Chapter 7

An Introduction to Model-driven Development of User Interfaces to realize Multimodal and Context-sensitive Applications for Smart Environments

Sebastian Feuerstack

Abstract

Computer systems as we know them are currently changing from single systems running a set of applications to complex networks of connected devices heavily influencing the users' everyday lifes. This leads to new requirements for applications and human-computer interaction. Networks of heterogeneous devices in homes and offices require flexible applications that can adapt to changing environments supporting the user in his daily life - anywhere, on any device and at anytime. This requires interactive systems to be able to switch modalities (between tangible, speech, and multi-touch interaction modes) and devices on demand of the user and to enable user interface distribution combining several devices to control one interactive system.

The tutorial gives an introduction to the model-driven development of user interfaces (MDDUI). The tutorial consists of two parts: First, beneath introducing the basic terms and definitions, multi-targeting processes and tools for the user-centered design of interfaces for different context of use are presented. By a discussion about the pros and cons of multi-targeting the second part about the actual switch from design-time to run-time-based approaches in MDDUI is introduced and two exemplary approaches are presented. Finally, current challenges in MDDUI are discussed to motivate the participation in this promising research field.

7.1. Introduction

The change from single computing systems towards smart environments connecting complex networks of devices leads to new requirements for applications and human-computer interaction. A smart environment extends the potential interaction options for

the user from a small amount of isolated utilized devices to combinations of devices that enable addressing a mixture of modalities, like speech, gesture or graphical-driven interaction.

Together with various interconnected sensors that make the environment aware of the user, interactions can be embedded in the environment: Not a specific device, but the whole environment can be used as the interface. This enables a user to seamlessly interact with a computer by choosing and combining a heterogeneous set of interaction devices and modalities based on his preferences.

Addressing various different devices and modalities thus requires the possibility to adapt the user interface to support the specific features of the device's software platform and the provided interaction capabilities. Model-driven user interface development (MDDUI), practiced for a long time in Human-Computer Interaction seem to be a promising approach to support software developers, developing interactive applications. It offers a declarative way of modelling such multi-platform user-interfaces and aims at modelling the different aspects of the user interface on certain levels of abstraction.

Abstraction is a very natural way that people usually like to do to manage too complex entities. To manage complex problems people usually start to discover the main aspects that should be taken into account as well as their relations. An example for identifying models in our daily practice is for instance our every day planning. After waking up we typically start the new day by thinking about the main activities that we would like to perform [53].

With the help of a software engineering process that changes the focus from (manual) implementation to the tool-driven design of models that can be directly executed by a run-time-system, software developers can be supported to address these challenges. This discipline of Engineering for Human-Computer Interaction (EHCI) is a crossroad of two disciplines: Human Computer Interaction (HCI) and Software Engineering (SE).

This article is published to accompany a tutorial held at the Webmedia 2010 conference to introduce the concepts of MDDUI to a broader audience. Therefore I start by reviewing some previous work in this area to introduce the basic terms and concepts of MDDUI. Since I cannot cite all relevant work for the sake of brevity I focus on presenting the initial efforts of MDDUI to generate multiple user interfaces for different devices to reflect the advantages and disadvantages of MDDUI in general and in relation to be applicable for generating and running context-sensitive and multimodal interfaces in smart environments. This discussion is used to motivate the recent shift of interest in MDDUI run-time systems that awake the declarative design models alive to support interface manipulation and a more flexible reaction on context-changes at run-time that I devote special attention to. I refer to my own research results as well as of other research groups to explain the new possibilities multimodal and context-sensitive interaction in smart environments that can be tackled by MDDUI run-time systems. Finally I end up with discussing three basic research challenges that need to be addressed in the next years to enable a seamless interaction in smart environments.

7.2. Model-driven User Interface Development (MDDUI)

The development of interactive systems is a complex problem that is continuously growing with the evolving technology enabling to consider the context of use during interaction or multi-modal access to an application. Model-based approaches, practiced since the late 1980's in Human-Computer Interaction seem to be a promising approach to support software developers, developing interactive applications.

MDDUI is driven by two basic ideas: First, by shifting the perspective in user interface construction from a developer to a user-centric one and second, by reducing the development effort for the creation of user interface that quickly gets cumbersome if an interface should run ad on more than one context (for instance on different devices).

User interface builders that enable a developer to create interfaces and to organize widgets in several dialog boxes are the general approach for a developer-centric approach. The organization of the user interface layouts mainly relies on the designer's experience or on external and therefore loosely coupled design documentation. Different to this, user-centered design environments focus to answer how user interface elements in a certain dialog box can be used to accomplish a particular user task [56].

A design process that is concerned with supporting the creation of interfaces for several contexts has been defined as muti-targeting. Multi-targeting describes a coordinated process of building user interfaces for a set of given context of use. A context of use is defined as a triple (U, P, E) where U represents any user stereotype, P, any computing platform, and E, the physical environment in which the user is carrying out her task with the designated platform [44].

The basic concept of addressing both, the user-centric development perspective and multi-targeting is to model the different aspects of the user interface on certain levels of abstraction. Each level of abstraction is described by a model that offers a declarative way of designing an interface. Therefore research in MDDUI is in general concerned with two basic efforts:

1. The identification of suitable sets of model abstractions and their relations for analyzing, designing and evaluating interactive systems
2. The specification of software engineering processes that change the focus from (manual) implementation to the tool-driven design of models

In the last 20 years of research various model-based approaches have been proposed. An overview paper has been written by [67]. In 2003 the Cameleon Reference Framework has been proposed [13] that serves as a reference for classifying MDDUI approaches that support multi-targeting or multiple contexts of use and has been applied to several MDDUI approaches at this time.

7.2.1 The Cameleon Unifying Reference Framework

The Cameleon Reference Framework [13] defines several design steps for the development of multi-context interactive applications. The basic development process is divided into four steps and is illustrated for two contexts of use in figure 1.

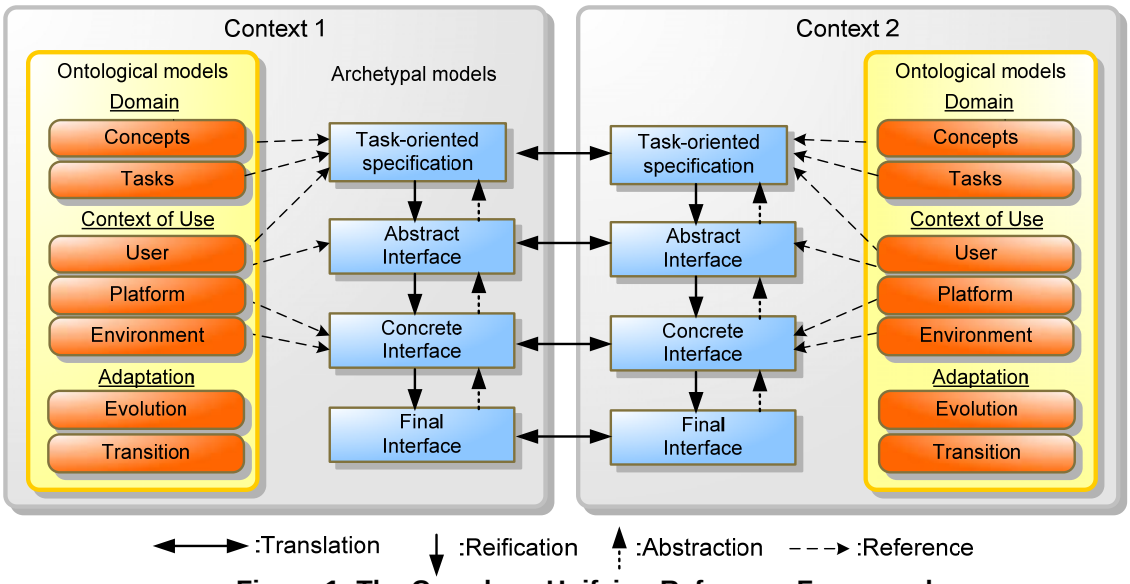

Figure 1: The Cameleon Unifying Reference Framework

7.2.2 Common declarative models of MDDUI approaches

The Framework describes every development step as a manipulation of the artifacts of interests in form of a model or a user interface representation. In distinguishes between ontological and archetypal models. The former one are meta-models and independent of any domain and interactive system, the latter are instantiated ontological models and are dependent on an interactive system for a given domain. Like illustrated in figure 1 there are three basic ontological models: Domain, Context, and Adaptation.

The *domain model* describes concepts and user tasks relative to a domain. A concept might be for instance an UML class diagram, an entity-relationship, or an ontology. The *task model* might be a model that breaks down a user's goals into several activities that need to be either performed by him or by an interactive system and organizes each goal through tasks into logical activities. An instantiated task model might describe how a concrete goal – for instance: "organize meeting" has to be performed by several subsequent tasks in a certain order. Each task is performed either by an interactive system or a user that might be described by a role specification. Tasks are associated to objects instantiated from a concept model and get manipulated during a task is performing.

