Chapter 1

GENERATING CONTEXT-SENSITIVE MULTIPLE DEVICE INTERFACES FROM DESIGN

Tim Clerckx, Kris Luyten, Karin Coninx
{tim.clerckx,kris.luyten,karin.coninx}@luc.ac.be
Limburgs Universitair Centrum
Expertise Centre for Digital Media
Universitaire Campus, B-3590 Diepenbeek, Belgium
http://www.edm.luc.ac.be

Abstract This paper shows a technique that allows adaptive user interfaces, spanning multiple devices, to be rendered from the task specification at runtime taking into account the context of use. The designer can specify a task model using the ConcurTaskTrees Notation and its context-dependent parts, and deploy the user interface immediately from the specification. By defining a set of context-rules in the design stage, the appropriate context-dependent parts of the task specification will be selected before the concrete interfaces will be rendered. The context will be resolved by the runtime environment and does not require any manual intervention. This way the same task specification can be deployed for several different contexts of use. Traditionally, a context-sensitive task specification only took into account a variable single deployment device. This paper extends this approach as it takes into account task specifications that can be executed by multiple co-operating devices.

Keywords: Model-Based user interface Design, Task Modeling, ConcurTaskTrees Notation, Context Sensitive, Multi Device

1. Introduction

Recent advances in mobile computing devices and mobile communication support more complex interaction between different devices. This allows users to migrate from their single "computer on the desk" setup to a heterogeneous environment where he/she uses several devices to accomplish his/her tasks. Although the provided hardware and software becomes more powerful, it makes designing the interface more complex. Different contexts (device constraints, environment of the mobile user,. . .) have to be taken into account. The nomadic nature of future applications also demands a way to design interaction using multiple devices.

Combining our previous work [9, 6] with context-sensitive task specifications [15, 16] we realize a supporting framework for the design and creation of context-sensitive multiple- and multi-device interaction. By multiple-device interaction we mean the user interface (UI) is distributed over different devices. The implementation has been tested as a component of the Dygimes framework [6].

The remainder of this paper is structured as follows: section 1.2 discusses the related work, introducing the state of the art in context-sensitive task modelling. To illustrate the context and testbed of this work, our framework Dygimes is introduced in section 1.3. This is followed by an overview of the design process needed to create a context-sensitive UI in section 1.4. Three stages are described: the creation of the task model, the extraction of the dialog model and the generated presentation model. This is followed by a case study to show how things work in practice. Finally, the obtained results and their applicability are discussed in the conclusion.

2. Related Work

Pribeanu et al. [15] proposed several possible approaches to adapt the ConcurTaskTrees notation [13] for context-sensitive task modeling. As pointed out in [15] and [16], the context of use of the application influences which parts of the task model are executed. A context-sensitive (or dependent) and a context-insensitive (or independent) part of the task model can be identified and processed accordingly. The context-sensitive part can be related to the context-insensitive part in multiple ways [15]:

- Both parts are specified in one task model: the *monolithic approach*

- The context-insensitive parts are connected to the context-sensitive parts with general arcs: *graph-oriented approach*

- The context-insensitive parts are connected to the context-sensitive parts with special arcs that can constitute a decision tree: *separation approach*

The last approach in particular is interesting: although it allows different parts for different contexts of use to be integrated in one model, there is a *decision tree* that provides a nice separation. We choose to insert *decision nodes* in the task specification instead of decision trees. Of course, decision nodes can have other decision nodes as descendants. The children of a decision node are possible subtrees where one of them will be chosen in a preprocessing step. Section 1.4 explains in detail how a concrete task specification can be obtained by preprocessing the decision nodes.

Paternò and Santoro [14] present a method to generate multiple interfaces for different contexts of use starting from one task model. The TERESA tool for supporting this approach is discussed in [11]. In contrast with their approach, we do not focus on the design aspect as much as they do, but emphasize the runtime framework necessary for accomplishing this. To our knowledge, the TERESA tool supports the creation of one task model for multiple devices, but currently does not take into account multiple devices interacting at once or the interface migrating from one device to another.

