Distributed User Interfaces in Ambient Environment

Jean Vanderdonckt[1], Hildeberto Mendonca[1], and José Pascual Molina Massó[1,2]

[1] Belgian Laboratory of Computer-Human Interaction (BCHI)
Louvain School of Management (LSM), Université catholique de Louvain
Place des Doyens, 1 – B-1348 Louvain-la-Neuve (Belgium)
{vanderdonckt,mendonca,molina}@isys.ucl.ac.be
[2] Laboratory of User Interaction & Software Engineering (LOUISE)
Inst. de Investigación en Informática de Albacete (I3A), Universidad de Castilla-La Mancha
Campus universitario s/n – S-02071 Albacete (Spain)
jpmolina@dsi.uclm.es

Abstract. Before developing an Ambient Intelligence (AmI) application, it is often required to examine how its user interface will be distributed across the various interaction surfaces of its physical environment and which types of services these user interfaces will provide to end-users. For this purpose, a virtual reality rendering engine has been developed that renders a user interface model in a physical environment expressed by an environment model and a service model which enables designers to prototype the definition and the distribution of such user interfaces. After the modeling phase, the user interface is rendered in a virtual reality scene that may be subject to prototyping thanks to the model rendering engine. Any suggestion for modification applied on the underlying models (i.e., user interface, environment, and service) is propagated in the virtual reality scene. Depending on the interaction surface where a particular user interface is rendered, some adaptation rules may be performed expressed in terms of model-to-model transformations.

Keywords: context-aware adaptation, distributed user interfaces (DUI), environment model, intelligent user interfaces, interaction surface, physical environment, service model, user interface prototyping.

1 Introduction

A preliminary problem that arises when designing and developing Ambient Intelligence (AmI) applications, but not the only one, consists in determining how portions of its User Interface (UI) will be distributed in a physical environment (e.g., an augmented room, a home, a virtual laboratory) [5,6] and which shape this UI will take (i.e., digital, physical, or mixed) [9]. Once this distribution is more or less determined, it is the responsibility of designers to identify which interaction surfaces [4] will build the ambient environment and which capabilities they will be equipped with [7]. This lead to the notion of Distributed User Interfaces (DUIs), which is a particular type of UI whose portions are distributed and executed in time and space in a particular physical environment.

When the development cost of a DUI turns out to be too high for considering alternative designs, it is likely that this exploration will be abandoned soon or limited

M. Mühlhäuser et al. (Eds.): AmI 2007 Workshops, CCIS 11, pp. 121–130, 2008.

largely limited. A rapid and cheap prototyping technique may be then preferred. In addition, the usability issues raised by distributing a UI across one or several locations [7] are serious, although the benefits have been empirically demonstrated [11]: the display size has a positive effect of task completion time, tasks are better performed in multi-displays environment, to name a few.

Therefore, we believe that rapid prototyping of DUIs in AmI remains a key issue: not only rapid prototyping could be used as a vehicle for developing and demonstrating visions of innovative DUIs, but also they could help showing various distribution configurations before going to full implementation. However, rapid prototyping is also a challenging problem since the design space involves multiple dimensions: the physical environment itself, the DUI, the interaction surfaces, and the capabilities offered by these interaction surfaces. In order to address this problem more specifically, this paper provides a context model decomposed into three facets (i.e., user, computting platform, and physical environment). The environment is itself linked to interaction surfaces and their capabilities expressed as a service model.

The remainder of this paper is structured as follows: some related work is reported in Section 2, the models used in our approach are defined in Section 3. Section 4 will exemplify this model with some application in an augmented room. Section 5 will sum up the benefits of the models provided and some future avenues for this work.

2 Related Work

Some advanced researches share some similarities with the work described in this paper or provided initial material from which an extension has been introduced. Only the ones closest to this work are reported.

DYNAMO-AID [3] provides a distribution manager which distributes the sub-tasks of a task model to various computing platforms in the same physical environment, thus fostering a task-based approach for distributing UIs across locations of the physical environment. In contrast, the approach of this paper does not rely on a task model but rather gives the freedom and the responsibility to designers to distribute UIs in the environment by providing them with the appropriate mechanisms. In addition, there is no genuine model of interaction surfaces coupled with services described in a model.

The MIGRATION project [1] distributes UIs across web-based platforms of a cluster and provides a service-oriented approach for ensuring migration of these portions across the elements of the cluster. Again, this could be expanded with a model of interaction surfaces and services if more services are desired.