The ontological *context of use models* enable reasoning about the *users, platforms, and environments* and define the targets of an MDDUI process. The user is relevant for the task specification (e.g. by a role specification) as described earlier. The platform model describes the targeted hardware (device capabilities like screen resolution) and software platform (like e.g. the operating system). Further on, elementary platforms (interaction devices) can be composed to clusters, to be addressed as a single target. Part of the software platform is the *interactor model* that describes the interactors available for the presentation of the user interface. The archetypal realization of an interactor model is for instance a toolkit (e.g. Java Swing or HTML), which contains several widgets (a list box, a button, a voice menu, or a set of navigation gestures). The description of an interactor includes it's look-and-feel, it's input and output capabilities like e.g. required space, the required modes (mouse, keyboard, speech), and side-effects to the context of

use (e.g. additional noise level in case of a speech output). Beneath information about the user, the platform and the environment of the original context of use model definition by [13] the ongoing time is another relevant context-information. It influences the relevance of information that is transmitted by an interface to the user and can be used to adapt the interface by removing information that got irrelevant or highlighting highly relevant parts for a particular time span.

Finally the environment model considers the physical surroundings of the user and the platforms available. This includes for instance the distances between platforms and the users or environmental conditions like noise or lighting level.

The Cameleon Framework further defines two ontological adaptation models. The first one, the evolution model defines triggers based on context of use changes and appropriate reactions that specify a configuration of the interactive system to switch to. The second, the transition model ensures by prologues and epilogues for each context-transition how to handle discontinuities that might occur at a context-change.

An interactive application design as it is described by the Cameleon Framework is based on *initial models* and *transient models*. Initial models a are based on manually entered design information whereas transient models are the result of a model-to-model transformation. There are four *design models* that are often used in MDDUI:

- A Tasks and Concepts model that combines tasks that have to be carried out by the final interactive system with a description of the domain-oriented concepts that are needed by these tasks. Concepts are represented as classes that get instantiated to objects used to reflect the concept manipulation.
- An Abstract User Interface (AUI) is used to describe the logical structure without targeting it to a specific modality or even a platform. An AUI refers to interaction objects that are described in terms of abstract interactors. The AUI groups the subtasks of the task model into interaction spaces (presentation units) according to a set of criteria like task model structural patterns, cognitive work load analysis or identification of the semantic relationships between tasks. Further on the AUI defines a navigation scheme between the interaction spaces and selects Abstract Interaction Objects (AIO) for each of the concepts that have to be independent of any mode of interaction.
- The Concrete User Interface (CUI) that replaces each abstract interactor with a concrete interaction object that is dependent of a modality and a set of associated platforms that implement the same modality and are similar regarding their capabilities (for instance the screen size, and the control features). The Concrete User Interface model (CUI) concretizes the AUI and defines the widget layout and interface navigation through these widgets. Different to the AUI, the CUI is designed for a concrete modality but independent of a specific platform. The CUI is used to present the final look and feel of the user interface but has to be considered as a mockup that only runs within a particular environment.
- On the lowest level of abstraction the Final User Interface (FUI) represents the operational user interface that can be run on a platform either by interpretation

through a renderer (for instance a web-browser) or by execution (after compilation into binary code).

7.2.3 Multi-path development processes

In the last section I have introduced a lot of design models that are relevant in MDDUI to realize an interactive application. But what methods exist to generate or manually design these models? There have been proposed several techniques that can be used that can be used selective – based on requirements ore preconditions of a project or be can be combined and applied in one process: Transformations, Translations, Graceful Degration, and Factorization. Further on different Entry Points mark models that can be used as an initial model to start the process. Finally by referring to examples the progress in the MDDUI process can be made more transparent and feedback can be considered earlier.

Reification, Abstraction, and Translation

There are three different types of model-to-model transformations that are applied to generate or manipulate a design model based on information of another source model: Reification, Abstraction and Translation.

Reification is done by performing forward engineering. Thus, like depicted in figure 1 the designer starts with an abstract design model and transforms it - after it has been specified completely – to a more concrete design model. Concrete models in MDDUI are more shaped to a particular platform and therefore contain more specific details as its abstract successors.

Contrary to reification is *abstraction*, which is in terms of software engineering usually called reverse engineering. Abstraction is a process of generalization of concrete artifacts. It is often an expensive one in terms of effort and complexity. This is because usually the abstraction process starts with an existing system and with the goal to obtain a conceptual description of its design for analysis and evaluation purposes. There are tools available to support model abstraction like ReversiXML [10] and WebRevenge [38] that support partial automation for the abstraction of applications for the web, but with the increasingly usage of dynamic pages driven by technologies - such as AJAX – show the limits of automated reverse engineering approaches. A different approach for reverse-engineering that relies on screenshots and manual annotation to support an automated usability analysis based on abstracted models has been implemented in the MeMo-workbench [29].

Translation is used to transform a design model that is designed for a specific context of use to a different one without switching the level of abstraction. An example would be an interface migration between two targets - for instance between a web page and a speech interface. Although a translation maintains the level of abstraction it can be performed for instance on the task and concepts level to change the tasks that are relevant for each target (which allows maximal flexibility). Or a translation is executed on a more concrete level like for instance to change widgets representing a list from a pull-down to a radio group between different HTML targets. The latter one enables much less flexibility since the application tasks remain the same - but offers more consistency to the user.

Multi-path development offers a lot of possibilities just by model-to-model transformations. Beneath the straight forward- and reverse-engineering approaches, in practice often a mixture between these methods is performed. This is because of changing requirements during the development (such as considering more targets). Additionally the process of transformation has been adjusted to especially to overcome the complexity that is introduced by the high level of (model-based) abstractions as well as the relatively long development process through various models. In the following several methods for reducing the complexity are presented: Graceful Degradation, Factorization, Entry Points and Development by Examples.

Graceful Degradation

The method of Graceful Degradation addresses the trade-off between continuity and adaptation and has been introduced first for safety-critical systems like for instance systems in an aerospace. There Graceful Degradation describes the ability of a system to continue its services proportionally to its components that fail or are malfunctioning. Thus, under bad conditions the system is not expected to fail completely but continue providing at least its basic functions that are offered by all the non-defect components.

Graceful Degradation can be applied to user interfaces as well [16]. There it consists of specifying the user interface for the least constrained platform first and than it requires to apply transformation rules to the original interface to produce interfaces to the more constrained platforms. These transformation rules include: Splitting rules, interactor and image transformation rules, moving and resizing rules to reuse the user interface, and removal rules [31].

Factorization and Decoration

The idea of Factorization is to support multi-targeting from a “specific-target-at-a-time” approach [69]. This enables the designers to focus on one target after another. After the second target design model(s) have been realized (independently from the first), common parts are identified by comparing the design models of both targets and are factored out in a separate model. Instead of factoring out common parts to a separate model, by Decoration the designers identify a reference target, which is usually like in Graceful Degradation the least constrained target, and express exceptions by annotations to the reference model. Factorization is similar to a refactoring process, where repeating parts are cleaned up and placed in a separate component that is referenced by each component from that the code has been factored out. A detailed discussion about factorization for different context of use for task modeling has been presented by [63]. An example of decorating task models by annotations to consider different context of use has been proposed by [18].

Entry Points

An Entry Point defines an initial design model that is used to start a MDDUI process. The Cameleon Framework allows entry points at any level of abstraction and in principle engineering from any entry point in all directions (abstraction, reification, translation). In practice most available MDDUI processes offer entry-points but implement forward engineering only. Thus, for instance in the SEESCOA approach [41] the abstract user interface is the entry point to a forward-engineering process, whereas in TERESA [49] two different entry points are available: The first one at the concepts-and task model and the second one at the abstract model level.

Development by Examples

Another approach two reduce the design complexity is to design model abstractions based on concrete instances of the system by following a test-driven development approach. In such an approach the designer creates examples after every transformation to test if the actual design model is able to capture the original requirements. Since by following a transformal development approach more concrete models are based on a transformation of abstract ones, examples that have been created for a more abstract model can be reused for testing subsequent models as well [28].

After I have introduced the basic models and processes lets take a look at the second general effort in MDDUI: the focus on tool-driven design of models.

7.2.4 Tool-driven model design

Since the development of tools for model-based user interface engineering has been done since the beginnings of MDDUI, there are two much tools available to discuss them all. Tool-development has been done in two different types of categories: First comprehensive development environments have been realized. These environments tightly integrate several tools with the advantage of offering a constant mode of operation that guides a developer through all models and transformations up to the final executable user interface. Second, tool-chains have been introduced that are based on standardized exchange formats that are defined for all ontological and archetypal models. Hence, various tools, each offering the design of an archetypal model or a transformation can be chained and selected based on the requirements of the project.

For the sake of brevity I focus on giving examples for both approaches in order to discuss the pros and cons of both approaches.

Integrated Development Environments

The Model-Based Interface Designer (MOBI-D) [56, 58] was one of the first proposals for an integrated model-based interface development environment that embraces the idea of designing and developing user interfaces by creating interface models. MOBI-D included model-editing tools, user-task elicitation tools, and an interface building tool called Model-Based Interface Layout Editor (MOBILE) [57]. MOBILE includes a knowledge-based decision support system to assist the designer. By a knowledge base of interface guidelines an inference mechanism traverses the attributes and objects of the domain model to propose suitable widgets to the designer.

The Transformation-based Environment for Designing and Developing Multi-Device Interfaces (TERESA) [6, 47, 48] is composed of a method consisting of four steps that are supported by one tool. First, by high-level task modeling of a multi-context application, a single task model is designed that addresses all possible contexts of use, the involved roles, as well as identifies all objects of the domain relevant for performing the tasks. Second, a system task model is developed for all the different platforms that should be supported. The system task model for a certain platform is allowed to refine the task model of the first step. Third, from the system task model an abstract user interface is derived that considers the temporal relationships and composes a set of presentations. Each presentation is structured by the means of interactors. Finally, by a user interface generator the final user interface presentation is generated for all platforms. The TERESA tool supports all these steps and interactively guides the

developer through this top-down approach and offers a set of transformations to generate voice and web-based applications as well.