Calvary et al. [3, 4] describe a process where a *Platform* and *Environmental Model* are used to represent context information. The process allows to create UIs for two running systems in different contexts. Although at several stages in the UI design process (Task Specification, Abstract UI, Concrete UI, Runtime Environment) a translation can take place between the two systems, the designer will have to change the task specification manually in the process if the context has an influence on the tasks that can be performed.

Nichols et al. [12] defined a specification language and communication protocol to automatically generate UIs for remotely controlled appliances. The language describes the functionalities of the target appliance and contains enough information to render the UI. In this case, the context is secured by the target appliance represented by its definition.

Ali and Pérez-Quiñones [2] also use a task model, together with UIML [1], to generate UIs for multiple platforms. The task model has to increase the abstraction level of the UIML specification, which is necessary to guide the UI onto different devices.

3. Dygimes

Most of the presented work is integrated in our framework Dygimes [6]. Besides supporting the ConcurTaskTree task specification, it uses high level user interface Descriptions (specified in XML) to define the set of abstract interactors necessary for completing the tasks specified in the task specification. One of the aims of this framework is to support design through selected models from Model-Based User Interface Design, and add support for transforming the design into multi/multiple-device UIs at runtime.

The Dygimes framework supports roughly the following steps for creating UIs (a more detailed description can be found in [6]):

1 Create a context-sensitive task specification with the ConcurTask-Trees notation

2 Create UI building blocks for the separate tasks

3 Relate the UI building blocks with the tasks in the task specification

4 Define the layout using constraints

5 Define custom properties for the UI appearance (e.g. preferred colors, concrete interactors,. . .)

6 Generate a prototype and evaluate it (the dialog model and presentation model are calculated automatically)

7 Change the task specification and customizations until satisfied

On the one hand it supports a clear separation between the creation of the UIs and the implementation of the application logic that underlies the UI. On the other hand there is built-in support to connect the UIs with the application logic without manual intervention [18].

The next section will describe how the design process for the context-sensitive UI and the generation of the UI works.

4. Design Process

The proposed approach extends the process for automatically generating prototype UIs from annotated task models introduced in [9]. Figure 1.1 shows the extended process where a context-sensitive task model is considered to generate UIs depending on the context at the time the UI is rendered. First, a context-sensitive task model is constructed and high-level UI building blocks are attached to the leaves as described in the previous section. Next, the context is captured and the proper context-specific ConcurTaskTree will be generated automatically. Subsequently the Enabled Task Sets (ETSs) are calculated. These are sets of tasks that can be enabled at the same time [13] and therefore contain the proper information to be rendered together in the resulting UI.

After this step, the appropriate dialog model is extracted automatically from the task model using the temporal operators [9]. Each dialog still is related to the set of tasks it presents, thus also to the appropriate UI building blocks it can use to present itself. The context-sensitive information in the task specification is taken care of in a "preprocessing" step, which we will explain now into further detail.

