A Run-time System for Context-Aware Multi-Device User Interfaces

Jan Van den Bergh *Kris Luyten* *Karin Coninx*

Limburgs Universitair Centrum
Expertisecentrum Digitale Media[1]
Universitaire Campus
B-3590 Diepenbeek, Belgium
{Jan.VandenBergh, Kris.Luyten, Karin.Coninx}@luc.ac.be

Abstract

In the last few years, the use of mobile computing devices has tremendously increased and expectations are that the growth of mobile computing will continue for many years to come. This evolution creates new challenges for the design of user interfaces. In order to effectively use the new technology, an application will need to be usable on different platforms, in different circumstances and by different users. Thus, the user interface of the application has to adapt to the context in which it will be used. In this paper, we will address the problem of context-aware computing and how a user interface can be adapted according to its context at runtime and still be usable. We will define the concept *context of rendering* in order to effectively incorporate the information provided by the *context of use* in the rendering of a context sensitive user interface. To achieve this, we will combine task models with abstract user interface descriptions and context dependent information. The approach that is taken will be illustrated by the discussion of a proof-of-concept implementation.

1 Introduction

With the future of computing laying in embedded systems and portable devices the current practice of ad hoc design of user interfaces (UIs) can pose serious problems. Even more because the market of these portable devices evolves very fast, which makes it almost impossible to know the properties of the devices on which a product must run when it is designed. These assumptions lead to the following properties of user interfaces for future applications: (1) they should be executable on multiple platforms, (2) their appearance needs to be, at least partially, defined at run-time and (3) they should maintain usability (i.e. be plastic (Calvary, G. et al. 2002)). In this paper we introduce a run-time system that allows for the execution of multi-device user interfaces whose appearance can be defined by both run-time and design time parameters, bundled into a context description. The system builds on the strong foundation of model-based and, more specifically, task-based design.

Several existing methods are also based on task-based design. Some of them use the ConcurTaskTrees (CTT) as notation for the task model. CTT is a hierarchical notation in which an abstract task is refined into more fine-grained abstract tasks, user tasks, interaction tasks and system tasks. Among tasks on the same level, temporal relations are specified such as “enables”,

[1] http://www.edm.luc.ac.be

"disables", "choice" and "concurrent". (Souchon, Limbourg & Vanderdonckt, 2002) proposed several methods to allow modeling for multiple contexts of use employing task models in the ConcurTaskTrees notation, while (Calvary et al., 2002) proposed to use translation and reification of different models to generate UIs for multiple contexts of use. Both approaches, however, have in common that they generate different user interfaces for different contexts. (Paternò & Santoro, 2002) proposed the use of a task model in ConcurTaskTrees (CTT) notation as a single base for the design for multiple UIs.

Another approach is taken by (Braubach et al., 2002). They let the user specify the user interface using a domain model, which contains a task model (UML Use Case Diagram), and an object model, a presentation model and a dialogue model. These three models are used in the runtime environment where they are combined with the components from GUI toolkits and domain specific objects. This approach delivers a flexible runtime environment, however the use of the Use Case Diagram implies severe limitations in the specification of the tasks and the runtime environment is rather extensive. In the following sections we will introduce our work, starting with a new definition of context, followed by a discussion of a proof-of-concept implementation.

2 Context

Multiple definitions of context are given in literature. In our approach, we adhere to the definition of context given in (Calvary, et al. 2002). It divides the *context of use* into two parts:

1. The attributes of the physical and software platform, such as the screen size and the available interactors
2. The environmental attributes that describe the physical surroundings of the interaction, such as the location in which the interaction is taking place.

The first part is quite easy to incorporate into the UI rendering mechanism since the properties it describes are directly related to the rendering. The latter part consists mainly of properties that can be measured by software, but need to be interpreted before the information provided by the measurements can be used to adapt the user interface to the context. Since this means that the renderer cannot directly handle the information provided by measurements of the environmental context, we define the *context of rendering* as follows:

1. The attributes of the physical and software platform as in the definition of the context of use
2. The environmental-dependent properties, which is the information provided by interpretation of the environmental attributes, such as the set of available tasks (enabled task set or ETS) that can be influenced by the location where the interaction takes place.