In Everywhere [10], DUIs could be rendered on various interaction surfaces such as large screens, white-boards, wall displays as well as personal surfaces. Only digital interfaces are considered as opposed to 3DSim [8], where only hardware interfaces are supported because it is the tool goal to prototype physical interfaces in a physical environment, thus also providing some sort of rapid prototyping. The material development of the physical UI is conducted after the prototype is validated.

VAQUITA [2] consists of a tool for reverse engineering an existing web page and redisplay it for another platform whose model has been previously defined. Per se, this tool is not a tool for supporting DUIs, but for retargeting a UI to another platform, according to the paradigm of multiple channel UIs [12,13,14].

In this paper, the environment model is produced as a virtual reality scene, thus allowing the rendering of both software (e.g., widgets) and hardware (e.g., physical buttons) objects, but this is achieved mainly for rapid prototyping purposes only. It is obvious that it cannot produce a physical UI, but the models used to prototype them could be passed to the development team afterwards. The tool provides basic operations such as copy a UI from one surface to another one, whether they belong to a computing platform or not, duplicate, and migrate.

The approach described in this paper is different from the above important pieces of work in that it explicitly relies on a physical environment model decomposed into surfaces, some of them being interactive some others not. These surfaces are then attached to a service model characterizing the capabilities they can offer to the UI rendering engine in virtual reality. The DUI could be physical, digital, or mixed.

3 Modeling Context of Use, Physical Environment, and Services

Figure 1 reproduces the meta-model of the various models exploited by the rendering engine and their relationships. The constituent models are then explained successively in the following respective sub-sections.

3.1 The Context of Use and the Physical Environment

The *Context of use* describes all the entities that may influence how the user's task is carried out with the UI [5]. It takes into account three relevant aspects, each aspect having its own associated attributes contained in a separate model: *user stereotype* (e.g., system experience, task experience, task motivation), *computing platform* (e.g., mobile platform, desktop, laptop, wall screen), and *physical environment* (e.g., office conditions such as lighting, level of noise). The physical context is here represented as a virtual reality scene, this is why we adopted a simplified representation inspired from X3D (see http://www.web3d.org/x3d/) where such as scene is composed of *surfaces* (i.e., any type of plane with its size, position, angle, etc.). This physical environment is populated by surfaces which could be connected together to form a topology of the scene (e.g., a floor, some walls and a ceiling to form an augmented lab), but also with *shapes* (i.e., any type of volume which could represent a scene object, such as a chair, a desk, cupboards). Defined as a sub-type of surface, the concept of *interaction surface*, firstly introduced in [4], is hereby referred to as any physical surface which can be "acted on or observed" so as to support user interaction with a system, whether visible or embedded. For instance, an interaction surface could be digital (e.g., a screen, a monitor, a wall display) or physical (e.g., a table equipped with camera tracking techniques, a pad with projection). The definition of physical (e.g., weight, size, material, solidity/fluidity/nebulosity) and modality attributes is in [4].

Each computing platform could be located precisely with respect to an environment surface and could hold none, one or many hardware platforms, which are declared as a general form of output (e.g., a display, a monitor, a screen). Each such platform is of course an interaction surface which could be acted on (by using pointers) and/or observed (by looking at the screen). Each interaction surface is defined by its shape, which is the area sensible to interaction. Hardware platforms are therefore considered as rectangular-shaped interaction surfaces. One could imagine probably other

