

# CUP 2.0: High-Level Modeling of Context-Sensitive Interactive Applications

Jan Van den Bergh and Karin Coninx

Hasselt University – transnationale Universiteit Limburg  
Expertise Centre for Digital Media – Institute for BroadBand Technology  
Wetenschapspark 2  
3590 Diepenbeek  
Belgium  
{jan.vandenbergh,karin.coninx}@uhasselt.be

**Abstract.** The Unified Modeling Language is mainly being used to communicate about the design of a software system. In recent years, the language is increasingly being used to specify models that can be used for partial code generation. These efforts are mainly focussed on the generation of the application structure. It has been used to a lesser extent to model the interaction with the user and the user interface. In this paper, we introduce CUP 2.0, a Unified Modeling Language profile for high-level modeling of context-sensitive interactive applications. The profile was created to ease communication about the design of these applications between human-computer interaction specialists and software engineers. We further argue that the data provided by the models, suffices to (semi-) automatically create interactive low-fidelity prototypes that can be used for evaluation.

## 1 Introduction

<sup>1</sup> With the advent of mobile computing, the interest in development support for context-sensitive interactive applications has also increased. Indeed, the usage of applications while the users are moving makes that the context in which interactive applications is no longer a static given. The small form-factor of most of these mobile devices makes that one should make optimal use of the features of such a device and the context it is being used in. For example, in a museum a digital mobile guide can automatically display information about the art works closest to the user. Another factor is that users no longer use a single computing device but they still want to use the same applications or services on these different devices. Such applications can range from websites to word processors or even games.

The design of such context-sensitive interactive applications is a complex task that can benefit from the use of models at different levels of abstraction. The

---

<sup>1</sup> The original article is available at <http://www.springerlink.com/content/m84v34218431793p/>.

abstraction can be useful to reduce the complexity when designing the overall interactive application and reduce the chance to get lost in low-level features, such as the detailed layout of the user interface of the application on a certain target platform.

In this work, we present CUP 2.0, a profile for the Unified Modeling Language (UML) for modeling context-sensitive user interfaces that improves on an earlier version [19]. The profile provides a set of stereotypes and the accompanying tagged values that can be used to construct high-level models for these context-sensitive applications. The models are based on the models that are used in the model-based user interface design but are expressed using the UML. They document the interaction of the user with the system, the data structures accessible through the user interface, the high-level structure of the user interface and the deployment of a user interface to a certain platform.

The rest of this paper is structured as follows: after a short discussion of some related work, we will give an overview of the models that are supported by the introduced profile, followed by detailed discussions of each of the models. Finally, we will provide a discussion of the profile and conclusions.

## 2 Related Work

The UML has already been used by several approaches to model the user interfaces of interactive applications. Wisdom [13] is a UML profile for modeling interactive applications that is targeted towards small organizations. It supports modeling of interactive applications using eight different models that are expressed using the UML use case, class, activity and state diagrams. The diagrams are extended using a set of stereotypes. All models are also on a fairly abstract level and the generation process to an abstract user interface description language (AUIML) from those models is provided. CanonSketch [1] is a tool that supports the presentation model, one of the models of the Wisdom-notation, and combines it with the Canonical Abstract Prototypes [3] (CAP)<sup>2</sup> to provide multilevel modeling and HTML for prototyping on a concrete level.

UMLi [5] extends the UML using the MOF-constructs to model user interfaces. The authors introduce two new diagram types. The presentation diagram, specifying the user interface structure, is represented using a notation similar to that of the deployment diagram (for the presentation model). An enhanced version of the activity diagram is used to represent the behaviour. They extended an open-source UML-modeling tool to support their notation.

Elkoutbi et al. [8] use annotated collaboration diagrams and class diagrams to model form-based user interfaces. From these diagrams, they can generate statechart diagrams. Based on these statecharts, complete functional prototypes are generated. The approach is concentrating on form-based user interfaces for a single user. The specifications that are used as input, however, have to be rigorously defined to support the generation process.

---

<sup>2</sup> The CAP notation uses nested rectangles and a set of icons to identify the type of interaction objects contained in user interface.

