Runtime Models for Ubiquitous User Interfaces: Statement of Interest for the W3C Workshop on Future Standards for Model-Based User Interfaces

Grzegorz Lehmann
DAI-Labor, TU-Berlin
Ernst-Reuter-Platz 7, D-10587 Berlin, Germany
Grzegorz.Lehmann@DAI-Labor.de

INTRODUCTION

The developments in computer technology in the last decade change the ways we utilize computers. The availability of a variety of multimedia and interaction devices, sensors and appliances makes the development of user interfaces for smart environments a challenging and time-consuming task. Using standard modeling languages and technologies, like e.g. HTML, it is impossible to create context-adaptive, multimodal, multi-user and multi-device user interfaces that assist the user in a smart environment. The high heterogeneity and dynamics of smart environments require a new type of modeling languages that would enable the creation of truly ubiquitous applications.

Applications in ubiquitous environments are required to adapt dynamically to context of use situations unforeseeable at design time. Ubiquitous user interfaces [1] must not only adapt to multiple situations, but also address multiple users, devices and modalities. UIs generated from models at design time fail to provide the required flexibility, as the design rationale is no longer available at runtime. To tackle this issue the use of user interface models at runtime has been proposed [8]. Making the UI models available at runtime enables the reasoning about the decisions of the designer when she is no longer available. Additionally, the shift of focus from design time to runtime is supported by the demands for end-user development and personalization of applications. These features require abstractions from the code-level details of the applications at runtime.

Although the idea of utilizing UI models at runtime is not new, we still lack a common understanding and suitable methodologies for the definition of runtime models. Moving the models from design time to runtime raises questions about the connection of the models to the runtime architecture, about synchronization and valid modifications of the models at runtime or the identification of model parts specified by the designer and those determined at runtime.

The issues of runtime utilization of user interface models are in the focus of the human-computer interaction research group at the DAI-Labor of the Technische Universität Berlin. The next section briefly describes interesting issues of user interface models and languages that we would like to discuss at the workshop.

ISSUES OF INTEREST

We see two groups of issues emerging from the utilization of user interface models at runtime. First group is related to the basic features of the modeling languages. In order to adapt applications at runtime we need models that capture the information about the context, the application and its user interface at runtime. Raising questions are e.g.: how can the runtime information flow into the models so the models are up-to-date? How can the design time rationale held in the models and defined by the designer be made available and modifiable at runtime? How can we make explicit which parts of a model are modifiable at runtime and which not? How can we describe possible model modifications and make them executable at runtime? How does that impact the design of UI modeling languages?

The second group of issues relates to the design of user interface modeling languages that support the adaptation to smart environments at runtime. What does the designer define in the models, if the ubiquitous UI has no fixed size, no fixed interaction device, no fixed user and is not executed in a fixed context? Which parts of the application can or must be automatically adapted by the runtime architecture? Which adaptations of the application are defined by the designer? How can the designer define the boundaries of automated adaptations for her application? What impacts do such automated adaptations have on the user experience?

MULTI-ACCESS SERVICE PLATFORM

The work of the HCI group at the DAI-Labor deals with the above issues: we follow a model-based approach, which provides the designer with a language for describing the desired user interface and its multimodal variants. Additionally the designer is empowered to define constraints regarding the adaptation of the user interface at runtime. The user interface models of the designed applications are executed in a runtime architecture – the Multi-Access Service Platform (MASP). The MASP is deployed in a smart environment and gathers context information about it in runtime context models. The context models gather information about environment (e.g. its users, interaction resources, appliances). Depending on the current situation described in the context models, the MASP automatically adapts the user interfaces of the applications (e.g. by distributing the UI to the interaction resources near the user and adequately adapting the size of the graphical elements).

Just as the issues introduced in the previous section, our work with the MASP can be split in two areas. First, we are working on a basis for the definition of user interface modeling languages that address the needs of ubiquitous user interfaces. The languages should enable the creation of models that capture the state of the user interface and the smart environment at runtime and are adaptable at runtime. For this purpose we have designed and implemented a meta-metamodel that provides the basic building blocks for meta-modeling ubiquitous user interfaces.

The second area of our work is the design of a user interface modeling language using the meta-metamodel. On the basis of the building blocks provided by the meta-metamodel we have created a set of user interface metamodels. The runtime concepts explicitly incorporated within the metamodels enable us to utilize the conforming models at runtime – execute them, inspect their state and reconfigure them.

The two following sections describe both areas of our work. The first provides details about the meta-metamodel, the second briefly summarizes our work on the user interface metamodels.

Meta-Metamodel - Basis of MASP's Modeling Language

In order to define metamodels of runtime user interface models we need a meta-modeling language that provides means for the expression of runtime concepts within the metamodels. Meta-modeling languages are defined in form of special metamodels, so called meta-metamodels. Below we present a meta-metamodel, which provides necessary constructs for formalizing metamodels of runtime user interface models.

