A Multimodal User Interface Model For Runtime Distribution

Dirk Roscher, Marco Blumendorf, Sahin Albayrak
DAI-Labor, TU-Berlin
Ernst-Reuter-Platz 7, D-10587 Berlin, Germany
{Dirk.Roscher, Marco.Blumendorf, Sahin.Albayrak}@DAI-Labor.de

ABSTRACT

Smart environments provide numerous networked interaction resources (IRs) allowing users to interact with services in many different situations via many different (combinations of) IRs. In such environments it is necessary to adapt the user interface dynamically at runtime to each new situation to allow an ongoing interaction in changing contexts. Our approach allows to dynamically select the combination of IRs that are suitable for the interaction in the current context at any time. The decision is based on information from a user interface model executed at runtime and a context model gathering information about the environment. The user interface model supports the CARE properties to specify flexible multimodal interaction.

Author Keywords
User interface distribution, model-based development, executable models

INTRODUCTION

Smart environments provide numerous networked interaction resources (IRs) allowing users to interact with services in many different situations via many different (combinations of) IRs. In such environments user interfaces (UIs) need to take a changing context of use into account [6]. This makes the alteration of the used IRs and thus the dynamic (re-) combination of the resources at runtime an important aspect.

In this work, we first present the requirements to dynamically adjust the used IRs at runtime (section 3). Afterwards, our model-based approach targeting the requirements is described. A UI model reflects the various UI elements as well as the relations between them in terms of CARE properties [7] to achieve multimodal interaction (section 4). At runtime, the modeled UI description in combination with information about the available IRs from an additional context model is used to continuously adjust the UI distribution according to the current situation (section 5). Before we describe our approach, related work is presented in the next section.

Pre-proceedings of the 5th International Workshop on Model Driven Development of Advanced User Interfaces (MDDAUI 2010): Bridging between User Experience and UI Engineering, organized at the 28th ACM Conference on Human Factors in Computing Systems (CHI 2010), Atlanta, Georgia, USA, April 10, 2010.

RELATED WORK

Model-based development is a promising approach in the context of multimodal UIs. According to the classification of the Cameleon Reference Framework proposed in [6] UI models feature four levels of abstraction: Concepts and Task Model, Abstract, Concrete and Final User Interface. Based on this general framework, several User Interface Description Languages (UIDLs) have been designed. The most relevant with respect to the goals of this work are UsiXML [11] and TERESA [2]. Their goal is to develop multimodal UIs but they only support a fixed set of IRs, whereas we aim to support sets of IRs changing at runtime.

The distribution of UIs has also been a topic of various research activities, ranging from the characterization of distributed UIs [8] to development support for specifying distributed UIs [12]. The approach of Elting and Hellenschmidt [9] supports simple conflict resolution strategies when distributing output across graphical UIs, speech syntheses and virtual characters. The main goal is the semantic processing of input and output in distributed systems. The dynamic redistribution and definition of dynamic UI models has thus not been the focus of the approach. The I-AM (Interaction Abstract Machine) system [1] presents a software infrastructure for distributed migratable UIs. It provides a middleware for the dynamic coupling of IRs to form a unified interactive space. The approach supports dynamic distribution across multiple heterogeneous platforms but does not support the arbitrary recombination of IRs and is limited to graphical output as well as mouse and keyboard input.

Our approach utilizes the modeled design information at runtime to dynamically adjust the combination of the used IRs. In the following we first describe the requirements that need to be fulfilled to allow the distribution of multimodal UIs at runtime. Afterwards, we show how these requirements are implemented by our approach.

DYNAMIC UI DISTRIBUTION

To support UI (re-) distribution at runtime several requirements needs to be fulfilled, that are derivated from the abstract architecture depicted in Figure 1:

1. A user interface description is needed that supports different variants of multimodal interaction and benefits from the advantages of specific modalities and modality combinations. Information about the supported modality combinations need to be available at runtime.

2. To combine IRs from different platforms, the user interface description needs to address single IRs.

3. Environment information must be gathered and kept up to date (e.g. IRs, users and their positions).

4. An instance that incorporates information about the UI and the environment is required which determines the most appropriate combination of IRs at any time.

5. The combination of arbitrary IRs from different platforms also requires a mechanism that allows to control IRs independently from each other.

The first two requirements are fulfilled by our multimodal UI model with different presentations and input possibilities as described in the next subsection. Afterwards we show how this model is used to create multimodal UIs by selecting the most appropriate combination of IRs (requirements 3 to 5).