The Dygimes system [39, 22] (Dynamically Generating Interfaces for Mobile and Embedded Systems) follows the same approach as TERESA and uses a task tree as the initial model to calculate the sets of enabled tasks (ETS). Based on the ETS a dialogue model is derived that forms a state transition network [40]. In Dygimes a tool supports the designer to attach XML-based user interface descriptions to all atomic tasks. After the mappings between the task tree and the user interface fragments have been specified, Dygimes offers a tool to describe spatial layout constraints helping the designer to ensure that the interface is rendered in a visually consistent manner. Finally, a run-time library can read the constructed UI specification, which includes the task specification and adapts both to the target platform and renders the user interface based on the calculated ETS.

The Dynamic Model-based User Interface Development (DynaMo-AID) project [17] is based on an extended version of CTT task models and supports dynamic context changes. They propose decision nodes and collect distinct sub-trees from which one will be selected at run-time. Further on, DynaMo-AID includes a concrete presentation model as well as a description of domain objects. Up to now it supports to generate an application based on UIML [1] and SEESCOA XML [4]. DynaMo-AID follows a prototype-driven methodology [20] that is inspired by the spiral model. It starts from a task model, followed by the generation of a dialogue model. Combined with the concrete presentation model a prototype can be generated, evaluated and refined. [3] criticizes that no details are available on the exact notation used by the DynaMo-AID tool for the textual description of the model.

The user interface generation process in DiaTask [59] works similar to DynaMo-AID. It starts with the specification of a task model, which is transformed to a dialogue graph. It's up to the designer to relate tasks with user interface elements that are represented using XUL as the concrete user interface language.

Tool-chaining

Vanderdonkt et al. have developed a whole bunch of tools [44, 21, 24, 46] that are based on a language (The USerInterface eXtensible Markup Language - UsiXML) making the models and their associated transformations explicit.

UsiXML is aimed at offering a comprehensive approach for describing the various levels of details and abstractions of a user interface depending on the context of use. Therefore it is structured following the basic abstraction levels offered by the Cameleon Reference Framework.

At the task- and concepts-level UsiXML [36, 37, 35] offers a task and a domain model that describes the objects and classes that are presented and manipulated by the user. At the abstract level, UsiXML's abstract user interface model specifies the groups of tasks and domain concepts that should be presented together at the same time to the user. Finally, the concrete user interface model is used to give a detailed specification of the user interface appearence, that is dependent on a certain modality (for instance to generate a graphical or a voice-based user interface).

Different to earlier approaches, that mainly focus on declarative descriptions of models on certain abstraction levels (like UIML[1] or XUL[2]), UsiXML explicitly considers two additional aspects: the context of use and support for a transformational development approach.

The context of use is considered by specifying three additional models: the user model that decomposes the users of the interactive system into stereotypes and describes them by attributes for instance to express their experience with the system or a specific task or their motivation. The environment model describes the global environment where the interaction between the user and the system takes place. Properties of the environment can be physically to describe the level of noise or the lightning, or psychologically to express the level of stress of the user. Finally the platform model is used to specify the relevant attributes of the hardware and software of the device that is used to present the user interface.

The transformal development support of UsiXML is based on graph transformations and is organized by graph rewriting rules equipped with negative application conditions and attribute conditions. This transformation can be interactively constructed or manipulated using an Editor [65] or can be processed automatically. Up to now, the transformations are used between the task and concepts level and the abstract user interface level, and between the abstract and the concrete user interface level.

7.3. The pros and cons of Multi-Targeting

Several pros and cons for following a model-based development approach in user interface design have been identified and intensively discussed in the recent years. The following section introduces the major advantages and actual shortcomings.

7.3.1 Advantages of transformational MDDUI

Compared to “classical” interface development performed with interfaces builders, the transformational development has advantages in methodology, re-usability, and consistency.

Methodology

Model-based approaches are driven by specifications that are subsequently derived by a predefined process. Starting the development cycle with a specification is a widely accepted software engineering principle as [32] notes. User-centered and user interface-centered development life cycles are supported. They let designers work with tasks, users, and domain concepts instead of thinking in engineering terms [68]. These models encourage to think more about artifacts that should be realized and force the designers to explicitly represent the rationale of design decisions [68].

Relying on declarative models is a common representation that design tools can reason about to criticize designs and to detect questionable features [11, 12]. Declarative models enable realizing automated advisers that can support the designer to refine the designs. Further on user interface construction tools can be based on declarative models

[1] http://www.UIML.org, last checked 8/8/2010

[2] https://developer.mozilla.org/en/xul, last checked 8/8/2010

that enable automated creation of portions of the user interface. During run-time a declarative representation by models can be used to generate context-sensitive help to assist the user like proposed in [66].

An interactive system specification by using models enable executing the system before all details of the user interface have been designed to enable early experiments with designs by an iterative development process before considerable coding effort has been spent [68].

Re-usability

For multi-platform development of user interfaces and user interface support for context dependent adaptations model-based tools can provide the fundament for an efficient development life cycle that offers an automatic portability across different devices. Furthermore the complete description of the whole interface in a declarative form allows reusing the most interesting components.

Consistency

Consistency is the big issue that needs to be guaranteed between the user interfaces that are generated for different target platforms. Model-based approaches foster consistency since user interfaces are systematically derived by a well structured transformal process. Since the final user interfaces share abstractions – at least at the initial design model (like the task- or the abstract user interface model) consistency between different FUIs is improved.

7.3.2. Disadvantages of transformational Development

Several shortcomings have been identified so far that need to be tackled down. Commonly cited are [55, 67, 50]: The high threshold, Unpredictability, the problem of propagating modifications, their proprietary models, the efficiency and performance. Further on transformational development has not targeted real multimodal systems and only support pre-defined context of use adaptations. In the following paragraphs I present the main downsides of multi-targeting to motivate the next chapter the presents model-driven run time environments to tackle some of these disadvantages.

High threshold

The high threshold of model-based approaches is one of the big issues that need to be solved to get a broader acceptance. Up to now, developers need to learn a new language in order to express the user interface specification. Thus, model-based approaches require models to be specified in special modelling languages and therefore require a form of programming that is not suitable for many interface developers or designers.

Design tools that enable visual programming by abstracting from a certain user interface language and are integrated in widely deployed development environments are a solution to lower the threshold for model-based approaches.

Unpredictability

Each abstraction by a certain model requires the designer to understand and think in the same abstractions of the model that is utilized. The higher the abstraction that the model offers compared to the final user interface, the harder it gets for the designer to understand how the model specifications are connected to the final user interface.

[30] proposes to rely on explicit transformation rules with a tool-based preview to reduce the unpredictability of model-based approaches. [35] applied such a graph-based transformational approach, but the pure amount of transformations to map between the various models of abstraction figured out hard to overview and maintain. Additionally, the wide range of user interface element concepts of the various platforms that need to be covered by the model-transformations is complex to handle and selecting the appropriate transformation between several alternatives (often there are several ways to realize a user interface) emphasizes as a difficult and challenging task.

Propagation of modifications

Supporting several models of abstraction for the design and specification of an interactive system includes allowing changes to each of these models later on. These changes need to be propagated to the other models to maintain consistency between all models. Whereas it is consent that support for abstract-to-concrete and concrete-to-abstract (reverse engineering) as well as various entry points should be supported by model-based approaches, tools and approaches that realize these options are still missing. [30] describes the propagation of modifications as tricky, but proposes to determine the side effects on the other models entailed by the application of rules.

Proprietary models

Most model based approaches have been strongly tied to their associated model-based system and cannot be exported or are not publicly available. UsiXML [36] was the first completely available model format description. Whereas it is currently target to standardization efforts, there is still a generally accepted set of model abstractions missing. Further on, model syntax has been explicitly specified in the UsiXML specification documents, but clear model semantics have not been sufficiently specified so far.

Efficiency and Performance

Efficiency and performance of model-based systems are rarely considered. Efficiency needs to be measured for the development cycle implementing multi-platform user interfaces. Performance has to be evaluated at run-time to test if the various supported adaptations do not restrict the user's performance.

Missing support for Multimodality

Multi-targeting is focused on generating isolated user interfaces by a step-by-step for different platforms or modalities (in practice for a specific markup or programming language), which is different to the generation of multimodal user interfaces that require a strong connection between the different interfaces for the connected modalities. The Cameleon Framework supports translations for user interfaces to adapt to new contexts at run-time but still misses the possibility to interconnect different modalities for multimodal fusion. [64] solves this by adding mappings for supporting synchronization between Voice CUI and HTML CUI, which has the disadvantage that because of the late introduction of the mappings for intermodal synchronization this could and up in a cumbersome task for huge applications, because each single element of the CUI has to be addressed manually by the developer to support synchronization.

Predefined Context of use

Whereas model-based approaches introduce various abstractions to design interactive systems, each enabling a comprehensive view of the whole application, they are currently focused on analysis- and design support. Thus, these approaches require the developer to consider all combinations of devices, which the interactive system should be able to adapt to during the development process. Each new mix of devices and modalities requires going through most of these design models again to compile a new user interface.

7.4. Beyond Multi-Targeting – Seamless Interaction in Smart Environment

In contrast to current PC-based systems, user interfaces for smart environments have a stronger need for integration and the adaptation to the usage situation, comprising the available devices, the situation of the user and other context information like e.g. light and noise level of the surrounding environment.