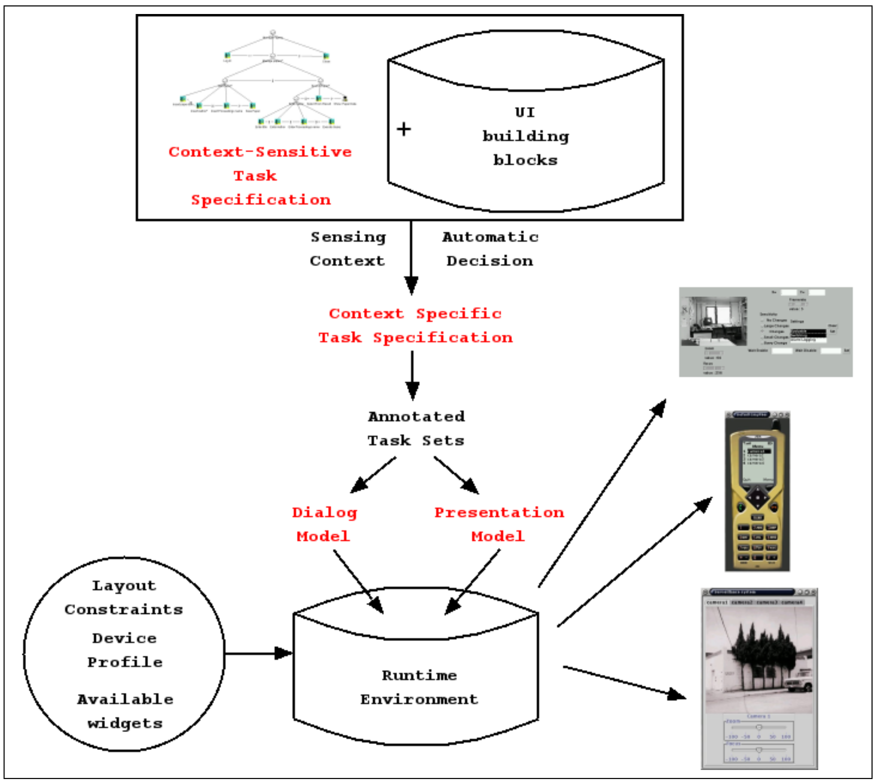

Figure 1.1. Context-Sensitive user interface Design Process

4.1 The Context-Sensitive Task Model

As pointed out in section 1.2, there are three proposed approaches to model context-sensitive task models. Instead of collecting decision trees, we propose another way where the context-insensitive part points directly to context-sensitive subtrees through *decision nodes*. These nodes are marked by the **D** in the example in figure 1.4. Although this resembles the graph-oriented approach, the context-sensitive subtrees are the direct children of the decision node. When the context-sensitive parts are resolved, the decision node will be removed and replaced by the root of the selected subtrees of that decision node.

The decision nodes are executed in the first stage of the UI generation process. This results in a normal ConcurTaskTree specification, but also one that is suitable according to the rules defined in the decision nodes. The normal ConcurTaskTree specification enables the provided algorithm to extract the dialog model automatically adapted to the current context.

In order to link the context detection and the task model, some information about which subtree has to be performed in which case is added to the decision node. Figure 1.2 shows a simple scheme (as a Document Type Definition) defining how rules can be specified for selecting a particular subtree according to a given context. Conditions can be defined recursively and numerical and logical operators are provided $(=, <, >, \vee, \wedge)$ to cope with several context parameters. In figure 1.3 an example is presented where the current context will be decided on the basis of comparing X and Y coordinates provided by a GPS module. The XML specification provides a way to exchange context information. Tool support is required encapsulate the use of XML from the designer.

Note the approaches described in [15, 16] focus on the design of the interface at the task level. This work shows how the task model is used at runtime to generate context-dependent UIs. This will be done by providing a framework (Dygimes, section 1.3) that can interpret a task specification and generate a presentation for the given task specification. The framework resolves the context dependencies beforehand, resulting in a presentation that is adapted to the context of use. The next section explains how we proceed from the task specification to the presentation of the UI by using a dialog model.

4.2 The Dialog Model

Before applying further processing of the task model, it has to be transformed into a concrete one (resolve all the decision nodes) in order to extract a dialog model. The *context-specific* task model is a normal ConcurTaskTree, suited for the current context of use and can be processed as any other ConcurTaskTree. The transformation can be done by replacing the decision node with the appropriate subtree representing a subtask suitable for the current context of use.

