Vanderdonckt, J. ; Bodard, F. (Ed.): *Guide ergonomique des interfaces homme-machine*. Namur : Presses Universitaires, 1994 (Collection "Travaux de l'Institut d'Informatique" 13)

[Vanderdonckt 2008] Vanderdonckt, J.: Model-Driven Engineering of User Interfaces: Promises, Successes, Failures, and Challenges. In: *Proceedings of ROCHI 08*, 2008

[Vanderdonckt et al. 2009] Vanderdonckt, J. ; Guerrero-Garcia, J. ; González-Calleros, J. M.: A Model-Based Approach for Developing Vectorial User Interfaces. In: *Inproceedings of the LA-WEB 2009*, 2009

[Vidal et al. 2005] Vidal, E. ; Thollard, F. ; Higuera, C. de l. ; Casacuberta, F. ; Carrasco, R. C.: Probabilistic Finite-State Machines-Part I. In: *IEEE Trans. Pattern Anal. Mach. Intell.* 27 (2005), July, No. 7, 1013–1025. – ISSN 0162–8828

[Vildjiounaite et al. 2007] Vildjiounaite, E. ; Kocsis, O. ; Kyllönen, V. ; Kladis, B.: Context-Dependent User Modelling for Smart Homes. In: *UM '07: Proceedings of the 11th international conference on User Modeling*. Berlin, Heidelberg : Springer-Verlag, 2007. – ISBN 978–3–540–73077–4, pp. 345–349

[Weiser 1991] Weiser, M.: The Computer for the 21st Century. In: *Scientific American* 265 (1991), September, No. 3, pp. 66–75. ISBN 1–55860–246–1

[Whiteside et al. 1988] *Chapter* 36. In: Whiteside, J. L. ; Bennett, J. ; Holtzblatt, K.: *Usability Engineering: Our Experience and Evolution*. Amsterdam : Elsevier Science Publishers, B. V., 1988, pp. 791–817

[Wittenburg et al. 2006] Wittenburg, P. ; Brugman, H. ; Russel, A. ; Klassmann, A. ; Sloetjes, H.: ELAN: a professional framework for multimodality research. In: *Proceedings of the 5th International Conference on Language Resources and Evaluation (LREC 2006)* Vol. 2006, 2006, pp. 1556–1559

[Wood and Kieras 2002] Wood, S. D. ; Kieras, D. E.: Modeling Human Error For Experimentation, Training, And Error-Tolerant Design. In: *Proceedings of the Interservice/Industry Training, Simulation and Education*, 2002, pp. 1075–1085

[Young et al. 1989] Young, R. M. ; Green, T. R. G. ; Simon, T.: Programmable User Models for Predictive Evaluation of Interface Designs. In: *CHI '89: Proceedings of the SIGCHI conference on Human factors in computing systems*. New York, USA : ACM, 1989. – ISBN 0–89791–301–9, pp. 15–19