The tight integration of user interfaces in the environment enables seamless services that are ubiquitous available and are able to accompany the user through his daily life in a smart environment. Therefore these services are able to adapt their behavior and presentation continuously to maintain there usability by switching and combining devices and interaction modes as well as transforming their presentation to reflect context of use changes.

So far several characteristics of seamless services in smart environments have been identified [7] that are introduced in the following section.

7.4.1. Characteristics of Seamless Services for Smart Environments

Plasticity

The term Plasticity is inspired by the property of materials that are able to expand and contract under natural constraints without breaking and provide continuous usage. The term has been first introduced and applied to HCI by Calvary [15, 14]. In HCI it describes the capacity of an interactive application to withstand variations of context of use while preserving usability. This also covers the contraction and expansion of the set of tasks in order to preserve usability. Plasticity is similar to multi-targeting since both terms address the diversity of context of use adaptations, but additionally express requirements in terms of usability.

User interfaces for smart environments much more rely on the possibility to adapt to the context of use. This on the one hand is due to the fact that interaction happens in various situations and under different circumstances, as well as due to the fact that multiple different devices might be used for the interaction.

Multimodality

A multimodal interface is able to “process two or more combined user input modes – such as speech, pen, touch, manual gestures, gaze, and head and body movements – in a coordinated manner with multimedia system output.” [52] The multimodal form of communication with a computer allows recognizing naturally forms of human language and behavior and is driven by the idea to support more transparent, flexible, efficient, and powerfully expressive means of human-computer-interaction [52].

There are four different relations that describe how modes can be combined based on the interaction technique that they offer: Equivalence, Assignment, Redundancy, and Complementary (the CARE properties) that have been defined in TYCOON [42].

Equivalence describes a combination of modalities in that all can be used to lead to the same desired meaning, but only one modality can be used at a time. Thus, a text input prompt can be for instance either handled by a spoken response or by using a written text typed by a keyboard.

By *assignment* a particular modality is defined to be the only one that can be used to lead to the desired meaning. (e.g. a car can only be controlled by a steering wheel).

Modalities can be used *redundant* if they can be used individual and simultaneously to express the desired meaning. Hence, for instance speech and direct manipulation are used redundantly if the user articulates “show all flights to São Paulo” while pressing on the Button “São Paulo” with the mouse.

In a *complementary* usage of modalities, multiple complementary modalities are required to capture the desired meaning. For instance a “put that there” command requires both speech and pointing gestures in order to grasp the desired meaning.

Nigay et. all [51] have detailed these relationships between modalities and applied to tasks, interaction languages and devices. In their understanding a physical device can be used to issue or present physical actions to the user (e.g. pushing a mouse button or uttering a sentence). These actions a referred to from symbols and expressions of an interaction language. An interaction language “is a language used by the user or the system to exchange information” and “defines the set of possible well-formed expressions […] that convey meaning”.

This allows a more formal definition of these relations and defining the *equivalent* and *assignment* relations as *permanent* if they hold over any state to reach a goal and as *total* of they hold for all tasks that a user can perform with a system.

The realization of redundant and complementary modality compositions is complex, since they require fusion mechanisms to process the different input received from the complementary modalities to grasp the desired meaning. There are two main types of fusion: First, by *early fusion* (or micro-fusion) signals can be integrated at a feature level and second information can be merged at a semantic level, which is referred to as *late fusion* [52]. Early fusion is more appropriate for closely temporally synchronized input modes, where one mode influences the recognition in the others. An example for early fusion for redundant modes is combining lip movements with speech recognition. Late semantic fusion (or macro-fusion) is often applied for processing less coupled temporary information, such as speech and pen input.

There are two additional challenges in multimodal fusion, the temporal distance of incoming data and the difference in input data structure if different types of modalities are used and are discussed for instance in [27].

To prevent misunderstandings and interpretation often a high degree of redundancy for multimodal interaction is chosen. But Oviatt pointed out in [52] that the dominant theme in users’ natural organization of multimodal input actually is complementary of input. Even during multimodal correction of system errors, where users are highly

motivated to clarify and reinforce their information delivery they express redundant information less than 1% of the time (tested for speech and pen input [52]).

Session Persistence

Users tend to follow various tasks in parallel. Whereas some tasks can be accomplished in short term, long term tasks might be interrupted by more important ones and continued later on. This requires the handling of interruptions and the possibility to continue a service later on. Session Persistence has been used for instance for handling database connections (and for data persistence) and for managing communication in stateless protocols like HTTP. Different to these technical processes of maintaining a network connection or an application state, user interface session persistence includes identifying a user and enabling him to stop an interaction anytime without loosing the application as well as the interaction state. Stopped sessions can then be continued on any devices and with any mode later on.

Migration

Different to a stationary PC setup, in a smart environment the user is able to move around his environment while interacting with an application. Beneath mobile devices that are continuously available to the user, other (stationary) devices that are part of the smart environment network like for instance televisions, stereos or cameras can be seamlessly integrated while the interaction takes place. Therefore a user interface requires supporting migration. [2] distinguish between *total* and *partial migration*. *Total migration* describes the capability of interfaces to move to another platform while preserving their actual state. Thus, the user can comfortably continue the interaction exactly at the point where it has been stopped on the previous platform. Migration needs to consider a switch in modality as well. If the target platform offers for instance voice recognition instead of a keyboard to enter text, the text input from the user can be spoken where it has been stopped after the user has left the PC.

Partial migration describes the distribution of parts of the interface to several platforms and modalities and is more complex to handle from both, the user's as well as the developer's perspective. For the user the interface has to include tools to select which parts of it should be distributed and offer help to indentify suitable target platforms and modalities [62]. From the developer's perspective the complexity is about to handle the composing of partial distributed interfaces to a new one. Composing is required if at least two parts of two different interfaces are targeted to run on the same platform/modality combination and can includes merging of elements to eliminate duplicates. Control interfaces are a typical example for partial user interface distribution where merging can be performed. In the iCrafter service framework [54] for instance interfaces control appliances of a presentation room can be aggregated by merging elements. There, in a presentation scenario that includes turning off several lights, all individual light user interface controls can be merged based on service interfaces that implement "PowerSwitchInterface" profiles.

Similar to classical window managers or operating system frontends that offer a set of basic user operations and forms of organization driven by the WIMP (Windows, Icons, Menus and Pointer) interaction style and direct manipulation, a frontend to the user for smart environments relies on these characteristics that I have presented in this section.

These characteristics have been utilized to realize run-time environments, which allow the creation of user interfaces beyond multi-targeting and enable their incorporation in different applications. In the following section I will present the actual state of run-time-systems supporting embedding interfaces in the environment.

7.4.2. Model-driven Run-time-Environments (MRE)

In smart environments the basic challenge for an interface is to withstand the continuously happening changes of the context of use: Users are no longer sitting in front of a stationary PC but are moving around the environment and switching between devices and modalities while interacting. This initiates a new level of comfort in human-computer-interaction – the user can now focus more on her tasks since restrictions like the limited interaction capabilities of a personal mobile device or by the fixed position of a desktop PC do no longer exist in such environments. Hence, the technical challenge is no longer about multi-targeting which can address the *predictive contexts of use* (that can be foreseen at design-time) but on interfaces that are prepared to consider the unpredictable context, which is called the *effective contexts of use* that really occur at run-time.

Current run-time systems tackle this challenge by not only describing the static structure and the behavior of the interface, but although include a description of the evolution capabilities in the models. As an example I present two different approaches, an *interactor-centric* and a *model-centric approach* for realizing such a run-time environment. The former one puts a strong focus on a modularized way of constructing interfaces through widgets and export the evolutionary part of the models into a semantic network. The latter one maintains the principle design models of multi-targeting but embeds the behavior as well as a description of the evolution capabilities into the (design) models. Both approaches add an extension mechanism that enables to introduce new adaptations and interaction capabilities to the models or the semantic network respectively.

Additionally the user in a smart environment gets much more in focus. Whereas for multi-targeting, the user was mainly the target of the design (for instance by a user-centered design process), in a smart environment the user can manipulate or change the system’s interaction behavior based on his preferences. Therefore run-time systems include a meta user interface enabling the user or a designer to inspect and manipulate the run-time state of the system.

The **COMETs approach**

The Context Mobile Widget (COMET) architecture style [26] offers self descriptive interactors that support plasticity. Thus, COMETs are able to publish their quality in use that they can guarantee for a set of contexts of use.

Following the COMETs architectural style, the user tasks of a task decomposition are reflected by individual COMETS that can be grouped to build a presentation for a task by recursively composing them of other COMETs. In such a graph every child COMET expresses itself with regard to its parent. Additionally COMETs can be defined based on task operators such as for instance the interleaving operator to render COMETs e.g. side-by-side.