Figure 1 shows an extended version of the conceptual meta-metamodel introduced in [3]. Each conforming metamodel defines *Types* composed of *Fields* and *ExecutableElements*. *Fields* represent relationships between types (often referred to as attributes, associations, references, etc.) and are classified as either *Definition-* or *SituationFields*. Model elements held in definition fields (definition element) are the design time elements of models, created by the application developer. Situation fields hold situation elements that store the state of the model as it is interpreted at runtime. The notion of definition and situation parts has been introduced in [4].

The *ExecutableElements* represent operations enabling the modification of model elements. Depending on whether the modifications influence the situation or the definition part of the model, *ExecutableElements* are refined as either *SituationModificationElements* (SME) or *DefinitionModificationElements* (DME). As explained in previous sections, the SMEs encapsulate the execution logic of the models conforming to the metamodel, whereas the DMEs represent possible model adaptations.

The execution logic of the model influences the state of the model held in the *SituationFields*. Therefore each SME defines, which *SituationFields* it modifies, using the *alters* association. Associating an SME with other SMEs by means of the *uses* association the meta-modeler expresses that the execution of the SME is composed of or includes the execution of the associated SMEs (as in the *start* and *finish* case of the task model example).

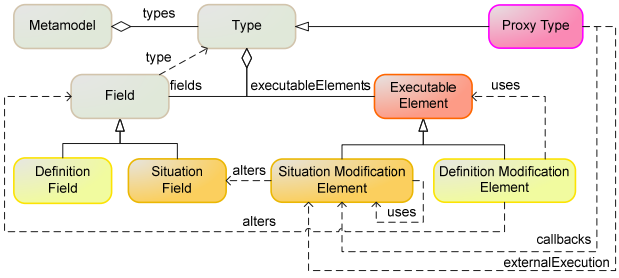

Figure 1: Meta-metamodel of runtime UI models

Performing an adaptation of the model may not only influence its definition part. It most definitely impacts its state as well. For this reason the DMEs can define *alters* and *uses* associations to both types of *Fields* and *ExecutionElements*.

Additionally, the meta-metamodel features a special *Proxy* type to enable the formalization of the causal connection of runtime models. The *Proxy* type classifies model elements connecting the model with the external world. The meta-modeling of the causal connection has been described in detail in [2].

The described constructs of our meta-metamodel are sufficient to distinguish the situation and definition parts of runtime models as well as to identify its execution and adaptation logic. In the next section we briefly describe the current state of our modeling language.

Metamodels of Ubiquitous User Interfaces

We are working on a modeling language for ubiquitous user interfaces, by extending a set of user interface metamodels compatible with the Cameleon reference framework [5] with runtime concepts of our meta-metamodel. The set of metamodels spans task, domain and service metamodels, abstract and concrete user interface metamodels, as well as a context and adaptation metamodel. Existing user interface modeling languages like UsiXML [6] and TERESA [7] are used as a basis. We enhance the metamodels with runtime concepts by making the runtime state information and the adaptation capabilities of the conforming models explicit.

Some of our most mature results comprise task and context metamodels. Both have been extended with runtime concepts specified in the meta-metamodel. The task metamodel has been extended with elements that describe the state of a task tree at runtime and its execution, as well as adaptation logic. *SituationModificationElements* encapsulate the execution logic of the conforming task models and special *DefinitionModificationElements* enable the adaptation of the models at runtime. For example, *removeChild* and *addChild* visualized in Figure 2 make it possible to add and remove tasks from a model.

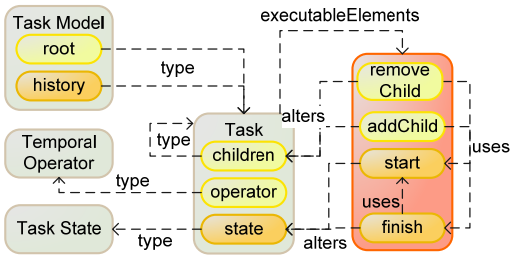

Figure 2: Runtime task metamodel

Modeling the context (the environment, the users and the platforms) is crucial for ubiquitous applications, which must dynamically adapt to changing situations. A runtime context model must provide means for reasoning about the current situation of the application and the user at runtime. Figure 3 pictures an excerpt of MASP's metamodel of runtime context models. It defines an *Environment* type composed of *Rooms* and *Users*. Users are located in rooms and have a 3D *position* (*SituationField*) in the environment expressed as *Vector*. Each user is associated with a *LocalizationTag* (e.g. a RFID tag). The metamodel furthermore specifies the necessary *SituationModificationElements*, e.g. the *setCoordinates* element that allows updating the position of a user at runtime.