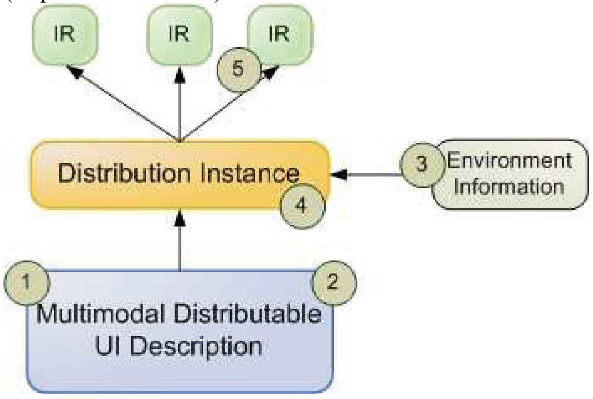

Figure 1: Abstract architecture for distributing UIs.

MULTIMODAL EXECUTABLE UI MODEL

The distribution of UIs at runtime requires certain information about the UI at runtime (part of requirement 1 and 2). To achieve this, our approach is based on the notion of executable models that combine the static design information, execution logic and runtime state information of the UI [4]. This allows to execute and observe their status at runtime as well as to access design information.

Our set of metamodels for specifying distributable multimodal UIs follows the Cameleon reference Framework [6] and thus we distinguish tasks and concepts, abstract interface, concrete interface and final interface as also done in TERESA and UsiXML (see Figure 2). To show how to develop a UI with our set of models, we explain the short example presented in Figure 2. The example is an excerpt from our cooking assistant and models a recipe selection scenario. The presentation of the recipe is possible via different modality combinations and can be confirmed via several input styles.

We first specify the workflow of the example with a task model (we use an extended version of Concurrent Task Trees for the definition [10]) and thus start with the definition of the "ConfirmSelection"-task (*T1: ConfirmRecipe*). Afterwards, the abstract interaction(s) for each task is specified by choosing between *OutputOnly*, *FreeInput*, *Choice* and *Command* (similiar to UsiXML and TERESA) or *ComplexInteractor* to aggregate several abstract interactions. So we choose one abstract interaction object for the presentation of the selected recipe (*A1:OutputOnly*) and one for the confirmation (*A2:Selection*). The abstract interactors are connected via mappings to the "ConfirmSelection"-task (see Figure 2). This is one huge difference between transformational approaches like UsiXML and the executable models approach. Because of the parallel execution of all models (task, abstract and concrete), the information from all models is available and does not need to be transformed from one model into another. Each model only contains the information of its abstraction level and information from different models is connected via mappings.

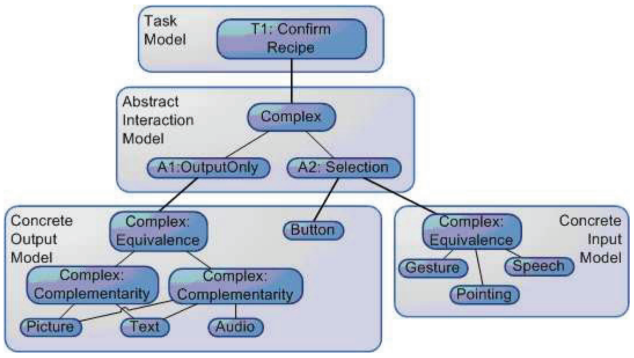

Figure 2: Model structure and interaction example.

In the next step the concrete interaction is specified by using the concrete input and concrete output model. The separation of input and output is another difference to other approaches but allows the independent addressing of single IRs (requirement 2). However, it requires to handle IRs with combined input and output like touchscreens. By utilizing the CARE properties, developers can specify their intention on how to combine the different modalities (requirement 1). Defining multimodal relations with the CARE-properties is similar to the ICARE software components [5]. In contrast to ICARE however, the components and thus the multimodal relationships are not statically related at design time but can be freely configured between arbitrary modalities through the integration in the interaction metamodel and evaluated at runtime.

To present the recipe (*A1:OutputOnly*) the developer chooses two different presentation possibilities: one for graphical output (*Picture* and *Text*) and one with additional natural language output (*Audio*). Each possibility is aggregated by a complex interactor and the Complementarity-attribute means that the children complement each other and must be presented together.

Both possibilities are also aggregated by a complex interactor with an Equivalence-attribute, marking both possibilities as equivalent. The confirmation of the recipe (*A2:Selection*) has only a graphical presentation (*Button*) which is directly mapped to the abstract interactor.