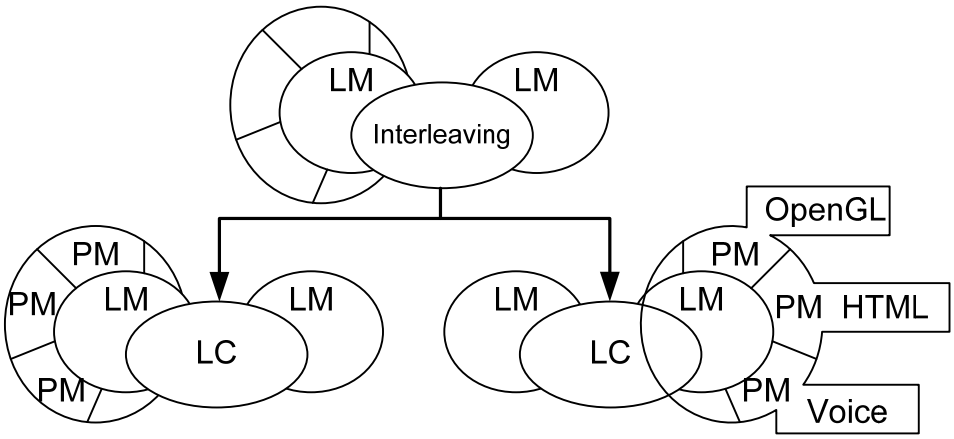

Figure 2: Graph of COMETs

Like illustrated by figure 2 each COMET is composed of three facets: 1) a logical consistency (LC), 2) a set of one or more logical models (LM) that the LC is associated to, and finally 3) a set of one or more physical models (PM) that are associated to a LM. Even if there is no fixed rule how to use these facets, the tasks, AUI, CUI, and FUI can be embodied in LCs, presentation LMs, PMs, and in technological primitives that target different languages and toolkits like for instance HTML or OpenGL respectively.

So far COMETs support the redundant and equivalent relations of the CARE properties by event propagation inside a COMET. By a domain specific language (COMET/RE) incoming events from one PM (e.g. graphics) can be specified to only get propagated by the LC to the other PMs if there is a redundant event incoming from another PM (e.g. voice).

At system run-time COMETs instantiate entities from a semantic network that describes the concepts of the different models of the CAMELEON framework like tasks, abstract containers, or concrete list boxes. The concepts of the network are structured with multiple types of relationships like for instance inheritance, abstraction, or composition.

If a graph of COMETs at run-time requires adapting to a new context of use an adaptation goal is specified and the semantic network can be queried to retrieve potential actions to transform the presentation.

The general advantage of the COMET approach lies in the fact that a COMET application can be easily extended to support new context of use adaptations by enhancing the semantic network without modifying the rest of a COMETs-based application. From a user's perspective the interactor-driven architecture has the advantage that every COMET directly expresses elements of the interface that can be directly manipulated by him (by specifying adaptation goals).

The Multi-Access Service Platform (MASP): Dynamic Executable Models

In contrast to the static design models of a multi-targeting approach, providing (only) a snapshot of the system under study at a given point in time, executable models provide the logic that defines the dynamic behavior as part of the model. The general idea is to equip executable models with "everything required to produce a desired functionality of a single Executable Models for Human-Computer Interaction problem domain" [43]. Hence, to construct an executable model, three capabilities have to be included: Beneath static element structures, the behavior, and the evolution of the system.

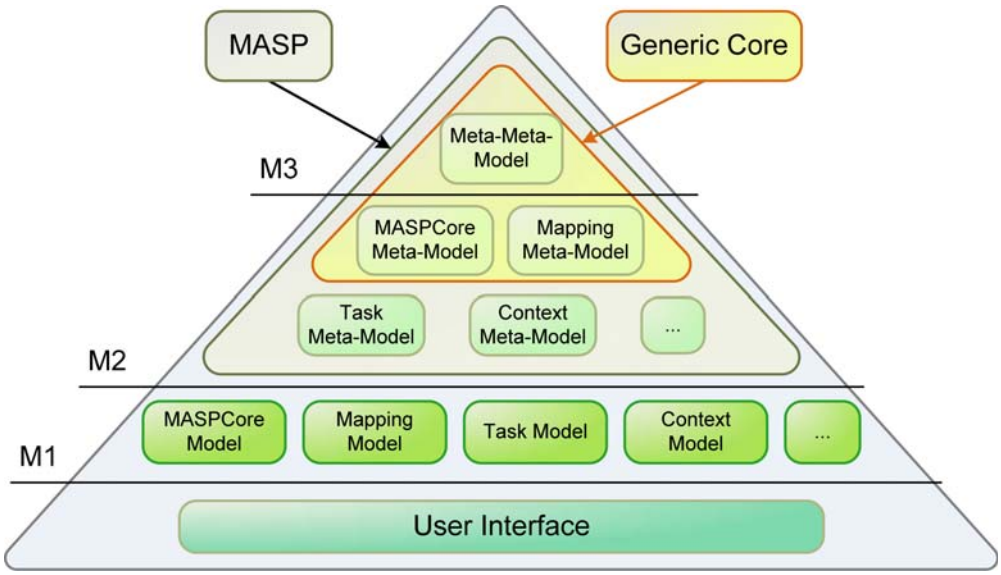

Figure 3: Executeable Models of the MASP in relation to the MOF Meta-Pyramid. Taken from [8]

Figure 3 illustrates the MOF Meta Pyramid in relation to the MASP approach that implements executable models. The core of the MASP offers an extension mechanism by a meta-meta model that distinguishes between three basic elements: definition-, situation- and execution elements that generalize all executable models.

The static structure of a design model is defined by definition elements stating all constants of a system that are not changing over time. Situation Elements define the current state of a model and capture all information that can change over time. Situation elements can trigger execution elements that are able to process and update the data of other situation elements.

The separation by the MOF model offers clear boundaries in the MASP during design-time: An application designer works with definition elements only, whereas a system architect modifies or enhances the run-time system by manipulating the meta-models of the M2 level (figure 3) to introduce for instance a new kind of task modeling or a next context description.

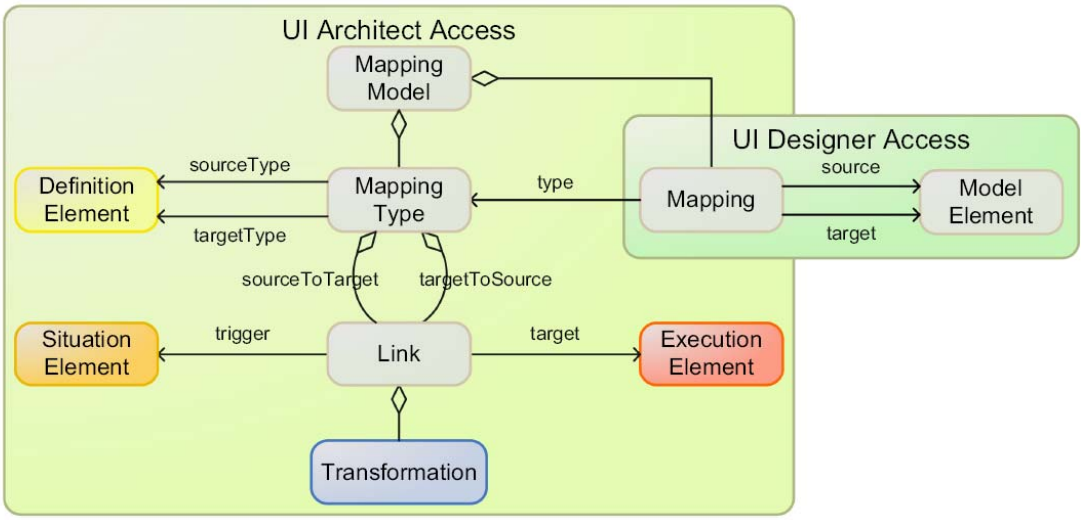

Figure 4: The Mapping Meta Model. Taken from [8].

Figure 3 depicts the mapping meta-model that connects multiple executable models in the MASP. It allows defining relations between their elements based on the structures given by the meta-meta-model and is therefore located at M2 layer of the MOF architecture. Defining the mappings in a separate model removes the problem of hard-coding them into the other models

The meta-mapping model specifies both, *mapping types* and the *mappings* that are used to synchronize elements of different models of the M1 layer, which define a particular application.

Like illustrated in figure 4, a *mapping type* refers to two definition elements that should be kept in sync. Therefore it contains *links* that get triggered by situation elements and call execution elements that process the data of the situation element. For transforming the situation element's data in an appropriate format to be processed by the execution element, a transformation can be specified. An example for such a mapping type definition is for instance the information gained from a task model, that an interaction task has been performed successfully by the user and that the interface presentation should be updated. Different to the mapping type, a concrete mapping reference a mapping type and relates it to actual application. Hence, an application developer can browse through the set of available mapping types and can create new mappings that reference the available mapping types. To implement such a mapping, the developer therefore has to set the specific source and target model elements to the mapping.

Discussion

The basic advantage of an interactor-centric approach is that interfaces can be composed by the same metaphors that are well known through interface builders: by drag-and dropping widgets on to the interfaces as well as positioning them based on the individual preferences of the user. For the user these widgets are seen as a black-box supporting the plasticity mechanisms internally. The model-centric architecture style abandons the functional grouping into separate interactors and directly executes the models of an MDDUI approach. A model-centric architecture can be directly deployed with the design models that have been developed following MDDUI as models are the

executing units instead of interactors. Since executable models just add further information like describing its behavior, already existing design models could be enhanced and already existing design tools for multi-targeting can still be used.

Meta user interfaces

Controlling services and their interfaces in a smart environment is a critical aspect in smart environments. They provide access to various services from numerous networked interaction devices for multiple users that accessing services in many different situations and via diverse combinations of devices and modalities.

Meta-user interfaces (meta-UI) are a special kind of end-user development environment that offer a set of function to control and evaluate the state of an interactive ambient space. This set is meta- because it covers all domain-dependent services that support human activities and is UI oriented because of its role enabling the user to control and to evaluate the ambient space [23].

Figure 5: Meta-UI Implementation of the MASP taken from [61].