In [9] we proved it is possible to generate simple UIs directly from the task specification. This was done through the automatic generation of a dialog specification from the task specification. In our approach, the dialog model is expressed as a State Transition Network (STN) and

```
<?xml version="1.0"?>
<!ELEMENT decision ((cond,true,false)
        |(value,case+))>
<!ELEMENT cond (value,value)>
<!ATTLIST cond type CDATA #IMPLIED>
<!ELEMENT value ( cond | #PCDATA )>
<!ATTLIST value type CDATA #IMPLIED>
<!ELEMENT true (#PCDATA)>
<!ATTLIST true platform #IMPLIED>
<!ELEMENT false (#PCDATA)>
<!ATTLIST false platform CDATA #IMPLIED>
<!ELEMENT case (value|cond)>
<!ATTLIST case platform CDATA #IMPLIED>
```

Figure 1.2. Decision DTD

```
<decision>
  <cond type="and">
    <value type="cond">
       <cond type="lt">
         <value type="context">
           GPS:Xcoord
         </value>
         <value type="int">
           1
         </value>
       </cond>
    </value>
    <value type="gt">
       <cond type="equals">
         <value type="context">
            GPS:Ycoord
         </value>
         <value type="int">
            54
         </value>
       </cond>
    </value>
 </cond>
 <true platform="context">left</true>
 <false platform="context">right</false>
</decision>
```

Figure 1.3. Decision XML example

each state in the STN equals an ETS. In the UI, the information about the tasks in an ETS have to appear together in the resulting UI. The transitions between dialogs are represented in the STN by transitions between states, marked with the tasks that can *trigger* the change. The transitions between the different ETSs ("dialogs") are identified by the different temporal operators connecting selected tasks located in the different ETSs. An extensive description of the algorithm can be found in [9]. An open source tool is provided that implements this algorithm and calculates a dialog model from the task specification at: `http://www.edm.luc.ac.be/software/TaskLib/`.

4.3 The Presentation Model

The last step has to render the dialog model on the available output devices. This is the presentation of (the different parts of) the concrete UI.

The nodes in the dialog model are ETSs. One such node represents all UI building blocks that have to be presented to complete the current ETS (section 1.3 showed that UI building blocks were attached to individual tasks). The tasks in an ETS are also marked with their target device, so two different situations are possible:

1 All tasks in an ETS are targeted to the same device

2 Not all tasks in an ETS are targeted to the same device

Situation (1) allows the UI to be rendered completely on one device. (2) demands that the UI to be *distributed* over different devices. For this purpose the device-independence of the abstract UI description has to be extended towards the use of multiple devices. On the level of the presentation model, the Abstract UI descriptions of a dialog are rendered as concrete dialogs, this can be accomplished by using two important techniques:

- Customized mappings from Abstract Interaction Objects (AIOs) to Concrete Interaction Objects (CIOs) [17]. The rendering engine for each device can choose for itself the concrete widget selected to present an AIO. This can be customized afterwards by the designer [7].

- Positioning of the widgets is done through constraints which are defined in a language-independent manner. The renderer can use the information about the hierarchical widget containment to split up the UI in different parts. Details of this approach can be found in [10].

Customized mapping rules and device-independent layout management are two important techniques for realizing device-independent distributed UIs.

It is possible several concurrent tasks located in the same ETS have to be rendered on different devices. Since the presentation building blocks are attached to the tasks as XML documents, the presentation for an individual device can be calculated for each device separately. Notice when concurrent tasks are rendered on separate devices, some kind of middleware will be necessary to support data-exchange between both tasks in a heterogeneous environment. In contrast with e.g. WebSplitter [8] the focus is not on distribution of content, but distributed support of task execution.