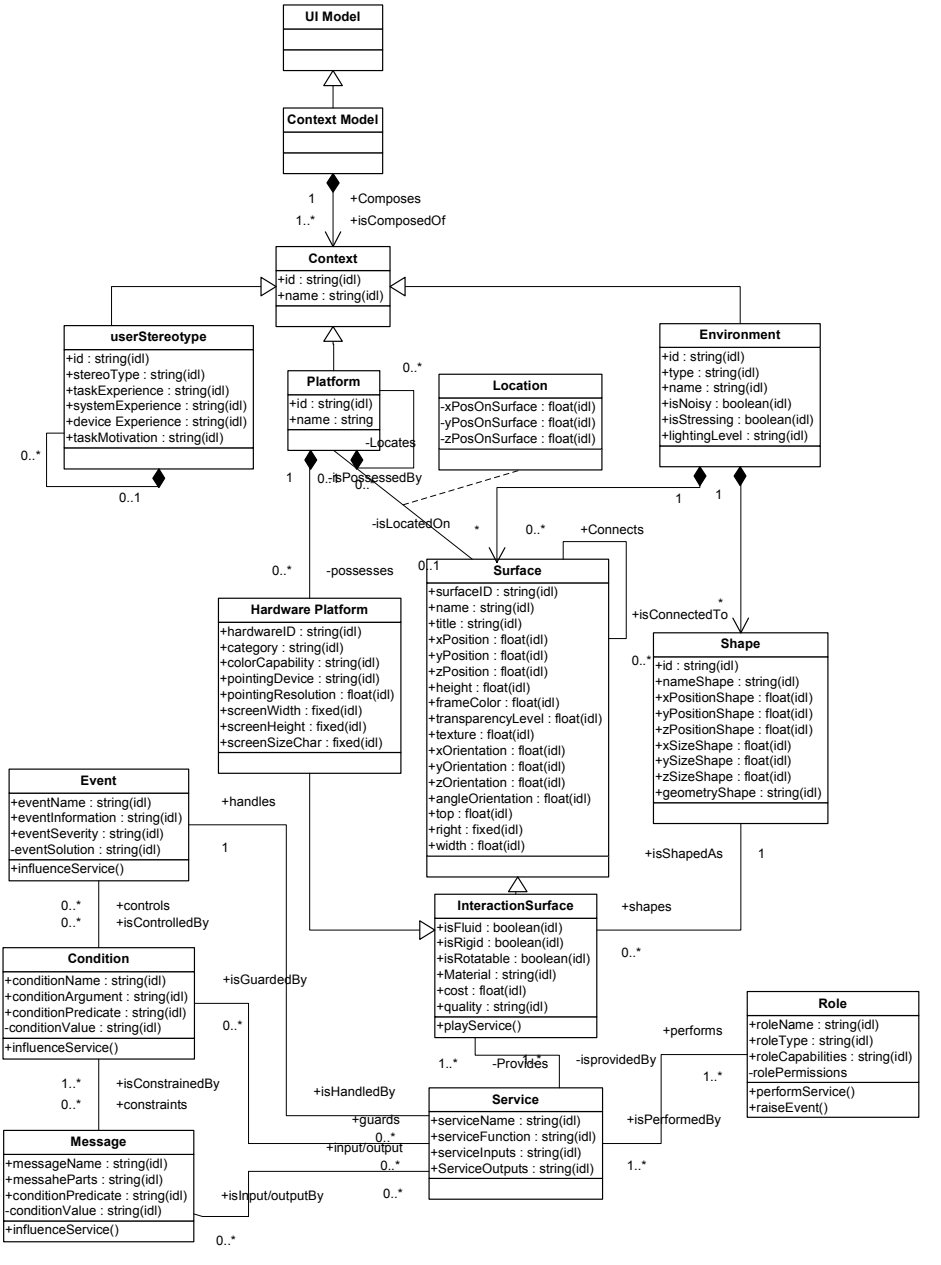

Fig. 1. The meta-model used for rendering a user interface in an ambient environment

shapes like a circle or an oval, but in our implementation, only convex surfaces are supported. The interaction surface type depicts the level of physicality of the surface used to render the DUI. A computer monitor and a public display are considered as digital surfaces, as opposed to projection surfaces that are considered as physical ones. This

distinction will impact the rendering of an object as a widget for a digital surface or as a physical device for a physical surface.

3.2 The Services

The service model is a meta-model comprising of five concepts useful for service modeling, calling, and composition inspired from the service model of [9]. This meta-model is based on generic service composition constructs derived after a thorough study of the current standards such as Business Process Execution Language (BPEL – http://en.wikipedia.org/wiki/BPEL) and Business Process Modeling Language (BP ML – www.bpmi.org). Based on this study and on requirements for DUIs [1], the following classes have been identified:

