Towards an Integrated Development Environment for Context-Aware User Interfaces

Tim Clerckx and Karin Coninx

Hasselt University
Expertise Centre for Digital Media
Wetenschapspark 2
B-3590 Diepenbeek (Belgium)
{tim.clerckx,karin.coninx}@uhasselt.be
http://www.edm.uhasselt.be/

Abstract. The emergence of mobile computing devices brings along the fact that users interact with computers in various environments. The user interface of a mobile system can be affected by environmental context. Several approaches succeed in providing architectures and frameworks to support the building and reuse of software components considering context information. Taking into account context information in designing the interaction of a system, however, has not yet been extensively investigated. In this paper we will discuss an Integrated Development Environment, DynaMo-AID, we are developing to support the design, prototyping, evaluation and deployment of context-aware interactive systems.

1 Introduction

Nowadays, people make common use of mobile devices. This brings on the variation of *context* surrounding the user and his/her current device, in contrast to the *static* environment of the traditional desktop computer. Accordingly, the user will avail oneself of several distinct platforms, for instance a desktop computer at work, a notebook while attending a meeting, a PDA at the airport and a cell phone while driving a car. Since users perform the same tasks on different platforms and consequently want to use the same applications to perform these tasks (for example using the same mail client on each device), it is of vital importance that user interfaces are able to migrate to other platforms and smoothly adapt to the current context of use. In this research we define context as the information that can be gathered from the environment (including platform, user preferences, physical environment...) which can influence the tasks the user wants to, can or may perform.

Much research effort has been spent on exploring new ways to design user interfaces that are suitable for multiple distinct target devices [17]. The target platform of the application, however, is merely one aspect of the broader term *context-awareness* of an interactive system. The problem we want to tackle in this research is the fact that context as a whole should be considered in collaboration with the development of interactive systems because context can influence the tasks the user wants to, can or may

Dagstuhl Seminar Proceedings 05181
Mobile Computing and Ambient Intelligence: The Challenge of Multimedia
http://drops.dagstuhl.de/opus/volltexte/2005/369

perform. This is why we try to combine an abstract approach of designing user interfaces (i.e. using declarative models) with the modeling of context. To ensure usability of the interactive system, designers should be provided with a prototype reflecting the changes context can apply on the user interface. Furthermore designers must be able to adjust the declarative models in order to obtain a usable interface corresponding to the postulated requirements described by the models.

In this paper we will discuss the work that has been performed to provide user interface designers with tool support in order to design, test, and evaluate context-aware user interfaces. Designers can specify a user interface by specifying several models which describe all aspects of the user interface (tasks, navigation, presentation...) including a context model describing what kind of context information can influence the user interface. When the designer has specified these models, a prototype can be generated in order to test the user interface, and to explore how changing the context affects the user interface. Considering the prototype, changes can be applied to the models in order to adjust the tasks, provide more presentable presentation components, and link the user interface models to the functional core of the system

The remainder of this paper is structured as follows. We will discuss relevant related work in section 2. In section 3 we will postulate the goals of our approach. In section 4 we will discuss our development process. We will elaborate on current results and further research directions in section 5. Finally conclusions are given in section 6.

2 Related Work

Writing software that can adapt to the context of use is an important component of present-day applications. Support for these types of application is still a major topic for research. The Context Toolkit [19, 9, 8] is probably the best known way of abstracting context and making it "generally" usable. A clear division of components that embed, aggregate and interpret context information is envisaged. A method and tool support are discussed in [10] to give end users the opportunity to develop context-aware applications. However, less emphasis is placed on the effects and use of context on and in the user interface; instead they are targeting end users for describing and testing context-aware behavior of applications.

Mori et al. [16] present a process to design device-independent user interfaces in a model-based approach. In this approach, a high-level task model is constructed to describe tasks that can be performed on several platforms. Afterwards, the designer has to specify which tasks of the high-level description can be performed on which device. When this is done, an abstract UI will be created followed by the user interface generation. In our approach we describe the differences between target platforms in one complete task model and provide the possibility to take into account other sorts of context information than platform.