MML [16] is a UML profile to model interactive multimedia applications. They use the notation we proposed in earlier work [19] to define the abstract user interface and link it with a multimedia specification, and state and activity diagrams. A skeleton of the interactive multimedia application using SVG and JavaScript can be generated from these models.

None of the above approaches, however, have dedicated support for modeling context-sensitive user interfaces. Some model-based approaches that do not use UML, however, have some support for modeling context-sensitive user interfaces. Clerckx et al. [2] propose a method that starts from a hierarchical task model from which they can generate a dialog model. This model can be annotated with high-level user interface descriptions. These models are combined with a context model to generate some concrete prototypes that can use simulated or real context input. All models can be manipulated graphically and are serialized to XML.

UsiXML [11] is a modeling language expressed using XML. It features support for the specification of task models, abstract and concrete user interface models, context models and model transformations. Tool support for various models is provided, however there is no published tool support for context-sensitive user interfaces.

### 3 Model Overview

The Context-Sensitive User interface Profile (CUP 2.0) is a UML 2.0 [14] profile that provides stereotypes and corresponding tagged values to increase support for the expression of the models, relevant to the high-level modeling of context-sensitive user interfaces, in a limited number of diagrams. Figure 1 gives an overview of the models that can be specified using the CUP 2.0 profile.

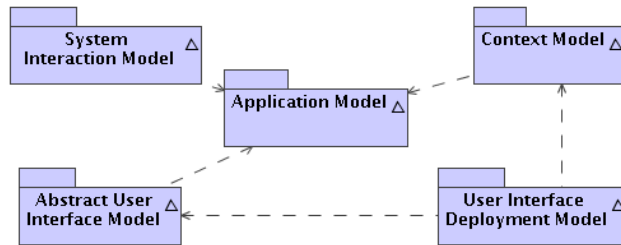
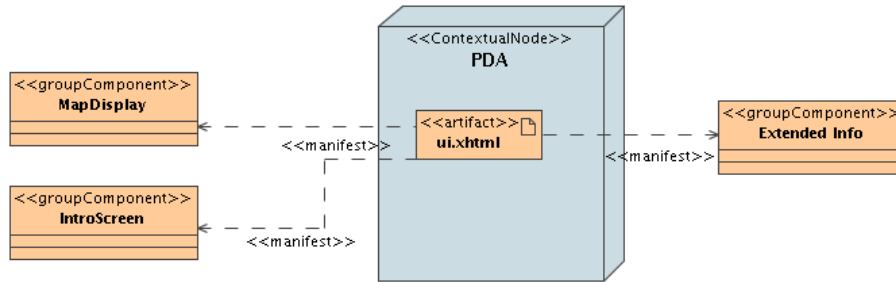


Fig. 1. Overview of the models supported by the UML profile CUP 2.0

The *application model* specifies the data structures and functionality that can be accessed through the user interface. This includes the data structures and functionality that is not part of the modeled application but that is used to provide relevant information (context) to the application. The model is used by



**Fig. 2.** Example of user interface deployment model: A context-sensitive mobile museum guide.

both the system interaction model and the abstract user interface model to provide details of the data structures which are respectively used in the interaction with the modeled application and in the user interface structure. The model is discussed in more detail in section 4.

A second model is the *system interaction model*. This model corresponds to the user task model, which is the core model in many model-based user interface design approaches. It is an hierarchical specification of the user’s tasks and user-observed tasks. In contrast to the most-used task model notation, the ConcurTaskTrees notation [15], it does not use a tree-based notation but uses the flow-based notation of the activity diagram. It does however support all temporal operators that are supported by the ConcurTaskTrees notation and is enhanced with support for context-sensitiveness. More details about this model can be found in section 6.

The structure of the context-sensitive user interface is specified in the *abstract user interface model*. A single model represents a user interface structure that is shared in multiple contexts and on multiple platforms (see section 7). The deployment of an abstract user interface to a certain platform or to a set of platforms for distributed user interfaces can be specified in the *user interface deployment model*. To accomplish this, the stereotype «contextualNode» can be applied to a Node to specify the relation with a certain context of use as specified in the context model. Figure 2 shows an example of a deployment of the user interface to a PDA. Specific contexts of use can be specified in the *context model*, which uses the classes defined in the application model. More details of the context model are found in section 5.