- *Service*: This abstract class represents a well-defined business function similar to basic activities such as those found in BPML. It contains four attributes: unique name, function, inputs, and outputs. An instance of this class can be defined as follows: Service: (name="cuiRendering", function="concreteUserInterfaceRendering", inputs="cuiModel, contextModel", outputs="cuiModel.instance"). This example shows a service named ”cuiRendering” that is meant for rendering a particular concrete UI. It requires several input parameters to carry out this task, such as, for instance, the UI model and the context model in which the UI should be rendered. The output parameter of this class includes the handle to the UI instance created in this context. Note that this service may exploit several sub-models such as the platform model to render properly the UI whose model is provided.
- *Event*: This abstract class describes occurrences during the process of service execution and composition. These can be both of a normal and exceptional nature (e.g., a serviced executed in case of an abnormal status). An instance of this class can be defined as follows: Event: (name="widgetRenderingError", context= "concreteUserInterfaceRendering", severity="unrecoverable", information="renderingStatus", solution="abandonExecution"). This example illustrates an event class called "widgetRenderingError". If the attribute "Severity" in this event class is set to "unrecoverable", then the execution needs to be abandoned. To signal the occurrence of "widgetRenderingError", a "widgetStatus" message must be sent to indicate that a particular widget cannot be rendered on a particular platform. For instance, the check box does not exist in WML 2.0, thus raising a rendering error. This is recoverable though and possible solutions could be indicated: an empty bow could be rendered instead or another service could be called, e.g. to replace the failing widget by a list box with the two opposite values.
- *Condition*: This class constrains the behavior of the service and possible compositions by controlling event occurrences, guarding activities and enforcing preconditions, post-conditions and integrity constraints. For this purpose, a condition class has four attributes, name, argument, predicate, value. A typical postcondition for "cuiRendering" could be that "a CUI has been rendered and could be manipulated by its handle". A typical precondition for "cuiRendering" could be that the platform model exhibits a resolution that is large enough to accommodate the rendering of a particular UI, i.e. not too large.
- *Message*: This abstract class represents a container of information (like properties in BPEL). Messages are used and generated by services as input and output, respectively. They are also used to signal events, and can be correlated to other

messages to express data dependencies. They have attributes such as name and parts.

- *Role*: This class provides an abstract description for a party participating in the service execution or composition. Roles are responsible for performing services and raising events. An instance of this class can be: Role: (name="renderingEngine", type="renderer", capabilities="(cuiRendering, cuiStoring, cuiRetrieving)", permissions="(cuiRenderingAllowed)"). The previous example describes "renderingEngine" as being of the type ”renderer”, both capable and authorized to render UI, to store a currently being rendered UI and to retrieve its status during execution. Note that the cuiRendering is executed prior to cuiStoring and cuiRetrieving.

To support the various services involved in Fig. 1, a rendering engine called VUI-Toolkit has been developed above the UsiXML (www.usixml.org) and expanded with the service model of Fig. 1 so as to render a concrete UI as a final UI in a virtual world. First, the environment model gives rise to a virtual world com-posed on surfaces, some of them being interactive. In particular, computing platforms could be located on some of these surfaces or considered as an interaction surface per se. Second, the toolkit abstracts objects belonging to Web3D languages (e.g., VRML, VRML97, X3D which are typically used in modeling virtual reality worlds and scenes). The next section exemplifies how this toolkit could be used to support the execution of an individual service, then the composition of several services taken together.

4 Executing, Composing Services for Distributed User Interfaces

4.1 User Interface Rendering

The *cuiRendering* service is responsible for rendering a UI or a portion of a UI on an interactive surface in the physical environment. For this purpose, the service is invoked with appropriate parameters such as the UI model, the environment model in which the rendering should occur and the interactive surface in it. Figure 2 reproduces a UI rendering of a simple Internet radio UI on a laptop in an augmented room.