5. A Case Study: Manage Stock

Figure 1.4 shows the *manage stock* example. The following situation occurs: the storekeeper of a warehouse keeps track of the stock using two devices. First a desktop PC is used to manage the purchase and sales of articles. Second an employee checks and updates the stock amounts using his PDA to note the changes immediately. When the amount of a certain article is updated by the desktop PC, for example when new goods are purchased, the employee receives a message on his PDA. When he/she stands in the vicinity of a printer supporting Radio Frequency Identifier (RFID) tags, this can be detected and the information of the product can be viewed and printed. As a result, the example contains

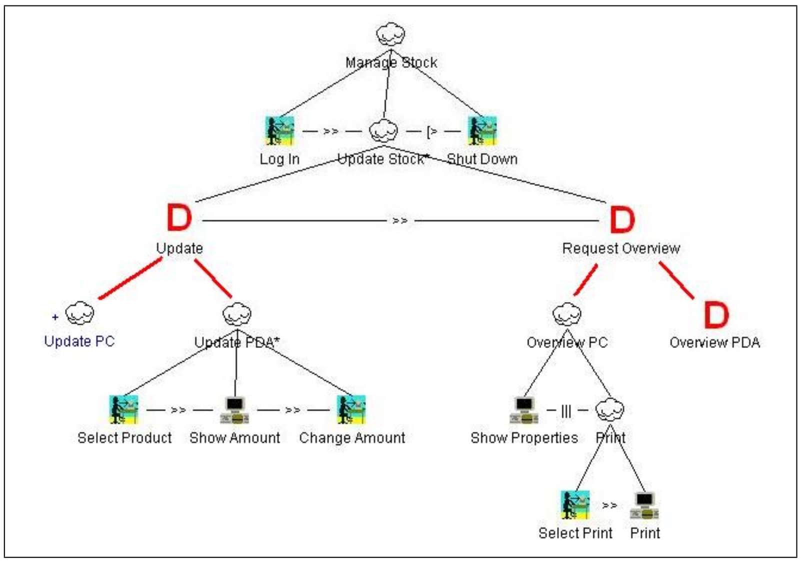

Figure 1.4. Context-Sensitive Task Model of the *Manage Stock* example

two types of context denoted by the decision tasks: platform (*Update* and *Request Overview*) and location (*Overview PDA*). To link the context handler to the appropriate decision node, decision rules need to be attached to these nodes. Figure 1.6 shows an example for the *Overview PDA* task. In this case there will be a call for the *canPrint* function in the RFID Reader.

The first step to automatically generate the UI is to convert the context-sensitive task model into a context-specific task model. This is why the condition in the decision XML has to be evaluated for each decision node and the decision node is replaced by its subtree which matches the current context. In the *Overview PDA* task example, there will be an evaluation of the *canPrint* function. If the return value equals *true* the *Properties (Printing)* subtree will replace the decision node, else the *Show Properties (No Printing)* will. Figure 1.8 shows the context-specific task model in case of using the PC to change the stock amounts

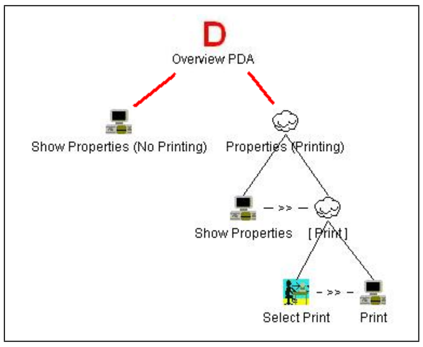

Figure 1.5. *Overview PDA* subtree

```
<decision>
 <cond type="equals">
  <value type="context">
   RFID:Reader:canPrint
  </value>
  <value type="boolean">
   true
  </value>
 </cond>
 <true platform="context">
  Show Properties (No Printing)</true>
 <false platform="context">
  Properties (Printing)</false>
</decision>
```

Figure 1.6. Decision rules for the *Overview PDA* task

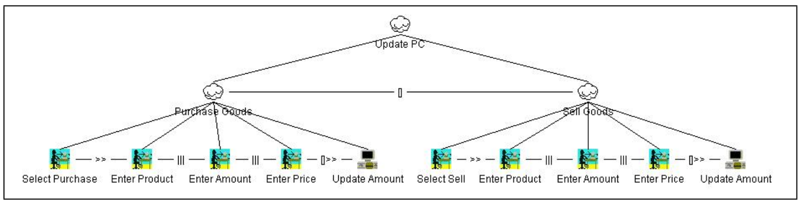

Figure 1.7. *Update PC* subtree

and the PDA to notify the employee within the reach of an RFID supporting printer.