Although both aspects of the context of rendering can describe a lot of aspects, the current implementation is able to incorporate the changes in the tasks that are to be performed, the screen size of the target device and to make decisions about the choice of interactors based on the preferred interactors and the available interactors in a certain context.

3 Modeling and rendering a User interface

In a task-based approach to the design of a user interface, one starts by defining the tasks that need to be performed. We chose the ConcurTaskTrees (CTT) notation (F. Paternò 2000), because it is

one of the most expressive and intuitive task models currently available. In our approach, the designed CTT is refined by adding abstract user interface descriptions (AUIs) to the appropriate leaf nodes of the CTT tree. These abstract user interface descriptions are made in an XML-format. For more details we refer to (Luyten & Coninx, 2001). Because we want to design truly multi-platform and context sensitive user interfaces, we will not require the designer to create platform specific versions of the user interface. Instead, the designer can optionally define different contexts.
Currently, a context of rendering description is limited to a set of possible mapping rules that map abstract interaction objects (such as a choice) to concrete interaction objects (such as an AWT list for the Java platform) and the task set that is to be active in that context. The rest of the contextual information, such as the screen size and the available concrete interactors, is detected at runtime and it is therefore unnecessary to model it explicitly at design time.

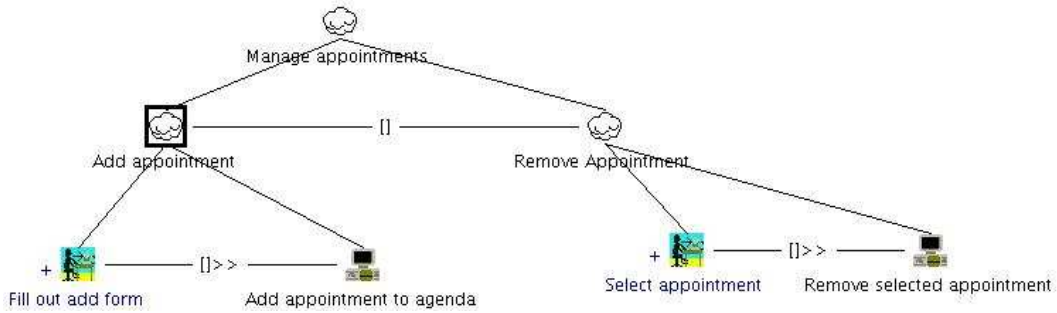

Figure 1 ConcurTaskTrees for the agenda-tool

When the necessary information for a multi-device user interface is modeled, it can be rendered. The rendering is done in three stages:

1. The AUIs that will be rendered are determined based on the information in the context of rendering, the enabled task set[2] that is "active"
2. The context information is read, such as the possible mappings, and determined, such as the screen size
3. The AUIs are translated to final user interfaces and rendered to the screen.

4 Rendering: an example

In this section, we will explain the rendering process as it is done in our current implementation into more detail using a minimal agenda tool as an example. The implementation extends the Dygimes framework, already used in previous work (Luyten & Coninx 2001). The agenda tool has only two interactive functions: adding appointments and deleting them.

Figure 1 shows a CTT-tree for this application. The AUI descriptions for the tasks "Add Appointment" and "Remove Appointment" are linked to the appropriate leafs of the CTT-tree. Listing 1 shows the code for the specification of the month in the former AUI description and the information is translated into an enabled task set description in a compact XML-format. Some mappings are specified in order to adapt the final user interface to the preferences of the user. Two of the specified mapping rules are shown in listing 2.

[2] The enabled task set can be automatically calculated from a CTT model (Paternò, 2000) and modified by changes in the context of use.

The listing shows two mapping rules as XML-tags that specify a mapping of an abstract interaction object (AIO) to a concrete interaction object (CIO). The rules have two mandated child tags in common: *aio* and *cio* which represent the type of AIO the rule applies to and the CIO it will be mapped on using that rule. The first rule has an optional *infocio* tag that specifies the CIO that will be used to display the information specified by the info tag of the abstract interactor in the AUI description. The second rule has an optional *name* tag that limits the application of the mapping rule to the interactors with a name that contains the string specified by the constraint. When there are multiple rules that could apply, the one with the best match is chosen. The absence of a *name* tag is equivalent to an empty *name* tag.