## 4 Application Model

The application model is specified using a class diagram. The model contains all classes of the application logic that are relevant for the user interface. In addition to those classes, also the context information and the interfaces of the relevant applications or services to get the relevant context information are included in

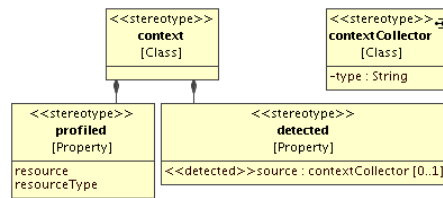


Fig. 3. Stereotypes of the UML profile CUP 2.0 relevant for the application model

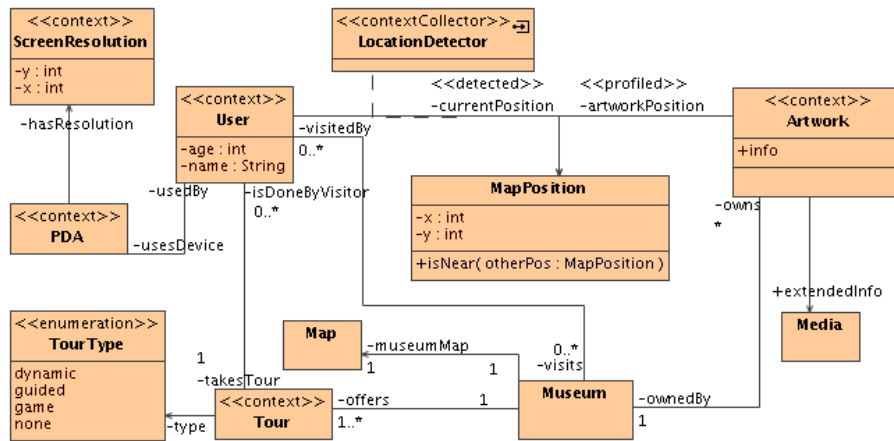


Fig. 4. Example of application model: A context-sensitive mobile museum guide.

the model. The latter classes are respectively identified using the stereotypes «context» and «contextCollector». The definition of a separate stereotype for the entities that gather context information is motivated by the fact that frameworks and toolkits built to support the development of context-sensitive applications use similar abstractions. Examples of such abstractions are context widgets in the Context Toolkit [6], contextors [4] and information spaces in ConFab [10]. A different name was chosen to be independent of the final implementation.

Each Property of classes with the stereotype «context», can have a stereotype indicating how the modeled information is gathered since this information can be important for the further design or eventual code generation. The two stereotypes that are supported are «detected» for context information that is delivered to the application directly from sensors or from any source after being manipulated, merged or derived by some service or application. Profiled context information is provided by an application or entered by a user and is indicated by the stereotype «profiled». The difference is also clear from the tagged values of these stereotypes. While the values of profiled context information can be gathered from a resource of a certain type (e.g. a URI referencing a file), the

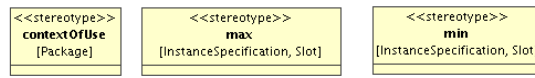


Fig. 5. Stereotypes of the UML profile CUP 2.0 relevant for the context model

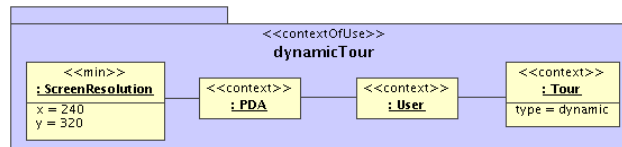


Fig. 6. Example of context model: A context-sensitive mobile museum guide.




















detected context information is gathered from a context collector. The choice to categorize context in profiled and detected was motivated by the implications this difference has on the design of the application; an appropriate user interface has to be defined to modify profiled context information, while detected information requires mechanisms to detect the information and possibly appropriate feedback to the user when problems are encountered. This categorisation of context is more extensively motivated in [19].