The next step uses a custom algorithm (described in [5]) to calculate the *enabled task sets* (ETSs):

$$\begin{aligned}
ETS_1 =\ & \{LogIn\} \Rightarrow P_{all} \\
ETS_2 =\ & \{SelectPurchase(P_{pc}), SelectSell(P_{pc}), ShutDown\} \Rightarrow P_{pc} \\
ETS_3 =\ & \{EnterProduct(P_{pc}), EnterAmount(P_{pc}), EnterPrice(P_{pc}), ShutDown\} \\
& \Rightarrow P_{pc} \\
ETS_4 =\ & \{EnterProduct(P_{pc}), EnterAmount(P_{pc}), EnterPrice(P_{pc}), ShutDown\} \\
& \Rightarrow P_{pc} \\
ETS_5 =\ & \{UpdateAmount(P_{pc}), ShutDown\} \Rightarrow P_{pc} \\
ETS_6 =\ & \{UpdateAmount(P_{pc}), ShutDown\} \Rightarrow P_{pc} \\
ETS_7 =\ & \{ShowProperties(P_{pda}), ShutDown\} \Rightarrow P_{pda} \\
ETS_8 =\ & \{SelectPrint(P_{pda}), ShutDown\} \Rightarrow P_{pda} \\
ETS_9 =\ & \{Print(P_{pda}), ShutDown\} \Rightarrow P_{pda}
\end{aligned} \tag{1.1}$$

P_x indicates on which platform the tasks can be executed. $x = all$ means the platform does not matter, and the task can be executed both on a PC or on a PDA. This example only contains tasks restricted to either a PC or a PDA because no ETS contains tasks marked P_{pc} and P_{pda}. Remark that the only difference between ETS_3 and ETS_4, and

ETS_5 and ETS_6 is they are children from another task. Afterwards, the

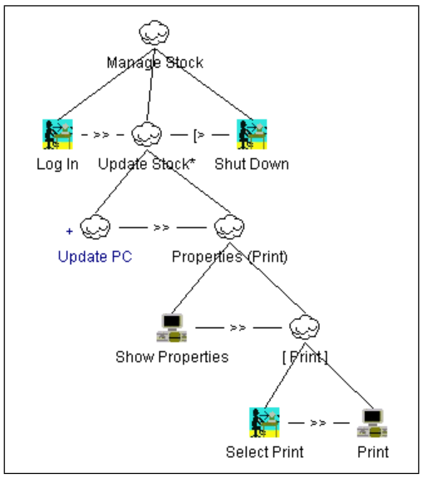

Figure 1.8. Context-Specific Task Model

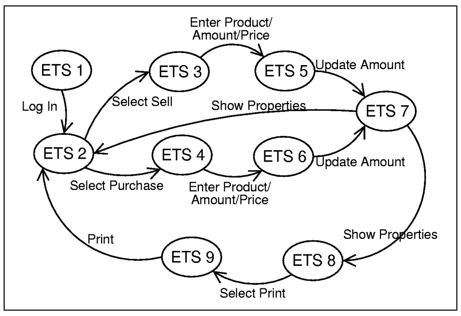

Figure 1.9. Dialog Model (The accept state caused by the *Shut Down* task is omitted to avoid cluttering the picture.)

dialog model (figure 1.9) is automatically extracted. Finally the actual UI is rendered by the runtime environment. Figure 1.10 shows the dialog model with the rendered UIs.

