Using UML 2.0 and Profiles for Modeling Context-Sensitive User Interfaces

Jan Van den Bergh Karin Coninx
Hasselt University
Expertise Centre for Digital Media
and
transnationale Universiteit Limburg
School of Information Technology
Wetenschapspark 2
BE-3590 Diepenbeek
Belgium
{jan.vandenbergh, karin.coninx}@uhasselt.be

ABSTRACT

Significant work has been established in both the HCI community and the software engineering community to structure and to rationalize development within their respective fields using abstractions that are crystallized into a limited set of models. Each of these models gives a precise definition of one of the aspects of the design. In this position paper we present a more detailed analysis describing how UML 2.0 and its light-weight extension mechanism, profiles, can be used to model both the dynamic and structural aspects of context-sensitive user interfaces.

Categories and Subject Descriptors

H.5.2 [**User Interfaces**]: Screen design—*Abstract User Interfaces*; D.2.2 [**Design Tools and Techniques**]: User Interfaces, Flow charts, Object-Oriented design methods—*UML 2.0, Model-Driven Design, Model-based Design of User Interfaces*

General Terms

Model Driven Development of User Interfaces, UML 2.0, profiles

1. INTRODUCTION

The number of devices and their capabilities is very divers and becoming even more so. Model-based Design of User Interfaces (MB-UID) is a methodology often proposed to support the design and development of multi-platform, and even context-sensitive user interfaces. In the software engineering camp, model-driven development is proposed for solving similar problems. Both approaches use different models, but sometimes they have similar semantics.

As part of our effort to model context-sensitive interactive applications, we want to express the models used in MB-UID using the UML 2.0 and extend it using profiles where necessary. In this paper, we present an analysis describing which diagrams in UML 2.0 [8] have semantics that correspond best to most used models in MB-UID. Based on this analysis we present our choice of UML-diagrams for abstract/high-level models and the profiles we designed to extend the semantics of the UML 2.0 models so that they better reflect the semantics of the MB-UID models. The reason for this approach is that usage of the same (basic) meta-model allows easier integration of the different approaches, which can be necessary to create truly usable (context-sensitive) applications. After all, usability issues are related to application characteristics beyond the user interface.

For the modeling of context-sensitive user interfaces, we consider a limited set of models as shown in figure 1. In this paper we will shortly discuss the different models and how they can be represented in UML 2.0. We will focus on the core models that deal with the representation of the dynamic and structural aspects of the user interface. The dynamics are described in the task and dialog model, while the structure is described in the abstract and concrete presentation models, optionally complemented with a user interface deployment model. Finally, conclusions and future work are presented in section 5.

2. MODEL OVERVIEW

In figure 1, the different models we consider to be important for model-driven design of user interfaces are graphically depicted. The figure is divided into four sections. The upper half of the figure contains the platform independent models (PIM), while the lower half contains the platform-specific models (PSM). Similarly, the models at the left describe some structural aspects, while the right-hand side describes the behavioral models. All models are specified using UML 2.0, except the one with the gray background.

In the upper right part of the figure, the task model is shown. It describes the tasks that have to be performed to reach a certain goal. The dialog model is a more concrete version of the task model in that it describes how the user can perform the tasks through a user interface and how the system responds to these actions. The task (and dialog) model is discussed into more detail in section 3, the reason for having two instances of the task model is explained in section 3.4.

The structural platform independent models consist of the context model and application model, which can be considered to grossly correspond to the domain model used in software engineering approaches, and the (abstract) presentation model. The latter describes the structure of the user interface in a way that is independent of platform and

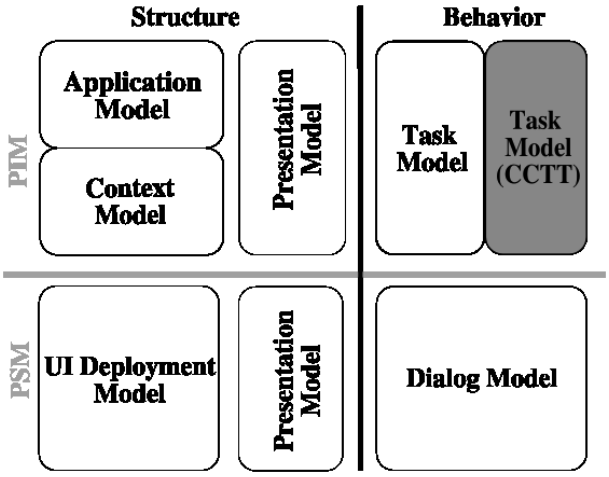

Figure 1: Models for specification of context-sensitive user interfaces

modality and is discussed into more detail in section 4.