The stereotypes that can be applied in the application model are shown in Figure 3, while Figure 4 shows an example application model. The example shows a partial application model of a museum guide. It clearly shows that the information that many relations exist between parts of the model that are part of the context and those that are not. It also shows that the location of a user is detected by a LocationDetector, while the location of the museum artifacts is profiled.

## 5 Context Model

The context model specifies the different situations in which an application can be used. For each context of use the context model contains a package with the stereotype «contextOfUse». Such a package can only contain instances of classes that have the stereotype «context» as specified in the application model. As such the context model is more open than the context model used in UsiXML [11], which uses instances of predefined classes to specify the contexts of use.

Each instance specifies one value to which a parameter of the context of use has to adhere. Ranges of values can be specified by specifying a minimum and a maximum (using the corresponding stereotypes), or by listing the possible values; when multiple instances of the same class are specified they represent alternatives. To avoid ambiguity, when both a minimum and a maximum value is provided, the involved instances should be linked. Figure 5 shows the stereotypes that can be applied to the model elements, while Figure 6 shows a small

Task Category	ConcurTaskTrees	Contextual ConcurTaskTrees	CUP
Abstract task			/
User task			
Contextual User Task	/		
Application task			
Contextual Application task	/		
Interaction task			
Contextual Interaction task	/		
Environment task	/		

**Table 1.** Icons of task categories in ConcurTaskTrees, Contextual ConcurTaskTrees and CUP 2.0

example model, demonstrating the usage of the different stereotypes. The specified context of use is relevant for users that follow a dynamic tour through the museum and have a PDA with a certain minimal resolution.

## 6 System Interaction Model

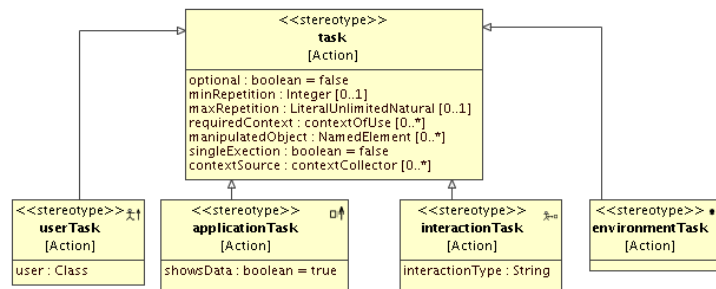
The system interaction model describes the interactions of the system with the user and the environment in which it is executed. It can be used to describe the *tasks* of both the users and the application as well as the relevant interaction with the environment in more detail. The basis for the system interaction model is the UML 2.0 activity diagram. In this model, all actions have to have the stereotype «task» or a derived stereotype applied to them.

A task corresponds to an UML Action. The task goal can be expressed using a local postcondition, if desired. Basic tasks – tasks that are not refined within the model – belong to four different categories. These categories are based on the categories of the Contextual ConcurTaskTrees [18] notation, an extension of the earlier mentioned ConcurTaskTrees notation that allows for specification of context influences. We defined four stereotypes with the appropriate tagged values, that cover all task categories present in the Contextual ConcurTaskTrees as can be seen in Table 1.

One notable difference is the elimination of the task category type *abstract task*, which is a task that can be refined into tasks that belong to different categories. Since there are a great number of ConcurTaskTrees models that do not follow this definition and a change in semantics would only be confusing, we decided to remove this task category and to use a generic stereotype «task» instead. In practise, this has the consequence that CallBehaviourActions and StructuredActivityNodes have to have the stereotype «task» and not one of the derived stereotypes.

The four stereotypes that correspond to the remaining task categories are:

«userTask» A user task is a task that is performed by the user without direct interaction with the application. A user task can however have indirect impact on an application. E.g. A museum visitor might carry an electronic



**Fig. 7.** Stereotypes of the UML profile CUP 2.0 relevant for the system interaction model

mobile guide while strolling, performing no direct interaction. The electronic guide can however get updates about the position of the user through the use of a positioning system in the museum. This can be modeled by applying the stereotype to an `AcceptEventAction` and specifying an interface to the positioning system in the tagged value *contextSource*. User tasks that are applied to other types of Actions are optional and will not be used during further specification of the system.