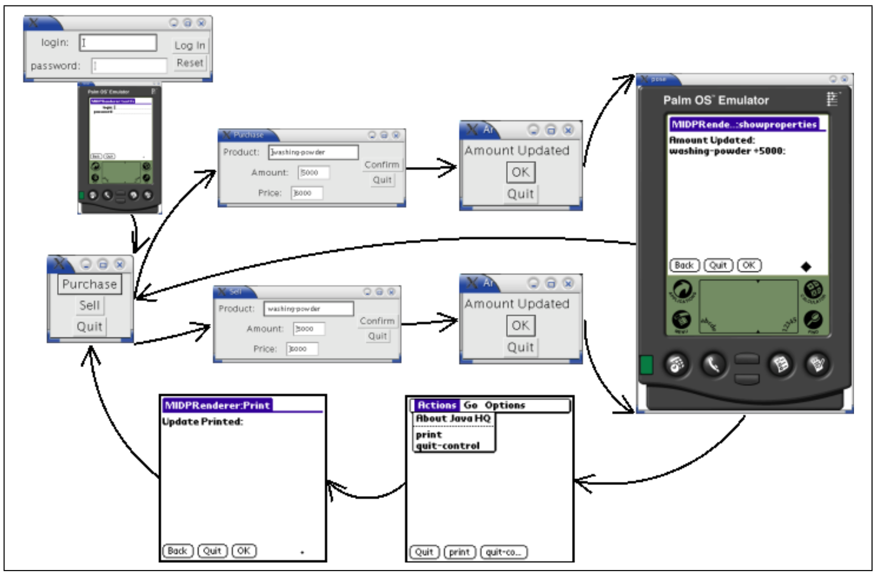

Figure 1.10. Dialog Model with the concrete dialogs

6. Conclusions and Future Work

This paper shows how context information can be integrated in interface design to generate multi- and multiple-device user interfaces at runtime. The ConcurTaskTrees formalism is combined with decision nodes and rules to allow the user interface to adapt to the context while still being consistent w.r.t. the design. An important case is where the context can indicate the change in interaction device while executing a task. Our model allows this change by providing an appropriate dialog model including the transitions between dialogs on the same device *and* transitions between dialogs on different devices. The presentation model also supports dialogs that are distributed over several devices. The precondition to make this work is the context must be frozen from the start until the end of the main task.

Future work involves finding a way to switch the context-concrete task model on a context change in order to recalculate the dialog and presentation model. This approach however comes with a lot of complications. First of all, the new dialog model may not be compatible to the old one and disrupts the continuity of the user interface. This is because the current state might not occur in the new dialog model. Also it is dangerous to adjust the user interface every time the context changes. In some cases the user can become confused about a sudden changed user interface.

Finally we believe the presented process is a first practical step towards involving context in design.

7. Acknowledgments

Our research is partly funded by the Flemish government and European Fund for Regional Development. The SEESCOA (Software Engineering for Embedded Systems using a Component-Oriented Approach) project IWT 980374 and CoDAMoS (Context-Driven Adaptation of Mobile Services) project IWT 030320 are directly funded by the IWT (Flemish subsidy organization). The authors would like to thank Bert Creemers for his contribution.

References

[1] Marc Abrams, Constantinos Phanouriou, Alan L. Batongbacal, Stephen M. Williams, and Jonathan E. Shuster. UIML: An appliance-independent XML user interface language. *WWW8 / Computer Networks*, 31(11-16):1695–1708, 1999.

[2] Mir Farooq Ali and Manuel A. Pérez-Quiñones. Using Task Models to Generate Multi-Platform User Interfaces while Ensuring Usability. In *Proceedings of CHI'2002 (Short Paper), Minnesota, USA*, 2002.

[3] Gaelle Calvary, Joelle Coutaz, and David Thevenin. Embedding plasticity in the development process of interactive systems. In *6th ERCIM Workshop "User Interfaces for All". Also in HUC (Handheld and Ubiquitous Computing) First workshop on Resource Sensitive Mobile HCI, Conference on Handheld and Ubiquitous Computing, HU2K, Bristol*, 2000.