Figure 5 depicts the meta-UI of the MASP with a running service interface. So far, the meta-UI offers four generic services that are implemented by the button bar at the bottom of the meta-UI: modality-, migration-, adaptation-, and distribution-configuration. By the modality configuration a user can set the utilized modalities. This is currently limited to the definition of equivalent related modalities, but helps the user to prevent unwanted side effects, like disabling a speech-recognition in noisy

environments or during a phone call for instance. The actual activated modalities are signaled by the status bar at the top of the meta-UI.

The migration and distribution configurations enable total respectively partial migration of interfaces. The migration is not limited to intra-modal remolding but supports a free selection of equivalent modalities for the target platform(s) of the migration. Finally with the adaptation functionality mixed-by-design objects [23] can be specified. Mixed-by-design objects couple physical entities with digital services that have been assembled during design-time. One such mixed-by-design object that is used in the MASP meta-UI is an active RFID tag that is coupled to a follow-me digital service. Activating this service results in a user-interface that continuously remolds to follows the user through the environment.

The COMETs Inspector implements such a meta-UI as well. Different to the MASP meta-UI that is so far limited by the mapping types that have been defined at system design-time, the COMETs inspector can be used to query the semantic network that stores and relates all instantiated interactors on all levels of the CAMELEON framework. Thus, it supports inspecting the current state of the system as well as issuing queries to the semantic network to retrieve potential transformation options. As stated in [25] the inspector is actual limited to basic operations like adding, removing or substituting elements, but coupled with the querying option to the semantic networks a first step to end-user development has been done since in principle transformation recipes and alternatives can be queried and applied by the user by this basic operations.

For a detailed discussion and characterization of meta-UIs based on a taxonomy I recommend [23].

7.5. Challenges for realizing Interfaces for Smart Environments

7.5.1 Modeling Real and Flexible Multimodal Interaction

Nowadays multimodal systems that support the user by a combination of speech, gesture and graphical-driven interaction are already part of our everyday life. Examples are combinations of speech-driven and graphical interfaces like in-car assistance systems, language-learning software, or tourist information systems. Recent video-game consoles just introduced gesture-based interactions like for instance the Nintendo Wii game console. Games can be controlled in a more natural and intuitive way by using hand gestures, balancing and moving the body. The market success of this console demonstrates that even new audiences can be addressed by enabling multimodal interaction to ease the usage of interactive systems.

In the recent years of research on multimodal systems there has been great success in demonstrating that one big research issue, the “multimodal fusion”, that is often motivated with the “put this there” example [9], can be handled. This example describes the problem of interpreting a user interaction spanning over several modalities (which requires the machine to combine the gesture, voice and graphical inputs of the user with his actual context to catch the user’s goals). In Germany for example, the SmartCom project [33, 73] realized such a system by using the blackboard metaphor to implement an interactive avatar that can interpret and react to multimodal user interaction. The multimodal fusion of speech, gestures, graphical inputs, and users’ emotions is driven by a standardized information exchange language (M3L) to enable the fusion of (uncertain) information of the different modalities that are connected to a multimodal setup. Beneath this large scale system, recent research combines component repositories containing re-usable device drivers and fusion mechanisms with interactive design tools to create and prototype various multimodal setups. One such example that receives a lot of attention recently is the Open Interface Platform [34] another popular one is iStuff [60].

This recent research results allow to rapidly creating sets of very different and even very specialized multimodal setups realizing control interfaces with the help of interactive editors. But so far they lack of efficient software engineering methods and notations to support the application design of multimodal interaction for such heterogeneous types of interaction devices and modalities.

Modeling multimodal systems that support various multimodal setups is an open research issue. A recent promising work by [64] that implemented a model-based development process to generate multimodal web interfaces stated the importance of considering the CARE-properties, but neglected the support of modeling complementary or redundant multimodal interaction to support multimodal fusion. UsiXML [37] and TERESA [5] do not offer a dialogue model but distribute it to several models (mainly the task model that specifies the interaction sequence and the abstract user interface model containing the navigation). Since these models are considered modality independent, supporting different navigations based on the used combination of modalities is difficult to implement. Based on the modality (e.g. tactile, speech or graphics (large screen vs. small screen with pagination) or the composition of modalities the navigation needs to be realized completely different.

7.5.2. The Mapping Explosion and Modeling Tool Problem

The mapping problem can be seen as a direct consequence following a model-based user interface development approach. The mapping problem has been firstly introduced by Angel Puerta and Jakob Eisenstein [58] who state: ”if for a given user interface design it is potentially meaningful to map any abstract model element to any concrete one, then we would probably be facing a nearly insurmountable computation problem”. Figure 3 illustrates the mapping problem between a set of models.

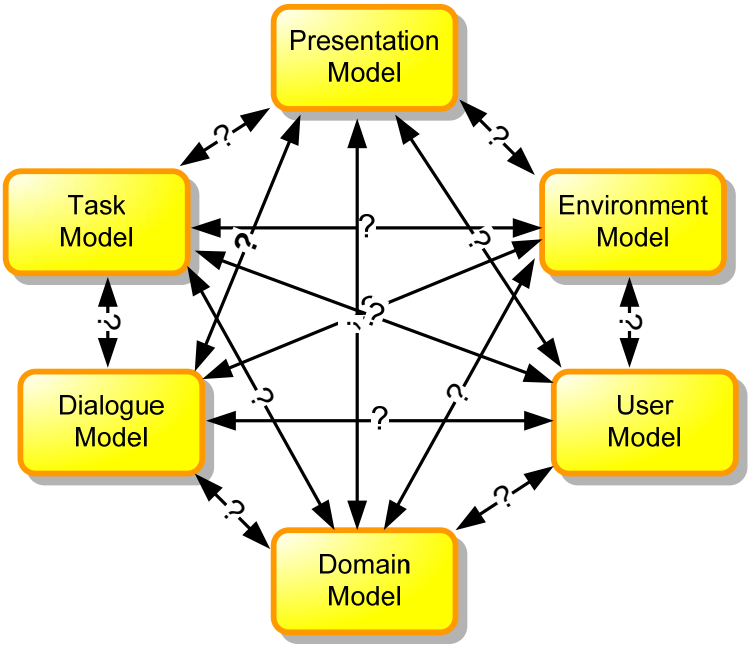

Figure 5: The mapping explosion problem

Solving the mapping problem comprises answering, which models should be combined, at which level of abstraction, and how the models are combined. With the introduction of run time systems that awake the original design models alive by making them executable, the original mapping problem explodes. Now there are two kinds of mappings that need to be specified: transformational mappings at design time and mappings at runtime that keep all running models in sync.

Like illustrated figure 3 the worst case would be to map every element of every model to every element of every other model. One could argue that by a transitional propagation the amount op mapping could be reduced. But there is an inherent performance requirement by the keep various modes and medias in sync at run time. Since each mapping typical requires transformational part that does the abstract-to-concrete (or vice versa) translation this can be challenging. To my knowledge so far performance evaluations of model-driven run time environments are missing.

There has been a lot of research about mapping mechanisms at design time that might be adapted to run time by an automated generation of runtime mappings based on the transformations that have been proposed at design time. For instance [71, 72] describe three mechanisms that can be used to establish mappings between the models: model derivation, model linking/binding, and model composition. [19] extended this

classification to five mechanisms: model derivation, partial model derivation, model linking, model modification, and model update.

Beneath the challenge to solve the mapping problem between the various models, so far the design of mapping is supported by tools only to a very limited extend - and so far only at design time. For instance [45] define a formal mapping between the domain, task and abstract user interface elements, which allows to establish a set of mappings either manually or automatically (in combination with the TransformiXML tool [65]). They offer eight different types of mapping relationships that can be defined.

Working with these transformational approaches based on mappings introduces two major problems. First, it is very formalized and programmatic approach that is hard to understand and to extend and second, supporting the flexible combining of modalities at run time for multimodal interaction would result in merging these mappings and requires them to interact which each other, which is considered even more complex to manage.

7.5.3 Participation of a Designer and End User Development

Important groups are not involved in current MDDUI design cycles: neither the end user as well as an interaction designer can participate actual MDDUI development processes. The end user suffers from complexity and the designer from gap in tools between those that support their workflow and those that are currently available in MDDUI.

Initial efforts have been done by realizing specialized tools for designers to include prototyping approaches on different levels of fidelity: For instance SketchiXML [24], a sketching editor is targeted to low-fidelity prototyping and replaces paper sketching of user interfaces, the VisiXML tool [70] supports vector drawing, whereas GrafiXML [36] is an advanced user interface editor supporting high-fidelity prototyping. One common aspect of all these prototyping tools is that they focus on the design of graphical user interfaces. Supporting interaction design for smart environments that includes considering context of use adaptation and flexible multimodal is so far not covered.

Further on, the fundamental challenge to develop environments that enable people without particular background in programming to realize their own interactive applications [53] still exists. Although first approaches based on the meta-UI paradigm have been proposed that I have presented earlier. These are targeted to enable end users to manipulate an interactive space and can be seen as an initial step to support end user development.

7.6. References

[1] Marc Abrams und Jim Helms: *User Interface Markup Language (UIML) Specification version 3.0*; Techn. Ber.; Harmonia Inc.; 2002.

[2] Renata Bandelloni und Fabio Paterno: *Flexible Interface Migration*; in *Proceedings of the 9th International Conference on Intelligent User Interfaces*; S. 148 – 155; Funchal, Madeira, Portugal; 2004; ACM Press New York, NY, USA.