«**applicationTask**» An application task is a task performed entirely by the application without user interaction. Examples of such tasks are showing information to a user or performing a computation. When an application task has influence on the platform or the environment, the affected data structures or systems can be indicated through the tagged value *manipulatedObject*. Examples of such influences are putting information in the system paste buffer and triggering an external logger that has an influence on future application execution.





«**interactionTask**» Direct user interaction with an application is modeled with an interaction task. Like the previously mentioned tasks, an interaction task can have effects on the environment which are indicated with tagged values. The type of user interaction is indicated through the tagged value *interactionType*.

«**environmentTask**» An environment task covers all actions that have an influence on the execution of the interactive application but are performed by an entity other than the user and the application. An example of an environment task is a car accident that happens on the route calculated by a car navigation system. Similar to the user task, an environment task will be modelled through an `AcceptEventAction` when it has an immediate effect on the execution of the application, such as in the example of the car accident, which triggers a recalculation of the route.

All stereotypes indicating task categories are derived from the stereotype «**task**», which defines some tagged values that are shared by all task categories. These tagged values are important to reduce the complexity of the diagrams:



the tagged value *optional* indicates whether or not a certain task is required or not, while the tagged value *repetition* indicates the number of times a task should be executed. The tagged values *manipulatedObject* and *requiredContext* are only applicable to basic tasks and thus are required to be empty sets for the stereotype «task». Figure 7 gives an overview of the stereotypes and their tagged values.

Temporal operator	Symbol	Activity diagram constructs
Enabling	>> and $\boxed{\>>}$	control and object flow 
Disabling	$\boxed{\>}$	InterruptableActivityRegion with InterruptionEdge 
Concurrency	and $\boxed{   }$	ForkNode and JoinNode with control or object flows 
Choice	$\boxed{\phantom{>}}$	Decision and MergeNodes with control flows 
OrderIndependent	=	same as concurrency but all tasks have tagged value <i>singleExecution</i> set to true
Interruption	>	concurrency with tagged value <i>singleExecution</i> set to true for the interrupting task

**Table 2.** Temporal operators in ConcurTaskTrees and corresponding activity diagram notation

If the tagged value *singleExecution* is set to true for a certain task, that task interrupts all other tasks that are running in parallel until it is completed. This has as consequence that when all actions following a **ForkNode** have this tagged value set to true, they have to be carried out one after the other. This makes that all temporal operators supported by the ConcurTaskTrees notation can be expressed using the UML activity diagram when the stereotypes in Figure 7 are applied as can be seen in Table 2.

An example of a system interaction model can be seen in Figure 8. The example shows a partial specification of a mobile museum guide that offers different types of tours. The diagram gives only details about one type of tour: the dynamic tour. This type of tour does not offer a specified trajectory to the user, but shows the user’s position in the museum as well as information about a nearby artwork if one is available. A user can ask more information about an artifact. This additional information temporarily blocks all other information. Note that this example is simplified for brevity and as such will not really result in a user-friendly application.