Calvary et al. [2] describe a development process to create context-sensitive user interfaces. The development process consists of four steps: creation of a task-oriented specification, creation of the abstract interface, creation of the concrete interface, and finally the creation of the context-sensitive interactive system. The focus however, lays upon a mechanism for context detection and how context information can be used to

adapt the UI, captured in a three-step process: (1) recognizing the current situation (2) calculating the reaction and (3) executing the reaction. In our approach we will focus on the exposure of a complete design process using extended versions of existing models, and how context reflects on these models. Furthermore we extend context by taking into account the effects of incoming and abolished services.

Limbourg et al. [14] describe a language, UsiXML, to describe context-aware user interfaces. Provided tool support, however, concentrates on transformations between models in order to transform abstract descriptions to concrete ones, and to reverse engineer concrete user interfaces. In our work we focus on incorporating context in user interface development. We use transformations between models [5] to assist in user interface design but the integration with the context model is done by the designer to avoid unexpected changes of the user interface when a context change occurs.

3 Developing Context-Aware Interactive Systems

In this section we want to outline the goals we want to achief in our approach. To create a development process to design and test interactive context-aware systems, the following aspects are considered to be important:

Abstract approach the influence of context on the application should be considered in early design stages. Dey stated in his doctoral dissertation [1] that a system is context-aware *if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task*. This is why attention should be devoted at the design stage when considering the user's tasks. Therefore, context should be approachable by system designers while drawing up abstract models (for instance a task model) in system design.

Incorporating support for different kinds of contex not only platform should be considered as a factor of context influencing applications. Other information can also be important [11, 13]: context data interpreted from rough sensor data, user preferences, location, network bandwidth...

Split different aspects of application design distinct parts of the application are designed by different people: user interface (interaction experts, graphical designers...), context acquisition and interpretation (hardware experts, AI specialists...), application core (software engineers, programmers). This is why these aspects have to be developed separately to make it possible to realize these three parts in parallel in order to speed up the development process.

Intuitive tool support a design process stays or falls with the support of intuitive tools allowing designers to analyze and model their ideas with regard to the desired system. Furthermore it should be possible to check the postulated requirements like checking scenarios (verification) and to support the generation of prototypes (validation concerning usability requirements).

Support for Services because users carry a mobile device while moving around, the device will be able to communicate with the changing environments, i.e. wireless networks, devices through Bluetooth connection... For instance when a user walks past a building and wishes to know more about the building. Suppose the device

can connect to a subnet of the wireless network of the building to download information about this building. This is what we call a service provided by the building to the user carrying the mobile device. In interface design, applications should be prepared to use such services, especially when user interaction with the service is possible.

In this research, we focus on designing the interactive part of a context-aware system. To maintain an abstract approach, the interaction will have to be specified by declarative models including a model representing context.

An important technical issue is usability of context-aware applications. Because abstract models are used and the user interface will react on context changes, it will be difficult to keep track of the usability of the system. This is why rapid prototyping should be supported. When the designer has defined some abstract models and has specified where the interaction can be influenced by context, a prototype should be generated by a tool to test the resulting user interface. When the designer has tested the prototyping, he/she can apply changes on the models, including the models representing context.

4 The DynaMo-AID approach

We have formulated a process (DynaMo-AID: Dynamic Model-Based User Interface Development [3]) to design the interactive part of context-aware applications and designed a supporting runtime architecture, which uses the artifacts generated by the design process.

Figure 1 shows a general overview of the development process. The DynaMo-AID design process supports the design of declarative abstract models, describing the context-aware user interface. The aggregate of the models, the interface model, can be serialized in order to export these models to a runtime architecture. To test the result of these models, the corresponding user interface can be generated in shape of a prototype to check the usability of the system. Considering the prototype, some changes to the models in the design process can be applied to alter for instance the presentation of the user interface or how context changes may affect the user interface [5].