3. TASK AND DIALOG MODEL

3.1 Generic Model Description

A task model describes the tasks of a user interacting with a system to accomplish a certain goal. It is a hierarchically structured model that describes the compositional and often also temporal relationships of a certain activity. The dialog model describes how the tasks can be performed by the user (in interaction with the system); it thus is a lower-level, platform-dependent version of a task model and can be derived to a great extend of the task model.

In case of context-sensitive applications, a task model can also contain references to context. A task and dialog model can contain references to concepts since many tasks involve some kind of object: e.g. the name of a person, (the date of) an appointment.

3.2 Properties

One of the well accepted task notations for MB-UID is the graphical ConcurTaskTrees (CTT) notation [4]. This notation uses a tree structure to decompose a task into subtasks, which are either performed by the user, the system or by the user in interaction with the system. A fourth category of task (abstract task) is used when a task has subtasks of a different type. Sibling tasks are connected using temporal operators and each task can have a loose, more fine-grained specification of the type of task, the involved objects, performance statistics, etc.

Some adapted notations have been proposed for specific purposes. The Contextual CTT [9] is one such notation, which introduces "environment tasks/actions" which are performed by an entity which is nor the system nor the user. Examples of environment actions can be location changes (in case of interaction with a mobile device) or a change in privacy (e.g. someone entering the room) or an incoming phone call while interacting with a mobile phone application.

3.3 Corresponding diagram

Since the task model defines the required behavior of the user (and the system), only behavior models can have the required semantics. Another criterion is that a single task model can be used in multiple scenario's. Furthermore, the users and their actions are central in this model. These properties can only be matched by two diagram types. The use case diagram highlights the system and the users for specific use cases. Therefore, the use case diagram can be useful for initial discovery and requirements analysis. It is not suitable to make a detailed design.

The activity diagram is the most suitable to represent the task model for the following reasons:

- Actions and the temporal relations between them are the focus of this diagram type.
- Links with the concepts that are manipulated by the actions can be specified.
- One can specify the actor for each action.
- One can establish hierarchies to cope with complexity; actions can be bundled in activities. Each activity can be shown in a separate diagram, but also within a containing diagram.
- The semantics using token-flow allow a tool or designer to verify which actions can be performed at a certain moment in time (and thus allow derivation of a dialog diagram, something which is also possible using CTT).

In related work, Nobrega et al. [6] already used the activity diagram as the basis for a new diagram type to express the CTT in a tree-based UML diagram. Pinheiro da Silva and Paton [2] defined a flow-based diagram type that defined special object flows to objects that represented user interface elements. It was based on the State Machine package, to create a special form of activity diagram [1]. The use case diagram [2, 7] and the class diagram [7] — it was extended using stereotypes to define a semantical equivalent of the CTT notation in UML 1.x — were also used to express the task model.

3.4 ConcurTaskTrees and Activity Diagrams

The ConcurTaskTrees notation is one of the most well accepted task modeling notations. We investigated whether mapping this tree-based structure with different task categories and temporal operators to the flow-based notation used in the activity diagram is possible. The mapping of the temporal operators could pose problems [6]. Although two temporal operators from the CTT notation, suspend/resume and order independence, do not have direct corresponding elements in UML, their behavior can be modeled using the activity diagram notation. The suspend/resume behavior can be modeled by nesting the suspending actions in a StructuredActivityNode with its attribute mustIsolate set to true, and placing it in parallel to the suspended actions. The order independent operator can be modeled in a similar fashion.

It is possible to create a hierarchy using the combination of actions and (invoked) activities. Furthermore, the categories

[1]In UML 1.x the activity diagram was based on the state machine diagram.

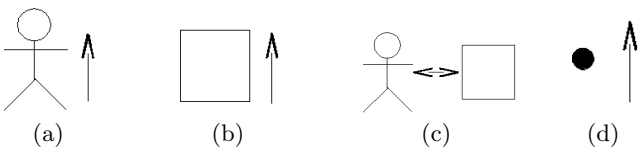

Figure 2: Action stereotypes: (a) user, (b) device, (c) interact and (d) environment

of tasks in (Contextual) CTT can be provided by extending the activity diagram with a profile containing a stereotype for each type of task/action (see figure 2 and 4). Note that we opted to specify no special stereotype for the category of abstract tasks. A more detailed explanation of the reasons for this decision can be found in [10].

As can be seen in figure 1, we opted to keep two representations for the task model; one, the (Contextual) CounTaskTrees, gives a tree-based representation of the model and highlights the user's mental model, while the second, using the activity diagram, uses a flow-based structure and gives the interactive system and the manipulated objects a more prominent place. In this way, the advantages of both notations can be combined.