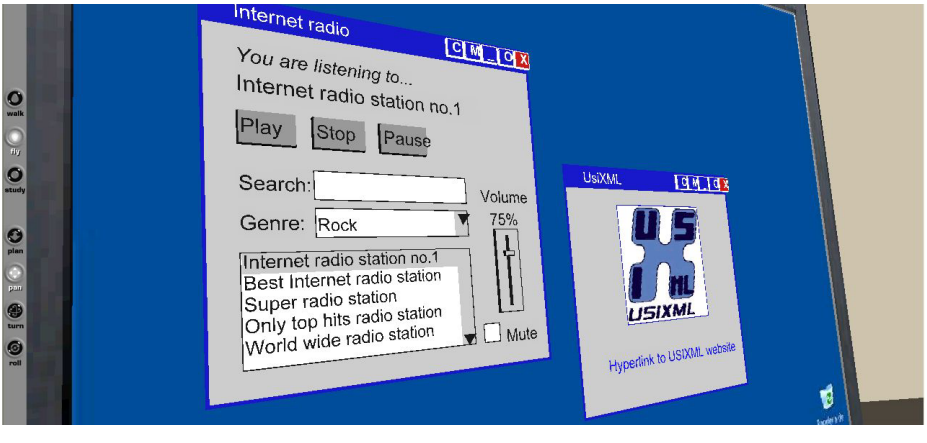

Fig. 2. Example of a user interface rendering

4.2 User Interface Migration

The *cuiMigration* service is responsible for migrating a UI or a portion of a UI from a source interactive surface in the physical environment to a target one. For this purpose, a composition of services is executed in the following sequence: *cuiStoring* for storing the current values of the UI widgets (e.g., their default values, their current values, the current behavior), *cuiUnrendering* to undisplay the UI from the interactive surface of concern, *cuiRetrieving* to retrieve an existing UI model instantiated through its handle, and *cuiRendering* to render it on another interactive surface. Figure 3 reproduces some steps during the migration. A message is first produced to indicate the beginning of a migration and when a new interactive surface is designated, an event triggers the composition of services on the appropriate surface. In the current implementation, single windows can be migrated as a portion of a UI, but not individual window elements.

Fig. 3. Example of a user interface migration

4.3 User Interface Adaptation

The *cuiMigration* service is responsible for adapting a UI or a portion of a UI to a particular platform. This service is invoked when the target interactive surface is not able to render the required UI due to space constraints.

4.4 Other User Interface Services

In AmI, other types of services are particularly useful such as, but not limited to:

- *uiPhysicalization*: this service is responsible for rendering a UI on a target interaction surface that is physical such as a projected screen or a physical device. For this purpose, different rendering functions select appropriate widgets or physical widgets depending on the type of interaction surface used.
- *uiDigitization* is the inverse process, that is a service responsible for converting a physical UI rendered on a physical interactive surface onto a digital one (screen).
- *uiDuplicating*: this service duplicates a UI rendered on an interactive surface onto another one, thus creating a duplicate. From this time, the two instances live their own life and are no longer coordinated.
- *uiCopying*: this service creates multiple copies of a singled rendered UI
- *uiSwitching*: this service is responsible for switching two UI between two interactive surface.

5 Conclusion

This paper introduced a service model for rendering DUIs in AmI as a support for rapid prototyping of DUIs by manipulation in a virtual reality scene. Some services are rather traditional (e.g., copy, duplicate, store, retrieve), some others are more advanced (e.g., adaptation, migration). The main advantage of the structure in terms of services is that functions can be precisely modeled and can be composed in a rigorous way, that is, a new service can be created as the composition of already existing services, thus creating a climate that is favorable for expanding the set of existing services. For example, when an interface is migrated from a PC to a PDA, it may simply be transferred thanks to the *cuiMigration* service and followed by a *cuiRendering* service or it may also trigger an adaptation service (e.g., *cuiAdaptation*). If one wishes to do this automatically, the three services can be composed together so as to form a new service.

The prototype can be obtained as soon as an underlying model is created. Then, it could be used in any software development method where the prototyping stage occurs. In the future, we would like to develop functions as web services so that everybody could benefit from them. In the virtual environment, one can explore alternative designs by calling services related to the desired operation. In this way, a designer may redistribute the DUI elements across the interactive surfaces of the environment and investigate different assembling or manifestation of the DUI elements as physical and/or digital objects. However, it is the manual operation what triggers the services, as there is no algorithm for distributing UI pieces on interaction surfaces. Besides,

even though transition between digital and physical worlds is easy in this virtual environment, rendering is currently limited by widgets provided by VUIToolkit.

The present work does not implement composition of services, neither dynamic discovery of services (as in [1,6,9]), but sets the pave towards this direction. There should be no reason why these services could not be invoked at run-time instead of design-time, but their implementation should be updated accordingly. Composition should be based on messages, but also on quality of services, e.g., find the closest surface that could render a given UI. Thus, further extension of this work could also include modeling quality of services provided by interaction surfaces, which may start from existing models, such as Q-WSDL, WSLA or DAML-QoS, to name a few.

Acknowledgments. We gratefully acknowledge the support of the SIMILAR network of excellence (http://www.similar.cc), the European research task force creating human-machine interfaces similar to human-human communication of the European Sixth Framework Programme (FP6-2002-IST1-507609). This research is fully funded by SIMILAR.