In the remainder of this section, we will discuss how the design process and the runtime environment are structured.

4.1 Designing the Interaction

Figure 2 provides an overview of the design process. The designer will have to specify several models to describe the interaction of a system from an abstract level:

Context-Sensitive Task Model First, a task model is specified describing the tasks user and application may encounter when interaction with the system is taking place. Because we want to develop context-aware user interfaces, tasks also depend on the current context-of-use. This is why we extended the ConcurTaskTree Notation [18] to specify tasks in different contexts-of-use. The notation is extended with an extra type of task, the decision task, describing which sub tasks (sub trees of

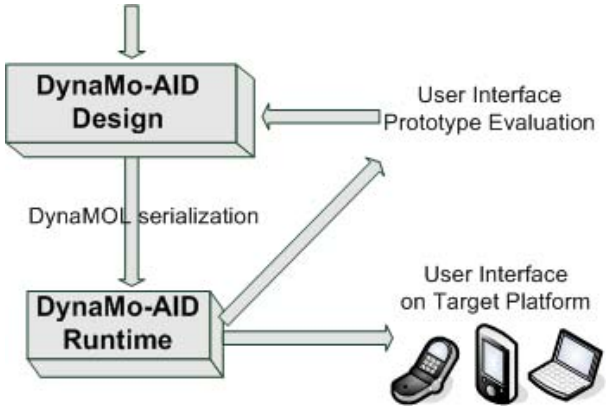

Fig. 1. DynaMo-AID Development Process

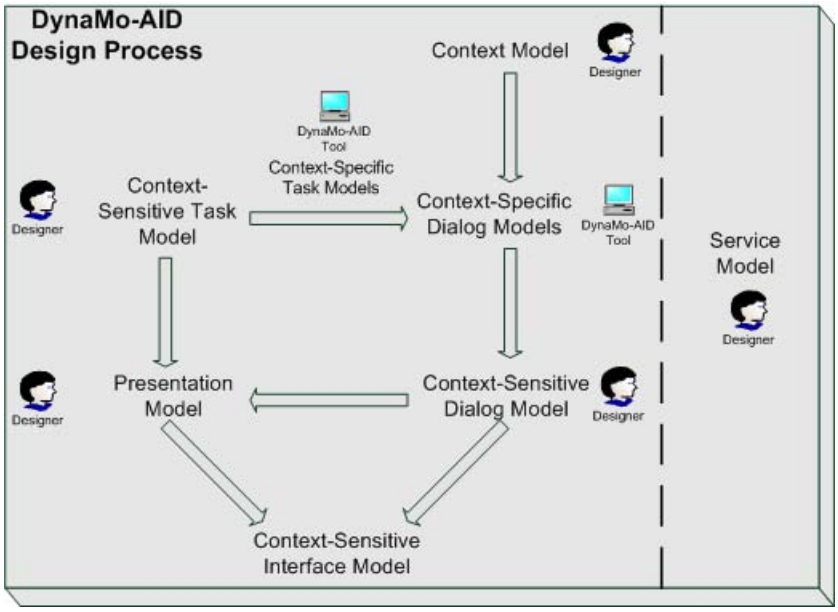

Fig. 2. DynaMo-AID Design Process

the decision task) are relevant in the current context of use. In this way the designer can describe different tasks for different contexts-of-use. Details on the extended ConcurTaskTree Notation are discussed in [4].

Context Model When the task model is specified, the designer has to denote what kind of context information can influence the interaction, i.e. the tasks. This can be done by selecting objects for context gathering (Concrete Context Objects or CCOs). These objects can be aggregated and interpreted by other objects (Abstract Context Objects or ACOs). The designer can do this by linking ACOs to CCOs and selecting from a set of predefined interpretation rules how the context information has to be interpreted. The ACOs represent abstracted context information. When the context model is specified, the designer has to link the ACOs to task model nodes (inter-model connection). In this way, the designer can denote which tasks can be performed in which context of use.