[3] Jan Van den Bergh: *High-Level User Interface Models for Model-Driven Design of Context-Sensitive User Interfaces*; Dissertation; Hasselt University and transnational University of Limburg School of Information Technology; 2006.

[4] Jan Van den Bergh und Karin Coninx: *Model-based design of context-sensitive interactive applications: a discussion of notations*; in *TAMODIA '04: Proceedings of the 3rd annual conference on Task models and diagrams*; S. 43–50; New York, NY, USA; 2004; ACM Press.

[5] Silvia Berti, Francesco Correani et al.: *TERESA: A Transformation-based Environment for Designing and Developing Multi-Device Interfaces*; in *ACM CHI 2004, Extended Abstract*; Bd. II; S. 793–794; Vienna, Austria; April 2004; ACM Press.

[6] Silvia Berti und Fabio Paterno: *Migratory MultiModal Interfaces in MultiDevice Environments*; in *ICMI '05: Proceedings of the 7th international conference on Multimodal interfaces*; S. 92–99; New York, NY, USA; 2005; ACM Press.

[7] Marco Blumendorf, Sebastian Feuerstack und Sahin Albayrak: *Multimodal Smart Home User Interfaces*; in *Intelligent User Interfaces for Ambient Assisted Living: Proceedings of the First International Workshop IUI4AAL 2008* (Hg. Kizito Mukasa, Andreas Holzinger und Arthur Karshmer). IRB Verlag; 2008.

[8] Marco Blumendorf, Grzegorz Lehmann et al.: *Executable Models for Human-Computer Interaction*; in *Interactive Systems. Design, Specification, and Verification: 15th International Workshop, DSV-IS 2008 Kingston, Canada, July 16-18, 2008 Revised Papers* (Hg. T. C. Nicholas Graham und Philippe Palanque); S. 238–251; Berlin, Heidelberg; 2008; Springer-Verlag.

[9] Richard A. Bolt: *"Put that there": Voice and Gesture at the Graphics Interface*; in *Proceedings of the 7th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '80)*; S. 262–270; New York, NY, USA; 1980; ACM Press.

[10] Laurent Bouillon, Jean Vanderdonckt und Kwok Chieu Chow: *Flexible re-engineering of web sites*; in *IUI '04: Proceedings of the 9th international conference on Intelligent user interfaces*; S. 132–139; New York, NY, USA; 2004; ACM.

[11] Robert Eric Braudes: *A framework for conceptual consistency verification*; Dissertation; George Washington University; Washington, DC, USA; 1990.

[12] Michael D. Byrne, D. Wood et al.: *Automating interface evaluation*; in *CHI Conference Companion* (Hg. Catherine Plaisant); S. 216. ACM; 1994.

[13] Gaelle Calvary, Joelle Coutaz et al.: *A Unifying Reference Framework for Multi-Target User Interfaces*; *Interacting with Computers*; 15(3), S. 289–308; 2003.

[14] Gaelle Calvary, Joelle Coutaz et al.: *Towards a New Generation of Widgets for Supporting Software Plasticity: The "Comet"*; in *Proceeidings of Engineering Human Computer Interaction and Interactive Systems, Joint Working Conferences EHCI-DSVIS 2004. ISSN 0302-9743* (Hg. Remi Bastide, Philippe A.

Palanque und Joerg Roth); Bd. 3425 von *Lecture Notes in Computer Science*; S. 306–324; Berlin/Heidelberg; 2004; Springer.

[15] Gaelle Calvary, Joelle Coutaz und David Thevenin: *Supporting Context Changes for Plastic User Interfaces: A Process and a Mechanism*; in *Joint Proceedings of HCI'2001 and IHM'2001* (Hg. Jean Vanderdonckt A. Blandford und P. Gray); S. 349–363; Lille; September 2001; Springer-Verlag.

[16] Hao hua Chu, Henry Song et al.: *Roam, A seamless application framework*; *Journal of Systems and Software*; 69(3), S. 209–226; 2004.

[17] Tim Clerckx, Kris Luyten und Karin Coninx: *DynaMo-AID: A Design Process and a Runtime Architecture for Dynamic Model-Based User Interface Development*; in *The 9th IFIP Working Conference on Engineering for Human-Computer Interaction Jointly with The 11th International Workshop on Design, Specification and Verification of Interactive Systems (EHCI/DS-VIS), ISBN 3-540-26097-8*; Bd. 3425; S. 77–95; Springer, Berlin; 2004.

[18] Tim Clerckx, Kris Luyten und Karin Coninx: *Generating Context-Sensitive Multiple Device Interfaces from Design*; in *Pre-Proceedings of the Fourth International Conference on Computer-Aided Design of User Interfaces CADUI 2004*; 2004.

[19] Tim Clerckx, Kris Luyten und Karin Coninx: *The Mapping Problem Back and Forth: Customizing Dynamic Models while Preserving Consistency*; in *TAMODIA* (Hg. Pavel Slavik und Philippe A. Palanque); S. 33–42; Prague, Czech Republic; 2004; ACM International Conference Proceeding Series.

[20] Tim Clerckx, Chris Vandervelpen et al.: *A Prototype-Driven Development Process for Context-Aware User Interfaces*; in *Proceedings of the 5th International Workshop,Task Models and Diagrams for Users Interface Design (TAMODIA 2006)*; Bd. 4385 von *Lecture Notes in Computer Science*; S. 339–354; Berlin / Heidelberg; August 2006.

[21] Benoit Collignon, Jean Vanderdonkt und Gaelle Calvary: *Model-Driven Engineering of Multi-Target Plastic User Interfaces*; in *The Fourth International Conference on Autonomic and Autonomous Systems (ICAS 2008)*; 2008; IEEE Computer Society Press, March 16-21, 2008 - Gosier, Guadeloupe.

[22] Karin Coninx, Kris Luyten et al.: *Dygimes: Dynamically Generating Interfaces for Mobile Computing Devices and Embedded Systems*; in *Mobile HCI* (Hg. Luca Chittaro); Bd. 2795 von *Lecture Notes in Computer Science*; S. 256–270. Springer; 2003.

[23] Joelle Coutaz: *Meta-User Interfaces for Ambient Spaces*; in *TAMODIA* (Hg. Karin Coninx, Kris Luyten und Kevin A. Schneider); Bd. 4385 von *Lecture Notes in Computer Science*; S. 1–15. Springer; 2006.

[24] Vanderdonckt J. Coyette, A. und Q. Limbourg: *SketchiXML: An Informal Design Tool for User Interface Early Prototyping*; in *Proceedings of RISE 2006 Workshop on Rapid User Interface Prototyping Infrastructures Applied to Control Systems RUIPICAS 2006* (Hg. V. Amaral M. Risoldi); Geneva; September 2006 2006.

[25] Alexandre Demeure, Gaelle Calvary et al.: *The Comets Inspector: Towards Run Time Plasticity Control based on a Sematic Network*; in *TAMODIA '06: Proceedings of the 5th annual conference on Task Models and DIAgrams,ISBN978-3-540-70815-5*; Bd. 4385 von *Lecture Notes in Computer Science*; S. 324–338; Berlin / Heidelberg; 2006; Springer.

[26] Alexandre Demeure, Gaelle Calvary und Karin Coninx: *COMET(s), A Software Architecture Style and an Interactors Toolkit for Plastic User Interfaces*; S. 225–237; 2008; Design, Specification, and Verification, 15th International Workshop, DSV-IS 2008, T.C.N. Graham & P. Palanque (Eds), Lecture Notes in Computer Science 5136, Springer Berlin / Heidelberg, Kingston, Canada, July 16-18, 2008, pp 225-237.

[27] Bruno Dumas, Denis Lalanne et al.: *Strengths and weaknesses of software architectures for the rapid creation of tangible and multimodal interfaces*; in *TEI '08: Proceedings of the 2nd international conference on Tangible and embedded interaction*; S. 47–54; New York, NY, USA; 2008; ACM.

[28] Sebastian Feuerstack: *A Method for the User-centered and Model-based Development of Interactive Applications*; Dissertation; Technische UniversitÃ¤t Berlin; 2008.

[29] Sebastian Feuerstack, Marco Blumendorf et al.: *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*; S. 134–141; Berlin, Heidelberg; 2008; Springer-Verlag.

[30] Murielle Florins: *Graceful Degradation: a Method for Designing Multiplatform Graphical User Interfaces*; Dissertation; Universite Catholique de Louvain; Louvain-la-Neuve, Belgium; July 11th 2006.

[31] Murielle Florins, Francisco Montero Simarro et al.: *Splitting Rules for Graceful Degradation of User Interfaces*; in *IUI '06: Proceedings of the 11th international conference on Intelligent user interfaces*; S. 264–266; New York, NY, USA; 2006; ACM Press.

[32] Carlo Ghezzi, Mehdi Jazayeri und Dino Mandrioli: *Fundamentals of software engineering*; Prentice-Hall, Inc.; Upper Saddle River, NJ, USA; 1991.

[33] Gerd Herzog und Alassane Ndiaye: *Building Multimodal Dialogue Applications: System Integration in SmartKom*; in *SmartKom: Foundations of Multimodal Dialogue Systems* (Hg. Wolfgang Wahlster); S. 439–452; Springer; Berlin, Heidelberg; 2006.

[34] Jean-Yves Lionel Lawson, Ahmad-Amr Al-Akkad et al.: *An open source workbench for prototyping multimodal interactions based on off-the-shelf heterogeneous components*; in *EICS '09: Proceedings of the 1st ACM SIGCHI symposium on Engineering interactive computing systems*; S. 245–254; New York, NY, USA; 2009; ACM.