4. PRESENTATION MODEL

4.1 Generic Model Description

The presentation model represents the structure of a user interface. It can be modeled at three levels of abstraction: abstract user interface, concrete user interface and final user interface. In the following sections we will only discuss the abstract presentation model. The concrete presentation model already deals with the real appearance of a user interface, which can be represented more adequately using specialized tools that can offer better support for all available widgets and style options. The final user interface is the running, deployed user interface.

The goal of the abstract presentation model is to define the structure of the user interface in a platform-independent manner. This implies that no assumptions can be made regarding platform, toolkit or modality. One reason to have such a model is to derive interfaces for different platforms and/or toolkits that have a consistent structure. Another reason is that the focus can be kept on structure and the functionality a user expects rather than on low-level implementation details. This is especially important when multi-platform designs are made.

4.2 Properties

There isn't as much consensus about the contents of the abstract presentation model as there is about for example the task model. The number of concepts that are available as well as their exact nature differs greatly. Some approaches [3, 10] offer a very limited set of abstract interactors, while others [5, 1] offer a larger set of interactors. While [3] focuses on aspects of the abstract interactors, others[10, 5, 1] offer a hierarchically structured set of interactors. Another distinction is that some authors focus on the abstraction of concrete interactors [3, 5], others focus on the related actions of the user [1, 10]. All these approaches however have in common that due to the abstraction the number of different interactors is still limited.

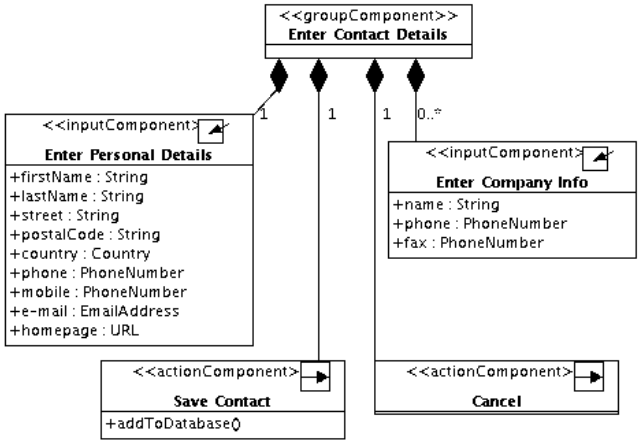

Figure 3: Example presentation model

4.3 Corresponding Diagram

Presentation models, both abstract and concrete [2], define the structure of a user interface by specifying relationships between different classes of objects, which can be expressed using class diagrams as is done in the Wisdom approach [7]. UMLi [2] on the other hand, defines a new diagram-type similar to a UML deployment diagram. Usage of a stereotyped deployment diagram as introduced in [10] to define the presentation structure was confusing for the software engineering community due to the differences in semantics.

Just as in the Wisdom approach, we propose to model the user interface structure using the class diagram. We however use a different set of stereotypes as we have different priorities for the presentation diagram (a subset of the stereotypes can be seen in figure 4). We want to emphasize the categories of the user interface parts a user will interact with: actionComponent (part of the user interface that triggers the activation of a function in the application's functional core), inputComponent (part of the user interface that allows the user to specify values) or outputComponent (part of the user interface that presents data, such as search results or a system notification, to the user). We do this by providing alternative representations for these stereotyped classes and by not specifying them as attributes of groupComponents, but as separate classes. By defining them as separate classes, we can also show the type of data that inputComponents manipulate and outputComponents present, as well as the operations that actionComponents can present. We do this because the type of data that is manipulated can be important to determine an effective representation for a specific context (platform, user, etc.).

An example of a very simple abstract presentation model is shown in figure 3. We can see from this example that at the abstract level, a class-name refers to the function of the class, not to how it is represented in an actual user interface. A groupComponent is connected with the user interface components it contains (in this case two inputComponents) through containment associations. These as-

[2]We will only discuss the abstract presentation diagram since the concrete presentation diagram can be modeled without extensions using profiles if desired.