Context-Specific Dialog Models Next, tool support will automatically extract a dialog model from the task model [15] for each context of use. Afterwards, inter-model connections are added automatically between states of the dialog model and tasks of the task model that are enabled for each particular state.

Context-Sensitive Dialog Model The next step is to link dialog model nodes (states) of the different dialog models to denote between which states context changes may occur. We used this approach rather than automatically link similar states in distinct dialog models to ensure designers really want these context changes to affect the interaction taking place.

Presentation Model To provide the interface model with information how the interaction should be presented to the user, designers have to compose abstract user interface components, and link these to the relevant tasks for each presentation model node. The presentation model nodes can be structured hierarchically in order to group presentation components for layout purposes. The designer can choose from several abstract user interface components: static, input, choice, navigation control, hierarchy, and custom widget. Furthermore, a URI (Uniform Resource Identifier) widget can be chosen containing a URI to an external user interface component (for instance a UIML document on an HTTP server). Finally the user interface components can be grouped, and structured in a hierarchical structure.

Context-Sensitive Interface Model The aggregate of the previous models results in a context-sensitive interface model.

Service Model the interaction with the main application is described by the previous models. Services requiring user interaction also need to be described. This can be done independent of the design of the main application. Services are modeled by describing the same declarative models as the main application. When a service is deployed, the model of the service and the main application will be merged to present the interaction of both.

Because of the high amount of interconnections between the distinct models, the tool we have implemented to support the modeling phase is graphical with semantic zooming to maintain an overview of the interconnections even when the focus of interest lies on one particular model. Usability tests are planned in the near future to test and improve usability of the graphical interface with the models. Figure 3 shows the

tool providing an overview of the task, context, dialog and presentation model and the connections between the distinct models.

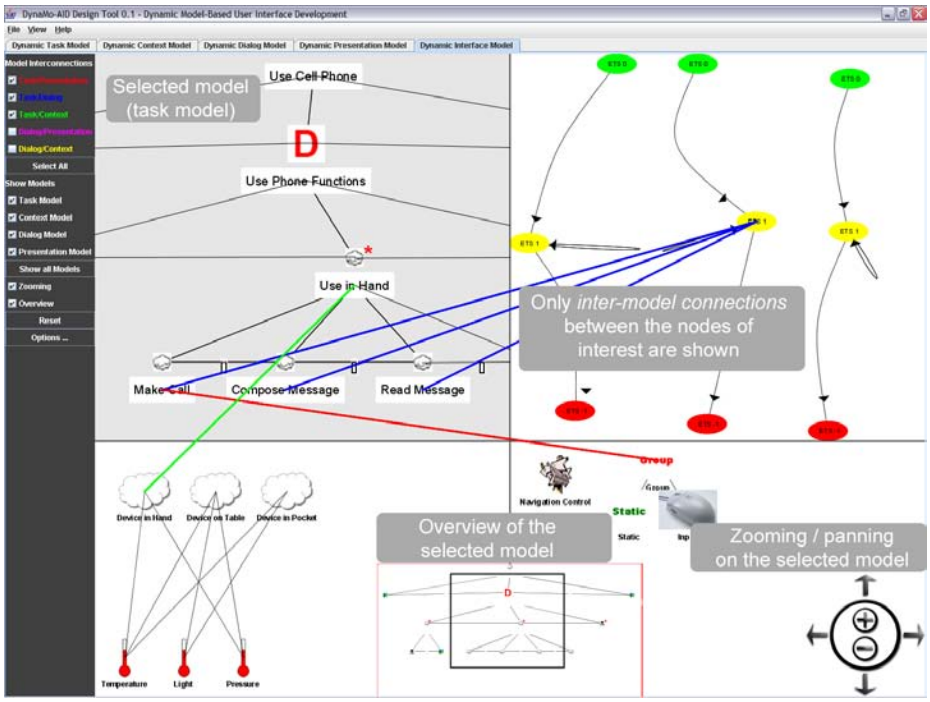

Fig. 3. DynaMo-AID Design Tool: overview of the declarative models and the interconnections

4.2 Runtime Architecture

The artifact generated by the design tool, i.e. an XML-based description of the interface model, can be used to deploy a user interface with the runtime architecture. The architecture takes care of communications between user interface, application core, and context data [6].