(a) Without mappings (b) With mappings (c) With mappings, small screen

Figure 2: Three different renderings of ConcurTaskTrees in figure 1

When the user interface of the application is rendered, the available tasks are read from the context description. For this simple example, there will be two tasks, except when specified otherwise. When the tasks are read, the applicable mappings are determined and the available screen size is calculated. There might be several rules that could be applied to map an AIO to a CIO, the correct rule is chosen based on the constraint that matches the specific AIO the closest. In our example,

Listing 1: Abstract User interface description

```
<?xml version =”1.0”?>
<ui>
  <title>appointment </title>
  …
  <interactor>
    <choice name=”dateMM”>
      <info>month</info>
      <choicetype>single </choicetype>
      <item>January</item>
      …
      <item>November</item>
      <item>December</item>
    </choice>
  </interactor>
  …
  </group>
</ui>
```

Listing 2: Two of the specified mapping rules

```
<mapping>
  <aio2cio>
    <aio>choice</aio>
    <cio>awt . VertCheckboxGroup</cio>
    <infocio>awt.Label</infocio>
  </aio2cio>
</mapping>
<mapping>
  <aio2cio>
    <aio>choice</aio>
    <cio>awt.Choice</ cio>
    <name>date</name>
  </aio2cio>
</mapping>
```

both mapping rules in listing 2 could be applied. The first rule has no *name* tag, which means it is the default. The second tag, however, will be used to represent the interactor *date* because it has the most stringent constraint. Figure 2(a) shows the user interface rendered without applying the mapping rules but chosen by a fully automatic mapping engine, figure 2(b) shows the same interface rendered with the mapping rules applied to it. The result of the rendering with the same mapping rules, but in a context where less screen space is available, is shown in figure 2(c).

5 Conclusion and Future Work

We presented a new concept, "context of rendering", that represents a translated and specialized "context of use" so that it can be directly used to adapt the rendering of the user interface taking into account various aspects of the context. We then explained a prototype-type implementation of this concept by extending the Dygimes framework.

The extension of the Dygimes framework in such ways so that it allows easy translation of "context of use" into "context of rendering" is planned as future work. The implementation of a tool that allows easy and guided specification of the mappings and an extension of the mapping system to translate high level constraints between widgets (such as "gives information on" or "cancels") into low level positional constraints such as ("left of" and "bottom left") are planned.

6 Acknowledgements

Our research is partly funded by the Flemish government and European Fund for Regional Development (EFRD). The SEESCOA project IWT 980374 is directly funded by the Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT).

7 References

Braubach, L., Pokahr, A., Moldt, D., Bartelt, A., Lamerdorf, W. (2002). Tool-supported interpreter-based user interface architecture for ubiquitous computing. In P. Forbrig et al. (Eds.), *Interactive Systems*, LNCS 2545 (pp. 89-103). Heidelberg: Springer.

Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Souchon, N., Bouillon, L., Florins, M., & Vanderdonckt, J. (2002). Plasticity of user interfaces: A revised reference framework. In C. Pribeanu & J. Vanderdonckt (Eds.) *Proceedings of TAMODIA 2002* (pp. 127-134). Bucharest: INFOREC Printing House.

Luyten, K. & Coninx, K. (2001). An XML-based runtime user interface description language for mobile computing devices. In C. Johnson (Ed.), *Interactive Systems*, LNCS 2220 (pp. 17-29). Heidelberg: Springer.

Paternò, F. (2000). *Model-Based Design and Evaluation of Interactive Applications*. Heidelberg: Springer.

Paternò, F. & Santoro, C. (2002). One model, many interfaces. In C. Kolski & J. Vanderdonckt (Eds.), *Computer-Aided Design of User interfaces III* (pp. 143-154). Dordrecht: Kluwer Academics Publishers.

Souchon, N., Limbourg, Q. & Vanderdonckt, J. Task modeling in multiple contexts of use. In P. Forbrig et al. (Eds.), *Interactive Systems*, LNCS 2545 (pp. 59-73). Heidelberg: Springer.