[4] Gaelle Calvary, Joelle Coutaz, and David Thevenin. Supporting Context Changes for Plastic User Interfaces: A Process and a Mechanism. In *Proceedings of IHM-HCI, 10-14 september 2001, Lille, France*, 2001.

[5] Tim Clerckx and Karin Coninx. Integrating Task Models in Automatic User Interface Design. Technical Report TR-LUC-EDM-0302, EDM/LUC, 2003.

[6] 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 *Human-Computer Interaction with Mobile Devices and Services, 5th International Symposium, Mobile HCI 2003*, pages 256–270, Udine, Italy, September 8–11 2003. Springer.

[7] Jan Van den Bergh, Kris Luyten, and Karin Coninx. A Run-time System for Context-Aware Multi-Device User Interfaces. In *HCI International 2003, Volume 2, Crete, Greece*, pages 308–312. Lawrence Erlbaum Associates, June 2003.

[8] Richard Han, Veronique Perret, and Mahmoud Naghshineh. WebSplitter: a Unified XML Framework for Multi-device Collaborative Web Browsing. In *Proceedings of the 2000 ACM conference on Computer Supported Cooperative Work*, pages 221–230. ACM Press, 2000.

[9] Kris Luyten, Tim Clerckx, Karin Coninx, and Jean Vanderdonckt. 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, Funchal, Madeira Island, Portugal, June, 2003*. Springer LNCS, 2003.

[10] Kris Luyten, Bert Creemers, and Karin Coninx. Multi-device layout management for mobile computing devices. Technical Report TR-LUC-EDM-0301, EDM/LUC, 2003.

[11] Giullio Mori, Fabio Paternò, and Carmen Santoro. Tool Support for Designing Nomadic Applications. In *Proceedings of the 2001 International Conference on Intelligent User Interfaces, January 12-15, 2003, Miami, FL, USA*, pages 141–148. ACM, 2003.

[12] Jeffrey Nichols, Brad A. Myers, Michael Higgins, Joseph Hughes, Thomas K. Harris, Roni Rosenfeld, and Mathilde Pignol. Generating remote control interfaces for complex appliances. In *Proceedings of the 15th annual ACM symposium on User interface software and technology*, pages 161–170. ACM Press, 2002.

[13] Fabio Paternò. *Model-Based Design and Evaluation of Interactive Applications*. Springer Verlag, ISBN: 1-85233-155-0, 1999.

[14] Fabio Paternò and Carmen Santoro. One model, many interfaces. In Christophe Kolski and Jean Vanderdonckt, editors, *CADUI 2002*, volume 3, pages 143–154. Kluwer Academic, 2002.

[15] Costin Pribeanu, Quentin Limbourg, and Jean Vanderdonckt. Task Modelling for Context-Sensitive User Interfaces. In Chris Johnson, editor, *Interactive Systems: Design, Specification, and Verification*, volume 2220 of *Lecture Notes in Computer Science*, pages 60–76. Springer, 2001.

[16] Nathalie Souchon, Quentin Limbourg, and Jean Vanderdonckt. Task Modelling in Multiple contexts of Use. In Peter Forbrig, Quentin Limbourg, Bodo Urban, and Jean Vanderdonckt, editors, *Interactive Systems: Design, Specification, and Verification*, volume 2545 of *Lecture Notes in Computer Science*, pages 60–76. Springer, 2002.

[17] Jean Vanderdonckt and François Bodart. Encapsulating knowledge for intelligent automatic interaction objects selection. In *ACM Conference on Human Aspects in Computing Systems InterCHI'93*, pages 424–429. Addison Wesley, 1993.

[18] Chris Vandervelpen, Kris Luyten, and Karin Coninx. Location Transparant User Interaction for Heterogeneous Environments . In *HCI International 2003, Volume 2, Crete, Greece*, pages 313–317. Lawrence Erlbaum Associates, June 2003.