References

1. Bandelloni, R., Paterno, F., Salvador, Z.: Dynamic discovery and monitoring in migratory in-teractive services. In: Proc. of the 4th Annual IEEE Int. Conf. Pervasive Computing and Com-munications Workshops PERCOMW 2006, March 13-17, 2006, IEEE Computer Society Press, Los Alamitos (2006)
2. Bouillon, L., Vanderdonckt, J.: Retargeting Web Pages to other Computing Platforms with VAQUITA. In: Proc. of IEEE Working Conf. on Reverse Engineering WCRE 2002, October 28 -November 1, 2002, pp. 339–348. IEEE Computer Society Press, Los Alamitos (2002)
3. Clerckx, T., Vandervelpen, C., Luyten, K., Coninx, K.: A Task Driven User Interface Archi-tecture for Ambient Intelligent Environments. In: Proc. of 10th ACM Int. Conf. on Intelligent User Interfaces IUI 2006, Sydney, January 29 -February 1, 2006, pp. 309–311. ACM Press, New York (2006)
4. Coutaz, J., Lachenal, C., Dupuy-Chessa, S.: Ontology for Multi-surface Interaction. In: Proc. of 9th IFIP TC 13 Int. Conf. on Human-Computer Interaction INTERACT 2003, Zurich, September 1-5, 2003, pp. 447–454. IOS Press, Amsterdam (2003)
5. Dey, A.K., Salber, D., Abowd, G.D.: A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications. Human-Computer Interaction Journal 16(2-4), 97–166 (2001)
6. Ghorbel, M., Mokhtari, M., Renouard, S.: A Distributed Approach for Assistive Service Provi-sion in Pervasive Environment. In: Proc. of 4th Int. workshop on Wireless mobile applications and services on WLAN hotspots WMASH 2006, Los Angeles, September 29, 2006, pp. 91–100. ACM Press, New York (2006)
7. Giaglis, G.M., Kourouthanassis, P., Tsamakos, A.: Towards a Classification Framework for Mobile Location Services. In: Mennecke, B.E., Strader, T.J. (eds.) Mobile commerce: technology, theory, and applications, pp. 67–85. IGI Publishing, Hershey (2003)
8. Nazari Shirehjini, A.A., Klar, F., Kirste, T.: 3DSim: Rapid Prototyping Ambient Intelligence. In: Proc. of the 2005 Joint Conf. on Smart objects and ambient intelligence: innovative context-aware services: usages and technologies sOc-EUSAI 2005, Grenoble, October 2005. ACM Int. Conf. Proc. Series, vol. 121, pp. 303–307 (2005)

9. Orriëns, B., Yang, J., Papazoglou, M.P.: Model Driven Service Composition. In: Orlowska, M.E., Weerawarana, S., Papazoglou, M.P., Yang, J. (eds.) ICSOC 2003. LNCS, vol. 2910, pp. 75–90. Springer, Heidelberg (2003)
10. Pinhanez, C.: The Everywhere Displays Projector: A Device to Create Ubiquitous Graphical Interfaces. In: Abowd, G.D., Brumitt, B., Shafer, S. (eds.) UbiComp 2001. LNCS, vol. 2201, pp. 315–331. Springer, Heidelberg (2001)
11. Tan, D.S., Czerwinski, M.: Effects of Visual Separation and Physical Discontinuities when Dis-tributing Information across Multiple Displays. In: Proc. of 9th IFIP TC 13 Int. Conf. on Human-Computer Interaction INTERACT 2003, Zurich, September 1-5, 2003, pp. 9–16. IOS Press, Amsterdam (2003)
12. Vanderdonckt, J., Puerta, A.R.: Computer-Aided Design of User Interfaces II. In: Proc. of 3rd Int. Conf. of Computer-Aided Design of User Interfaces CADUI 1999, Louvain-la-Neuve, October 21-23, 1999. Information Systems Series, pp. 21–23. Kluwer Academics, Dordrecht (1999)
13. Wolff, A., Forbrig, P., Reichart, D.: Tool Support for Model-Based Generation of Advanced User Interfaces. In: Proc. of the MoDELS 2005 Workshop on Model Driven Development of Ad-vanced User Interfaces MDDAUI 2005, Montego Bay, CEUR Workshop Proceedings, vol. 159 (October 2, 2005)
14. Ziegert, T., Lauff, M., Heuser, L.: Device Independent Web Applications – The Author Once – Display Everywhere Approach. In: Koch, N., Fraternali, P., Wirsing, M. (eds.) ICWE 2004. LNCS, vol. 3140, pp. 244–255. Springer, Heidelberg (2004)