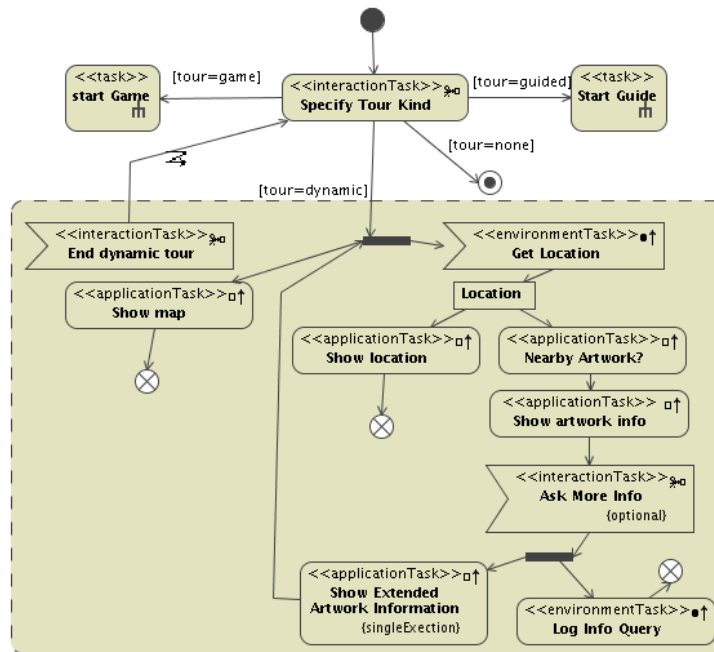
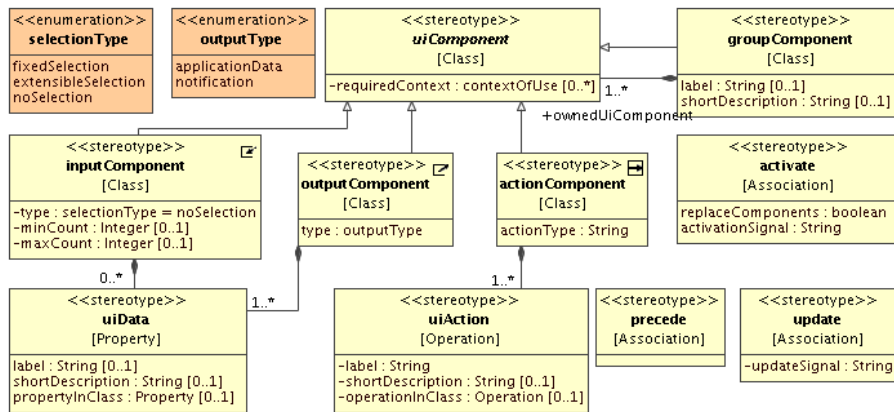


Fig. 8. Example of system interaction model: A context-sensitive mobile museum guide.

## 7 Abstract User Interface Model

The abstract user interface model provides information about the structure of the user interface independent of the platform it will ultimately be deployed on. This means that we abstract from the concrete components and drastically reduce the number of components, coming to a minimal set of kinds of user interface components. The components are differentiated according to the functionality they offer to the user. We identified four types of abstract user interface components: *input components*, which allow users to enter or manipulate data, *output components*, which provide data from the application to the user, *action components*, which allow a user to trigger some functionality, and *group components*, which group components into a hierarchical structure.

In the UML, we represent the abstract user interface model (AUIM) using a class diagram. All classes in a AUIM need to have a stereotype identifying a type of abstract user interface component. There are also restrictions on the associations that can be specified between the classes, they need to indicate containment or have one of the stereotypes discussed later in this section applied to them. The definition of the stereotypes is shown in Figure 9. Only one of these stereotypes can be applied to one class. There is one exception to this rule: a group component can also be an input component, but in this case the input component must be a selection over the contained user interface components.



**Fig. 9.** Stereotypes of the UML profile CUP 2.0 relevant for the abstract user interface model

One should note that the classes with the stereotypes «inputComponent» or «outputComponent» can each have multiple attributes that would each be represented using a separate user interface component in a notation such as the Canonical Abstract Prototypes [3]. Each of the attributes has the stereotype «uiData». The tagged value *propertyInClass* can be used in case there is a reference to a property of a class. Additional meta-information, such as a label or more detailed information can be provided using the remaining tagged values. All Operations related to an action component must have the stereotype «uiAction» that allows to specify information similar to the stereotype «uiData» for each Property of an input component or output component.

The visibility specification for each Property and Operation with the stereotype «uiData» or «uiAction» is adapted to be more relevant to their meaning in the model, but remains consistent with the UML specification:

- public** Public visibility means that the associated part of the user interface is visible to not only the user of the application, but also other persons that might see the user interface. This visibility is, for example, appropriate for the part of a presentation application that shows slides.
- protected** Protected visibility means that the associated part of the user interface is only visible to the user of the user interface. This might mean that the value of an input component with protected visibility is hidden when shown on a public display. An example of user interface components for which this visibility is appropriate are the controls for moving through slides in a presentation application.
- package** Parts of the user interface that have package visibility are only accessible to other parts of the user interface, but are not shown to the users of the user interface. This visibility should be avoided in the abstract user interface model.