[35] Quentin Limbourg: *Multi-Path Development of User Interfaces*; Dissertation; Universite Catholique de Louvain, Institut d'Administration et de Gestion (IAG); Louvain-la-Neuve, Belgium; September 2004.

[36] Quentin Limbourg, Jean Vanderdonckt et al.: *USIXML: A Language Supporting Multi-path Development of User Interfaces*; in *EHCI/DS-VIS* (Hg. Remi Bastide, Philippe A. Palanque und Joerg Roth); Bd. 3425 von *Lecture Notes in Computer Science*; S. 200–220. Springer; 2004.

[37] Quentin Limbourg, Jean Vanderdonckt et al.: *USIXML: A User Interface Description Language for Context-Sensitive User Interfaces*; in *Proceedings of the ACM AVI'2004 Workshop" Developing User Interfaces with XML: Advances on User Interface Description Languages*; S. 55–62; 2004.

[38] L.Paganelli und F.Paterno: *A Tool for Creating Design Models from Web Site Code*; *International Journal of Software Engineering and Knowledge Engineering*; 13(2), S. pp. 169–189; 2003.

[39] Kris Luyten: *Dynamic User Interface Generation for Mobile and Embedded Systems with Model-Based User Interface Development*; Dissertation; Transnationale Universiteit Limburg, School voor Informatietechnologie; October 2004.

[40] Kris Luyten, Tim Clerckx et al.: *Derivation of a Dialog Model from a Task Model by Activity Chain Extraction.*; in *Interactive Systems: Design, Specification, and Verification, 10th International Workshop (DSV-IS)*; S. 203–217; Funchal, Madeira Island, Portugal; June 2003.

[41] Kris Luyten, Tom Van Laerhoven et al.: *Runtime transformations for modal independent user interface migration*; *Interacting with Computers*; 15(3), S. 329–347; 2003.

[42] Jean-Claude Martin: *TYCOON: Theoretical Framework and Software Tools for Multimodal Interfaces*; *Intelligence and Multimodality in Multimedia interfaces, AAAI Press*; 1998.

[43] Stephen J. Mellor: *Agile MDA*; Techn. Ber.; Project Technology, Inc.; June 2004.

[44] Vanderdonckt-J. Michotte, B.: *GrafiXML, A Multi-Target User Interface Builder based on UsiXML*; in *Proceedings of 4th International Conference on Autonomic and Autonomous Systems ICAS 2008*. IEEE Computer Society Press; 16-21 March 2008.

[45] Francisco Montero, Vctor López-Jaquero et al.: *Solving the Mapping Problem in User Interface Design by Seamless Integration in IdealXML*; in *Proceedings of 12th Int. Workshop on Design, Specification, and Verification of Interactive Systems (DSV-IS'2005)*; S. 161–172; 2005.

[46] Francisco S. Montero und Victor Lopez-Jaquero: *Fast HI-FI prototyping by using IdealXML*; Techn. Ber.; Departamento de Sistemas Informaticos, Universidad de Castilla-La Mancha; March 2006.

[47] Giulio Mori, Fabio Paterno und Carmen Santoro: *CTTE: Support for Developing and Analyzing Task Models for Interactive System Design.*; *IEEE Trans. Software Eng.*; 28(8), S. 797–813; 2002.

[48] Giulio Mori, Fabio Paterno und Carmen Santoro: *Tool support for designing nomadic applications*; in *IUI '03: Proceedings of the 8th International Conference on Intelligent User Interfaces*; S. 141–148; New York, NY, USA; 2003; ACM Press.

[49] Giulio Mori, Fabio Paterno und Carmen Santoro: *Design and Development of Multidevice User Interfaces through Multiple Logical Descriptions*; *IEEE Transactions on Software Engineering*; 30(8), S. 507–520; 2004.

[50] Brad Myers, Scott E. Hudson und Randy Pausch: *Past, present, and future of user interface software tools*; *ACM Transactions on Human-Computer Interaction*; 7(1), S. 3–28; 2000.

[51] Laurence Nigay und Joelle Coutaz: *Multifeature Systems: The CARE Properties and Their Impact on Software Design*; in *Intelligence and Multimodality in Multimedia Interfaces*; 1997.

[52] Sharon Oviatt: *The Human-Computer Interaction Handbook: Fundamentals, Evolving Technologies and Emerging Applications, 2nd Edition*; Kap. Multimodal Interfaces, S. 413–432; Lawrence Erlbaum; 2. Aufl.; 2008.

[53] Fabio Paterno: *Model-based Tools for Pervasive Usability*; *Interacting with Computers*; 17(3), S. 291–315; 2005.
[54] Shankar Ponnekanti, Brian Lee et al.: *ICrafter: A Service Framework for Ubiquitous Computing Environments.*; in *Ubicomp*; S. 56–75; 2001.
[55] Angel R. Puerta: *The MECANO Project: Comprehensive and Integrated Support for Model-Based Interface Development.*; in *Computer-Aided Design of User Interfaces (CADUI'96)*; S. 19–36; 1996.
[56] Angel R. Puerta: *A Model-Based Interface Development Environment*; *IEEE Softw.*; 14(4), S. 40–47; 1997.
[57] Angel R. Puerta, Eric Cheng et al.: *MOBILE: User-centered interface building*; in *CHI '99: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*; S. 426–433; New York, NY, USA; 1999; ACM Press.
[58] Angel R. Puerta und Jacob Eisenstein: *Towards a General Computational Framework for Model-Based Interface Development Systems*; in *IUI99: International Conference on Intelligent User Interfaces*; S. 171–178; New York, NY, USA; January 1999; ACM; ISBN:1-58113-098-8.
[59] Daniel Reichard, Peter Forbrig und Anke Dittmar: *Task Models as Basis for Requirements Engineering and Software Execution*; in *Proceedings of TAMODIA 2004*; S. 51 – 57; Prague, Czeck Republic; 2004; ACM Press.
[60] Meredith Ringel, Joshua Tyler et al.: *iStuff: A Scalable Architecture for Lightweight, Wireless Devices for Ubicomp User Interfaces*; Proceedings of UBICOMP 2002; 2002.
[61] Dirk Roscher, Marco Blumendorf und Sahin Albayrak: *Using Meta User Interfaces to Control Multimodal Interaction in Smart Environments*; in *Proceedings of the IUI'09 Workshop on Model Driven Development of Advanced User Interfaces*; 2009.
[62] Dirk Roscher, Marco Blumendorf und Sahin Albayrak: *A Multimodal User Interface Model For Runtime Distribution*; in *Proceedings of the CHI'10 Workshop on Model Driven Development of Advanced User Interfaces*; Atlanta, Georgia, USA.; 2010.
[63] N. Souchon, Q. Limbourg und Jean Vanderdonckt: *Task Modelling in Multiple contexts of Use*; in *Interactive Systems:Design, Specification, and Verification (DSV-IS) 2002*; Bd. 2545/2002; S. 59–73. Springer Berlin / Heidelberg; 2002.
[64] Adrian Stanciulescu: *A Methodology for Developing Multimodal User Interfaces of Information Systems*; Dissertation; Universite Catholique de Louvain; 2008.
[65] Adrian Stanciulescu, Quentin Limbourg et al.: *A transformational approach for multimodal web user interfaces based on UsiXML*; in *ICMI '05: Proceedings of the 7th International Conference on Multimodal Interfaces*; S. 259–266; New York, NY, USA; 2005; ACM Press.
[66] Piyawadee Noi Sukaviriya: *Dynamic Construction of Animated Help from Application Context*; in *ACM Symposium on User Interface Software and Technology*; S. 190–202; 1988.
[67] Pedro A. Szekely: *Retrospective and Challenges for Model-Based Interface Development*; in *DSV-IS 96: 3rd International Eurographics Workshop on Design, Specification, and Verification of Interactive Systems* (Hg. François Bodart und Jean Vanderdonckt); S. 1–27. Springer; 1996.
[68] Pedro A. Szekely, Piyawadee Noi Sukaviriya et al.: *Declarative interface models for user interface construction tools: The MASTERMIND approach*; in

Proceedings of the IFIP TC2/WG2.7 Working Conference on Engineering for Human-Computer Interaction; S. 120 – 150; 1995; ISBN:0-412-72180-5.

[69] D. Thevenin: *Adaptation en Interaction Homme-Machine: Le cas de la Plasticite.*; Dissertation; Ph.D. thesis, Universite Joseph Fourier, Grenoble I; 2001.

[70] Jean Vanderdonckt: *A MDA-Compliant Environment for Developing User Interfaces of Information Systems*; in *CAiSE 2005: Advanced Information Systems Engineering.* (Hg. Oscar Pastor und Joao Falcao e Cunha); Bd. 3520 von *Lecture Notes in Computer Science*; S. 16–31. Springer; 2005.

[71] Jean Vanderdonckt, Quentin Limbourg et al.: *Using a Task Model to Derive Presentation and Dialog Structure*.

[72] Jean Vanderdonckt, Quentin Limbourg und Murielle Florins: *Deriving the Navigational Structure of a User Interface*; in *Proceedings of IFIP INTERACT'03: Human-Computer Interaction*; 2: HCI methods; S. 455; 2003.

[73] Wolfgang Wahlster: *SmartKom: Fusion and Fission of Speech, Gestures, and Facial Expressions*; in *Proc. of the 1st International Workshop on Man-Machine Symbiotic Systems*; 2002.