Figure 4 gives an overview of the runtime architecture. The architecture exists of 3 distinct components, communicating with each other through an asynchronous messaging system:

DynaMo-AID Runtime Server The server-side part of the architecture consists of 5 separate components. The *Context Control Unit (CCU)* will detect the current context, supplied by the abstract context objects and the dialog controller will be notified when a context change significant for the interaction has occurred. A *Dialog Controller* will handle communication between user interface on the target device,

the CCU, and the application core. The dialog controller decides when the user interface has to be changed, either by user interaction, a context change, or a message from the application core. *Abstract Context Objects (ACOs)* are connected with Concrete Context Objects (CCOs) which provide low-level context information. ACOs will aggregate and interpret the information of several CCOs in order to provide the CCU with abstract context information. A *Context Repository* registers context changes in order to take decisions about context information based on historical information. The *Application Core (AC)* is used to retrieve information to be presented in the user interface. The AC also sends messages when data is updated relevant for the user interface. The application core can also depend on context information from the CCU.

DynaMo-AID Runtime Client The Runtime Client runs on the device the user will be interacting with. The *Runtime Environment* enables communication with the Runtime Server. To present the user interface, several back ends can be used. Up to now we have tested with the Dygimes[7] renderer and an initial version of a .NET renderer.

DynaMo-AID Context Sensing Clients These are CCOs encapsulating some sort of context information and providing the information communication with ACOs for abstraction and interpretation. The CCOs can be located on the target mobile devices as well as anywhere in the user's environment.

We choose to put CCU and dialog controller on an external server because of several advantages:

- Relieve the mobile device from complex data and calculations as there are: interpreting the model components, retrieving, interpreting and using context information...
- registration of more than one entity (device, user interface instance) to the dialog controller:
  - sending events to the design tool: this can be used for monitoring the interaction with the prototype, visualize which model components are currently active, logging the interaction...
  - sending events to more than one interaction device: this can be used for the support of distributed user interfaces.
- asynchronous communication between the prototype and the design tool. In this way, the prototype can be rendered on a platform different from the platform and programming language of the design tool. A .NET prototype communicating with the design tool implemented in Java for instance.
- Keeping a repository of past context information, making it possible to base decisions on historical context information.
- Communicating with services using the same communication protocol used for the interaction and context information.

5 Future Work

Up to now, we have constructed the development process discussed in the previous section. We have implemented tool support to design the declarative models describing the