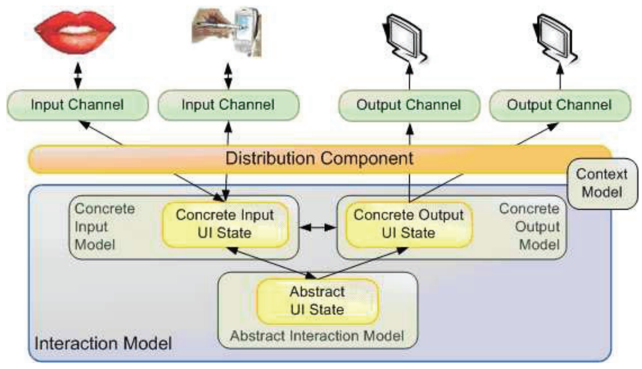

Figure 3: Overview of the runtime architecture.

Beneath the used possibilities, the concrete output model supports *SignalOutput*-elements to include more limited modalities like blinking lights or haptic feedback (vibration) and *DynamicOutput* to create multiple output, e.g. for a dynamically created number of elements.

To confirm the recipe, the developer provides three possibilities (*Gesture, Speech* and *Pointing*), which are aggregated by a complex interactor with an Equivalence-attribute, defining that they all can be used to provide the same input to the system (Figure 2). The next section describes how the modeled description is used within a runtime architecture to deliver flexible multimodal interaction.

RUNTIME DISTRIBUTION

Based on the needed components and the requirements that need to be fulfilled to realize the anticipated dynamic distribution at runtime, we developed and implemented the runtime architecture depicted in Figure 3. The different components and their behavior are described next.

On the execution of the set of models, the central task model calculates a set of active tasks. This triggers the mappings that are connected with each task and results in the activation of the mapped abstract interactors. The mappings connected to the abstract interactors are in turn triggered and the result is a set of active complex CUI elements in the concrete input and output model provided to the distribution component.

The second information provider is a context model that includes different observers to get information about the available IRs (requirement 3). IRs are connected to our runtime system via so called channels [3]. One channel is responsible for establishing and maintaining the connection to one IR (if needed via a network). This includes not only the registration within the context model but also the capability to receive and send information to the IRs. This concept allows the independent addressing of the IRs over platform borders (requirement 4).

We have implemented different channels which connect various interaction technologies, including browsers for graphical output through an AJAX-based channel implementation or connect automatic speech recognition via Dragon Natural Speaking and Text-To-Speech engines by implementing the Media Resource Control Protocol (MRCP). All models (user interface and context) are implemented with the Eclipse Modelling Framework (EMF). The direct mapping of EMF to Java supports the bridging of model and the distribution component on the implementation level which allows the distribution component to observe the models for state changes.

The distribution component is notified whenever a new set of concrete interactors is activated, and matches the supported modalities to the available modalities of the available IRs by adhering to the following goals:

Input: support as many (equivalent) input resources as possible while considering the specified CARE relations between the input elements. This aims at leaving the control about the used IRs to the user by supporting a wide range.

Output: find the most suitable combination of output resources while considering the specified CARE relations between the output elements. Distributed output thus aims at utilizing the most suitable combination of IRs to convey the UI. The selection of IRs depends on their capabilities and context information like the resource location.

The algorithm first determines the IRs that can be utilized by the user. Therefore the available IRs are queried from the context model together with information about the premises and localization and direction information. Based on the type of the IRs, the algorithm calculates if the resources are currently usable. E.g. displays are considered usable when they are within the vicinity of a user and haptical input IRs are considered usable when they are within the range of the user. The resulting set of usable IRs determines the usable modalities and thus the types of concrete interactors that can be distributed.

In the next step the algorithm analyzes the CARE relations of the active concrete interactors. The specified UI model contains trees of complex interaction elements with simple elements as leaf nodes. As only the leaf nodes have to be distributed, the relations defined by their parent complex interactors influence their distribution. The simple interactors are automatically of type "assigned" and can thus be directly distributed if a corresponding type of IR is available. Interactors combined via complex elements of type complementary or redundant must be distributed together to reflect their meaning. This means that to make an interaction, defined as redundant, available to the user, all modalities addressed by the childs of the complex interactor have to be available. The equivalence relation is used to specify different (combinations of) interactors that transport the same information in case of output or allow the user to provide the same information in case of input. This makes the system more reliable and reduces ambiguity and inconsistency. With respect to the distribution goals specified above, a different handling of the equivalency relation for input and output has been realized. For input the distribution of as many equivalent interactors as possible results in more possibilities for the user to provide the needed input. For output a selection of the most feasible interactors avoids confusion and unwanted redundancy.