**private** Private visibility is used for parts of the user interface whose contents may not be seen by a user without being masked. An example of a user interface component with private visibility is a password field.

We also defined some stereotypes for associations between abstract user interface components to express relationships other than containment. These relationships indicate constraints on the structure of the user interface which are implied by the system interaction diagram and thus reduce the number of *hidden dependencies* within the abstract user interface model. These relationships can also be used to specify relationships between user interface components within the model that are specified in different diagrams. At the same time they also increase *visibility*. The reduction of hidden dependencies is important to effectively support *modification*, a good visibility is also important for *exploratory design* [9].

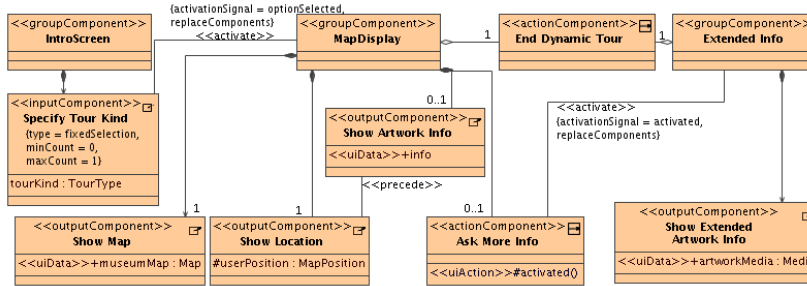
The first stereotype is «**precede**», which indicates that one user interface component should be presented to a user before another user interface component. The precedence can be spacial, temporal or both. The usage of this stereotype is limited to user interface components that are contained by the same group component and can be used to establish an order in which the user interface components are presented to the user. A second stereotype, «**activate**», can be applied to an association to indicate that a user interface component activates another component. The activated components can be added to the currently active components or can replace them. A third stereotype for associations is «**update**». Application of this stereotype to an association indicates that the contents of the target user interface component is updated by the source user interface component.

An example of an abstract user interface model is shown in Figure 10. The depicted model corresponds to the part of the system interaction model that shows the functionality offered in the case of a *dynamic* tour. The figure shows three group components that the user can interact with. The first group component contains one interaction component that allows the selection of a type of tour. When the user selects a type of tour, a second group component is activated and replaces the one that contains the interaction component, as can be seen from the tagged values on the association. This group shows a map, the current user position and, optionally<sup>3</sup> some information about a nearby artwork and, also optional, an option to show more information about the artwork. This information is shown within a group component *Extended Info*, which replaces the group component *MapDisplay*.

The abstract user interface description we use assigns one type of user interaction to a component, similar to the approach taken for XForms [7], UMLi [5] and Wisdom [13]. TERESA XML [12] also uses this approach but defines a deeper hierarchy that contains special components for inputs of simple datatypes and selections based on the number of options. UsiXML [11] only has one type of user interface components having facets that are based on the type of interaction.

---

<sup>3</sup> This can be derived from the multiplicity specified for the containment relations.



**Fig. 10.** Example of abstract user interface model: A context-sensitive mobile museum guide.

CUP-profile	XForms tags
groupComponent	group
- contained number of elements of same type > 1	repeat
uiData in inputComponent,	
- selectionType is none	input
- max. selectionCount = 1	select1
- max. selectionCount > 1	select
uiData in outputComponent	output
uiAction in actionComponent	trigger or submission

**Table 3.** CUP 2.0 stereotyped Elements and XForms counterparts

## 8 Discussion

The profile can be useful for designers to have rather unambiguous and relatively compact models of a context-sensitive interactive application. Nevertheless, the ability to generate some parts of the models and ultimately generate code templates, can help the designer to be more productive. Therefore we explored the possibilities for automation.

We have identified two main areas where transformations as specified in the model-driven architecture [17] can be applied. The first is a model-to-model transformation from the system interaction model to the abstract user interface model. The second is the generation of high-level user interface descriptions from the abstract user interface model. The user interface deployment model can be used to add style to the different user interface skeletons and add some design guidelines specifically for the target platform.