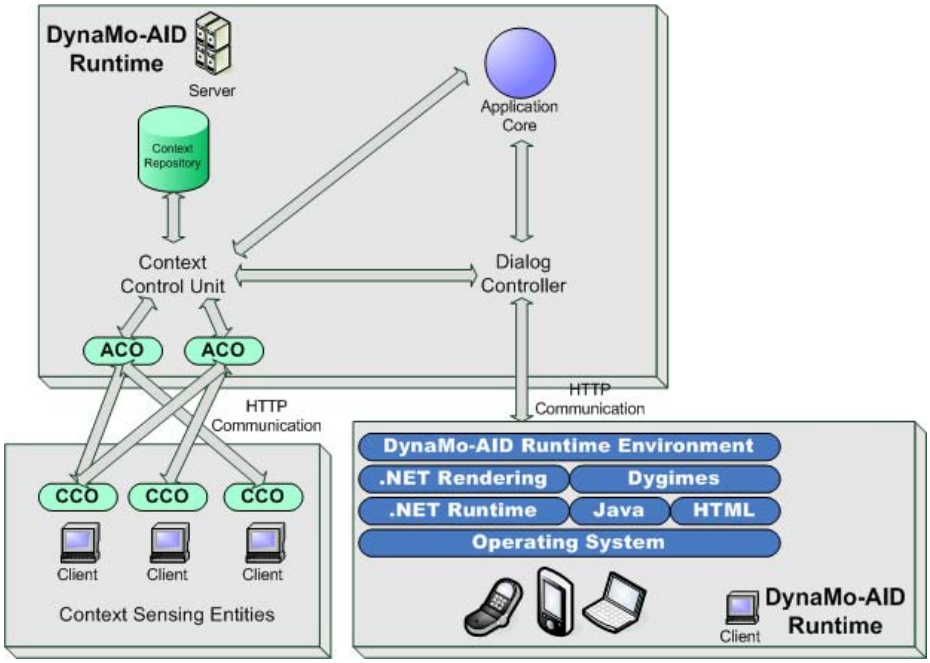

Fig. 4. DynaMo-AID Runtime Architecture

user interface. Furthermore we have implemented a proof-of-concept runtime architecture on the Dygimes-framework. In this implementation we made use of both simulated context and context information detected by hardware sensors to test the rendered prototypes. The communication between the Concrete Context Objects encapsulating context information and the server-side part of the context processing is currently implemented with a direct socket connection.

The remainder of this research will focus on:

- incorporating services in the design tool and the runtime architecture;
- improve the support for evaluation of the generated prototypes;
- applying changes to the models after evaluation such as correcting tasks, providing more presentable presentation components, and linking the user interface models to the functional core of the system;
- deploying the context-aware interactive system.

6 Conclusion

The core of this research is to incorporate the influence of context information on the user interface from early design stages. In this matter, the separation between the user interface, context data and the functional core of the application is taken into account. Up till now we have formulated a design process and added the necessary tool support to design, test, and evaluate context-aware user interfaces. Next step is to extend this approach to make it possible that context changes that are difficult to predict are incorporated in the design, in particular the appearance and disappearance of services.

7 Acknowledgements

The authors would like to thank Frederik Winters and Kris Luyten for their contributions to the work discussed in this paper. Part of the research at EDM is funded by EFRO (European Fund for Regional Development), the Flemish Government and the Flemish Interdisciplinary institute for Broadband technology (IBBT). The CoDAMoS (Context-Driven Adaptation of Mobile Services) project IWT 030320 is directly funded by the IWT (Flemish subsidy organization).

References

1. Anind Dey. *Providing Architectural Support for Building Context-Aware Applications*. Ph.D. thesis, December 2000, College of Computing, Georgia Institute of Technology, 2000.
2. Gaelle Calvary, Joëlle Coutaz, and David Thevenin. Supporting Context Changes for Plastic User Interfaces: A Process and a Mechanism. In *Joint Proceedings of HCI 2001 and IHM 2001. Lille, France*, pages 349–364, 2001.
3. Tim Clerckx, Kris Luyten, and Karin Coninx. DynaMo-AID: a Design Process and a Runtime Architecture for Dynamic Model-Based User Interface Development. In Kazman and Palanque [12], pages 142–160.