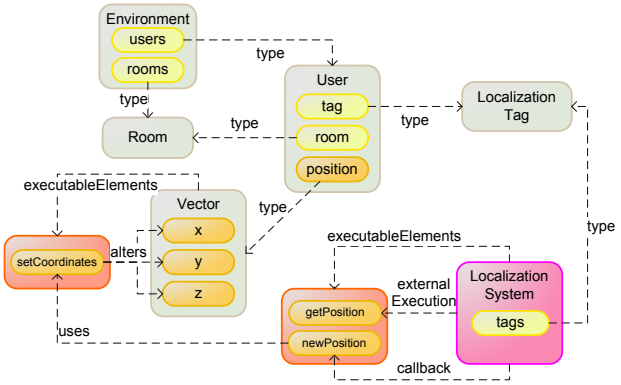

Figure 3: Metamodel of an executable context model.

The *LocalizationSystem* type represents a localization system in the environment. It defines the *newPosition* element for updating position information of users based on their localization tag and the data gathered from the localization sensors. Whenever a tag position change is detected, the *newPosition* element is triggered and uses the *setCoordinates* element to update the position of the user associated with the tag. In a similar way the other context data (e.g. information about available interaction resources) flows into the model, so the model provides an up-to-date view on the environment at runtime.

Reasoning about the runtime context model enables the detection of situations and a triggering of adaptations. Combining the runtime view of the environment provided by the context model with the reconfiguration capabilities of the runtime task models (e.g. *removeChild* and *addChild*) enables the implementation of interesting adaptation scenarios - a task tree of an application can be reconfigured based on the current position of the user in the environment. For example, in the kitchen the user may be able to perform additional "cooking" tasks that are not possible in the living room.

These simplified examples should show the potential of user interface model utilization at runtime. The adaptation scenarios implemented in the MASP span a broader range of models and feature more sophisticated behavior. In addition to the task and context models, the MASP modeling language also enables the definition of:

- abstract and concrete user interface models describing multimodal interactors of a MASP application
- constraint-based layout models, holding designer's statements about the size and position of the graphical user interface elements
- service models specifying the backend service calls of applications

Using the MASP modeling language we have implemented various applications running in a smart home testbed of the Technische Universität Berlin. I will be happy to present them and some of their adaptation capabilities at the workshop, as well as explain more details about our approach of runtime model utilization.

REFERENCES

[1] Marco Blumendorf. *Multimodal Interaction in Smart Environments A Model-based Runtime System for Ubiquitous User Interfaces*. PhD thesis, Technische Universität Berlin, 2009.

[2] Marco Blumendorf, Grzegorz Lehmann, and Sahin Albayrak. Bridging models and systems at runtime to build adaptive user interfaces. In *EICS '10: Proceedings of the 2nd ACM SIGCHI symposium on Engineering interactive computing systems*. ACM, 2010.

[3] Marco Blumendorf, Grzegorz Lehmann, Sebastian Feuerstack, and Sahin Albayrak. Executable models for human-computer interaction. In T. C. Nicholas Graham and Philippe Palanque, editors, *Interactive Systems. Design, Specification, and Verification: 15th International Workshop, DSV-IS 2008 Kingston, Canada, July 16-18, 2008 Revised Papers*, pages 238–251, Berlin, Heidelberg, 2008. Springer-Verlag.

[4] Erwan Breton and Jean Bézivin. Towards an understanding of model executability. In *FOIS '01: Proceedings of the international conference on Formal Ontology in Information Systems*, pages 70–80, New York, NY, USA, 2001. ACM.

[5] Gaëlle Calvary, Joëlle Coutaz, David Thevenin, Quentin Limbourg, Nathalie Souchon, Laurent Bouillon, Murielle Florins, and Jean Vanderdonckt. Plasticity of user interfaces: A revised reference framework. In *TAMODIA '02: Proceedings of the First International Workshop on Task Models and Diagrams for User Interface Design*,

pages 127–134. INFOREC Publishing House Bucharest, 2002.

[6] Quentin Limbourg, Jean Vanderdonckt, Benjamin Michotte, Laurent Bouillon, and Víctor López-Jaquero. Usixml: A language supporting multi-path development of user interfaces. In Rémi Bastide, Philippe A. Palanque, and Jörg Roth, editors, *EHCI/DS-VIS*, volume 3425 of *Lecture Notes in Computer Science*, pages 200–220. Springer, 2004.

[7] Giulio Mori, Fabio Paternò, and Carmen Santoro. Design and development of multidevice user interfaces through multiple logical descriptions. *IEEE Trans. Softw. Eng.*, 30(8):507–520, 2004.

[8] Jean-Sébastien Sottet, Gaëlle Calvary, and Jean-Marie Favre. Models at runtime for sustaining user interface plasticity. Presented at Models@run.time workshop (in conjunction with MoDELS/UML 2006 conference), 2006.