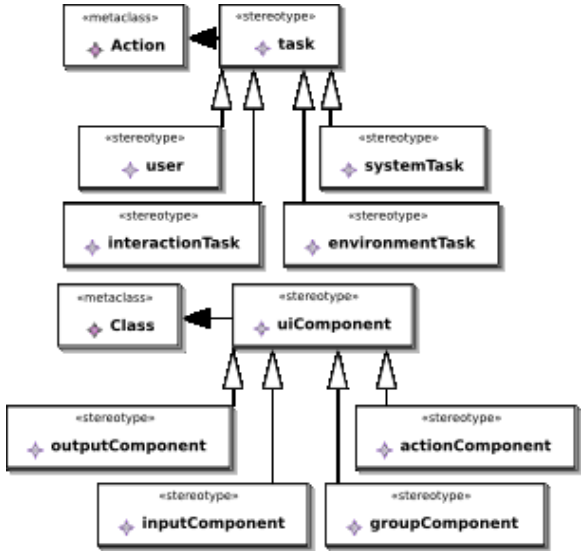

Figure 4: Relevant parts of the UML profile

sociations can have multiplicities, allowing effective specification of a variable set of inputComponents, such as the "Enter Company Info" inputComponent in the example. An inputComponent, such as "Enter PersonalInfo", can have several aspects that will be represented by a set of user interface components in a final user interface.

We are investigating the use of additional profiles to express meta-information. This meta-information can specify how public the data or controls in some parts of the interface are. For example: the interface of presentation-software might have marked the part that displays the presentation as public, while the controls to manipulate the presentation are private and thus are only shown on the monitor of the person giving a talk. Similarly, a log in can be marked private, while the password can be marked secret to denote that its contents should be masked or hidden. The use of such meta data can help the designer (or a transformation tool) to choose concrete user interface elements and/or to allocate those elements to interaction devices.

Another profile can contain the level to which a part of a user interface can be affected by context information. Such a profile might contain stereotypes such as: "context-init", initialized by a value provided by a source in the environment of the system and the user, "context-updated", updated with context information while active, etc. Awareness of parts of the user interface that are influenced by external factors or initialized by external entities can be benifitial to the overall feel of the application. It can persuade the software engineer to introduce the necessary measures that ensure that responsiveness is maintained and that the user interface reacts appropriately when the context input lacks besides ensuring that the influences of the context are possible.

5. CONCLUSIONS AND FUTURE WORK

We presented an approach to create context-sensitive user interfaces using models expressed in UML 2.0. The approach is being tested by creating a limited context-sensitive user interface using the presented approach.

In the near future, we will investigate how the different models can be combined or synchronized so that the designer can more easily identify where a specific diagram belongs in the overall design. This will involve combining the structural models into a single view that combines both the context and the abstract presentation model and will define several associations (such as update, enable or disable) that can be used to express the relationships across and within the models.

6. ACKNOWLEDGEMENTS

Part of the research at Expertise Centre for Digital Media is funded by the ERDF (European Regional Development Fund), the Flemish Government and the Flemish Interdisciplinary institute for BroadBand Technology (IBBT).

7. REFERENCES

[1] Larry L. Constantine. Canonical abstract prototypes for abstract visual and interaction design. In *Proceedings of DSV-IS 2003*, number 2844 in LNCS, pages 1 – 15, Funchal, Madeira Island, Portugal, June 11-13 2003. Springer.

[2] Paulo Pinheiro da Silva and Norman W. Paton. User interface modelling in umli. *IEEE Software*, 20(4):62–69, July–August 2003.

[3] Quentin Limbourg and Jean Vanderdonckt. *Engineering Advanced Web Applications*, chapter UsiXML: A User Interface Description Language Supporting Multiple Levels of Independence. Rinton Press, December 2004.

[4] Giulio Mori, Fabio Paternò, and Carmen Santoro. CTTE: support for developing and analyzing task models for interactive system design. *IEEE Transactions on Software Engineering*, 28(8):797–813, 2002.

[5] Giulio Mori, Fabio Paternò, and Carmen Santoro. Design and development of multidevice user interfaces through multiple logical descriptions. *IEEE Transactions on Sofware Engineering*, 30(8):507–520, aug 2004.

[6] Leonel Nobrega, Nuno Jardim Nunes, and Helder Coelho. Mapping concurtasktrees into uml 2.0. In *Proceedings of DSV-IS 05*, 2005.

[7] Nuno Jardim Nunes and Joao Falcao e Cunha. Towards a uml profile for interaction design: the wisdom approach. In *UML 2000*, pages 117–132. Springer, 2000.

[8] Object Management Group. *UML 2.0 Superstructure Specification*, October 8 2004.

[9] Jan Van den Bergh and Karin Coninx. Contextual ConcurTaskTrees: Integrating dynamic contexts in task based design. In *Second IEEE Conference on Pervasive Computing and Communications WORKSHOPS*, pages 13–17, Orlando, FL, USA, March 14–17 2004. IEEE Press.

[10] Jan Van den Bergh and Karin Coninx. Towards Modeling Context-Sensitive Interactive Applications: the Context-Sensitive User Interface Profile (CUP). In *SoftVis '05: Proceedings of the 2005 ACM symposium on Software visualization*, pages 87–94, New York, NY, USA, 2005. ACM Press.