4. Tim Clerckx, Kris Luyten, and 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, 13-16 January 2004, Funchal, Isle of Madeira, Portugal*, 2004.
5. Tim Clerckx, Kris Luyten, and Karin Coninx. The Mapping Problem Back and Forth: Customizing Dynamic Models while Preserving Consistency. In Philippe Palanque, Pavel Slavik, and Marco Winckler, editors, *3rd International Workshop on Task Models and Diagrams for user interface design 2004 (TAMODIA 2004)*, pages 33–42, Prague, Czech Republic, Nov 15–16 2004.
6. Tim Clerckx, Kris Luyten, and Karin Coninx. Designing Interactive Systems in Context: From Prototype to Deployment. In *Accepted for the 19th British HCI Group Annual Conference (HCI 2005)*, Napier University, Edinburgh, United Kingdom, Sep 5–9 2005.
7. Karin Coninx, Kris Luyten, Chris Vandervelpen, Jan Van den Bergh, and Bert Creemers. Dygimes: Dynamically Generating Interfaces for Mobile Computing Devices and Embedded Systems. In Luca Chittaro, editor, *Mobile HCI*, volume 2795 of *Lecture Notes in Computer Science*, pages 256–270. Springer, 2003.
8. Anind K. Dey and Gregory D. Abowd. The context toolkit: Aiding the development of context-aware applications, 2000. Workshop on Software Engineering for Wearable and Pervasive Computing, June 6, 2000, Limerick, Ireland.
9. Anind K. Dey and Gregory D. Abowd. Support for the Adaptation and Interfaces to Context. In Ahmed Seffah and Homa Javahery, editors, *Multiple User Interfaces, Cross-Platform Applications and Context-Aware Interfaces*, pages 261–296. John Wiley and Sons, 2004.
10. Anind K. Dey, Raffay Hamid, Chris Beckmann, Ian Li, and Daniel Hsu. a CAPpella: Programming by Demonstration of Context-Aware Applications. In *Conference on Human Factors in Computing Systems (CHI 2004), Vienna, Austria, April 24-29*, 2004.
11. Ken Hinckley, Jeff Pierce, Mike Sinclair, and Eric Horvitz. Sensing Techniques for Mobile Interaction. In *The 13st Annual ACM Symposium on User Interface Software and Technology, UIST 2000, November 5-8, 2000, San Diego, CA, USA*, volume 2(2) of *CHI letters*. ACM Press, 2000.
12. Rick Kazman and Philippe Palanque, editors. *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, Tremsbüttel Castle, Hamburg, Germany, July 11-13, 2004, Pre-Proceedings*, 2004.
13. Panu Korpipää, Jani Mätyjärvi, Juha Kela, Heikki Keränen, and Esko-Juhani Malm. Managing context information in mobile devices. *IEEE Pervasive Computing, Mobile and Ubiquitous Systems*, 2(3):42–51, July-September 2003.
14. Quentin Limbourg, Jean Vanderdonckt, Benjamin Michotte, Laurent Bouillon, and Victor López-Jaquero. USIXML: a Language Supporting Multi-Path Development of User Interfaces. In Kazman and Palanque [12], pages 89–107.
15. Kris Luyten, Tim Clerckx, Karin Coninx, and Jean Vanderdonckt. Derivation of a Dialog Model from a Task Model by Activity Chain Extraction. In Joaquim A. Jorge, Nuno Jardim Nunes, and João Falcão e Cunha, editors, *Interactive Systems: Design, Specification, and Verification*, volume 2844 of *Lecture Notes in Computer Science*, pages 191–205. Springer, 2003.
16. Giulio Mori, Fabio Paternò, and Carmen Santoro. Tool Support for Designing Nomadic Applications. In *Proceedings of the 2003 International Conference on Intelligent User Interfaces, January 12-15, 2003, Miami, FL, USA*, pages 141–148, January 12–15 2003.
17. Giulio Mori, Fabio Paternò, and Carmen Santoro. Design and Development of Multidevice Interfaces through Multiple Logical Descriptions. *IEEE Transactions on Software Engineering*, 30(8), August 2004.

18. Fabio Paternò. *Model-Based Design and Evaluation of Interactive Applications*. Springer Verlag, ISBN: 1-85233-155-0, 1999.
19. Daniel Salber, Anind K. Dey, and Gregory D. Abowd. The Context Toolkit: Aiding the Development of Context-Enabled Applications. In *Proceedings of the 1999 Conference on Human Factors in Computing Systems (CHI '99), Pittsburgh, PA, May 15-20*, pages 434–441, 1999.