Based on these interpretations of the CARE relationships the algorithm first calculates the distribution of the output interactors. The algorithm decides between the different equivalent interactor combinations by selecting the one supporting the most modalities. This is based on the assumption that the designer utilizes the advantages of each modality, so that more modalities result in a better presentation. More sophisticated extensions that consider additional context information are currently evaluated. Afterwards, the distribution of input interactors is calculated. Thereby the algorithm distributes all elements that are supported by the usable IRs to allow as many input possibilities as possible. It is crucial that during the distribution of the input interactors the algorithm pays attention to coupled input and output as e.g. in case of a touchscreen. The resulting distribution configuration consists of tuples of concrete interactors and IR references. Before sending the interactors to the channels, the presentation of the output interactors is accomplished by a layouting algorithm [11] which takes into account the spatial and temporal relationships of the interactors as well as the workflow model to not scatter related interactors.

In case of the little example, the algorithm would distribute the interactors as follows: For input the algorithm tries to support as many IRs as possible and thus determines at maximum the gesture, voice input and pointing interactors to support a keyboard, a microphone and a mouse respectively. For output a screen is required and an optional loudspeaker would be integrated if available. The algorithm would adapt the distribution accordingly, when e.g. the user is changing the position and the distribution component determines that some IRs are no longer available to the user and others just became available.

CONCLUSION

We presented an approach for dynamically selecting the IRs at runtime. Based on the modeled modality relations defined as CARE properties, which are available at runtime due to the utilization of executable models, and information about the actual context, a distribution algorithm calculates the most appropriate set of IRs.

In the future we plan to develop a multimodal widget set to ease the development of such multimodal and distributable UIs. We also want to analyze further factors that influence the distribution algorithm. Furthermore, automatic calculation raises the problem of unsatisfactory results. To overcome this issue for distribution of UIs, we started the development of a meta user interface allowing users to configure the distribution according to their needs.

REFERENCES

1. N. Barralon, J. Coutaz, and C. Lachenal. Coupling interaction resources and technical support. In *HCI International 2007*, Volume 4555 of LNCS, pages 13-22, 2007.
2. S. Berti, F. Correani, G. Mori, F. Paternò, and C. Santoro. Teresa: A transformation-based environment for designing and developing multi-device interfaces. In *CHI 2004*, volume II, pages 793-794, 2004.
3. M. Blumendorf, S. Feuerstack, and S. Albayrak. Multimodal user interaction in smart environments: Delivering distributed user interfaces. In *Constructing Ambient Intelligence*, AmI 2007 Workshops Darmstadt.
4. M. Blumendorf, G. Lehmann, S. Feuerstack, and S. Albayrak. Executable models for human-computer interaction. In *Proc. of the DSV-IS Workshop 2008*, pages 238-251, Berlin, Heidelberg, 2008.
5. J. Bouchet, L. Nigay, and T. Ganille. Icare software components for rapidly developing multimodal interfaces. In *Proc. of ICMI 2004*, pages 251-258, New York, USA, 2004.
6. G. Calvary, J. Coutaz, D. Thevenin, Q. Limbourg, Laurent Bouillon, and Jean Vanderdonckt. A unifying reference framework for multi-target user interfaces. In *Interacting with Computers*, 15(3):289-308, 2003.
7. J. Coutaz, L. Nigay, D. Salber, A. Blandford, J. May, and R. M. Young. Four easy pieces for assessing the usability of multimodal interaction: The care properties. In *INTERACT 1995*, pages 115-120, 1995.
8. A. Demeure, J.-S. Sottet, G. Calvary, J. Coutaz, V. Ganneau and J. Vanderdonkt. The 4c reference model for distributed user interfaces. In ICAS 2008. IEEE Computer Society Press.
9. C. Elting and M. Hellenschmidt. Strategies for self-organization and multimodal output coordination in distributed device environments. In *Workshop on Artificial Intelligence in Mobile Systems 2004*.
10. S. Feuerstack, M. Blumendorf, and S. Albayrak. Prototyping of multimodal interactions for smart environments based on task models. In *Constructing Ambient Intelligence*, AmI 2007 Workshops.
11. Q. Limbourg, J. Vanderdonckt, B. Michotte, L. Bouillon and V. López-Jaquero. Usixml: A language supporting multi-path development of user interfaces. In *EHCI/DS-VIS*, Volume 3425 of LNCS, pages 200-220. 2004.
12. J.P. Molina, J. Vanderdonckt, P. González, A. Fernández-Caballero and M.D. Lozano. Rapid prototyping of distributed user interfaces. In Proc. of CADUI'2006, pages 151-166. Springer-Verlag, 2006.