To test the feasibility of the prototype generation, we choose XHTML + XForms [7] as a target language and investigated how the prototype generation could be established. The mapping of the elements in the abstract user interface model to XForms tags is shown in Table 3. A `uiAction` is translated into a submission if a value is specified for the tagged value `operationInClass`, and into a trigger otherwise. In XForms each component can make references to separately defined object structure in instances. This object structure as well as its XML-

Schema can be derived from the tagged value *propertyInClass* of the attributes with a «uiData» stereotype. The fully-qualified name of its class can be used to generate a meaningful hierarchy of xml-tags, while the datatype itself can be used to define the types in XMLSchema.

The effects of the activation of components can be converted to bind tags with the right **relevant** settings. Precedence relations between user interface components are reflected in the order of the corresponding XForms controls in the document. The update relationships can also be translated into bind-tags with the right nodeset and optionally calculate attributes. Conversion of application or context-driven updates to the user interface are more difficult since they cannot be described declaratively in XForms.

## 9 Conclusion

Despite the fact that there is no dedicated support for multimedia applications, as is offered by the MML (see section 2) and that tool support for the proposed transformations is ongoing or planned as future work, we can conclude that the revised UML profile, CUP 2.0, presented in this paper offers some benefits over related approaches. The profile allows a detailed description of both the behavior and structure of the user interface of *context-sensitive* interactive applications using a limited amount of constructs of the UML using regular UML modeling tools that allow metamodel extension through profiles.

The profile also allows a clear specification of all datatypes that are involved, allowing to make optimal use of specifically designed user interface components for complex datatypes on platforms where they are available. Finally, the fact that all information is expressed in UML makes it easier to integrate the user interface specification with the specification of the application core.

**Acknowledgements** This research was partly performed within the IWT project Participate of Alcatel Bell. Part of the research at the Expertise Centre for Digital Media is funded by the European Regional Development Fund (ERDF), the Flemish Government and the Flemish Interdisciplinary institute for Broadband Technology (IBBT).

## References

1. Pedro F. Campos and Nuno J. Nunes. CanonSketch: a User-Centered Tool for Canonical Abstract Prototyping. In *Proceedings of EHCI-DSVIS 2004*, volume 3425 of *LNCS*, pages 146–163. Springer, 2005.
2. Tim Clerckx, Frederik Winters, and Karin Coninx. Tool support for designing context-sensitive user interfaces using a model-based approach. In *Proceedings TaMoDia 2005*, pages 11–18, Gdansk, Poland, September 26–27 2005.
3. 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.

4. J. Coutaz and G. Rey. Foundations for a Theory of Contextors. In *CADUI*, pages 13–34. Kluwer Academic Publishers, 2002.
5. Paulo Pinheiro da Silva and Norman W. Paton. User Interface Modelling in UMLi. *IEEE Software*, 20(4):62–69, July–August 2003.
6. Anind K. Dey, Daniel Salber, and Gregory D. Abowd. A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications. *Human-Computer Interaction (HCI) Journal*, 16(2-4):97–166, 2001.
7. Micah Dubinko, Leigh L. Klotz, Roland Merrick, and T. V. Raman. XForms 1.0. W3C, <http://www.w3.org/TR/2003/REC-xforms-20031014/>, 2003.
8. Mohammed Elkoutbi, Ismail Khriess, and Rudolf Keller. Automated Prototyping of User Interfaces Based on UML Scenarios. *Automated Software Engineering*, 13(1):5–40, January 2006.
9. Thomas Green and Alan Blackwell. *Cognitive Dimensions of Information Artifacts: a Tutorial*, 1.2 edition, October 1998.
10. Jason I. Hong and James A. Landay. An architecture for privacy-sensitive ubiquitous computing. In *Proceedings of MobiSYS'04*, pages 177 – 189. ACM Press, 2004.
11. 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.
12. Giulio Mori, Fabio Paternò, and Carmen Santoro. Design and Development of Multidevice User Interfaces through Multiple Logical Descriptions. *IEEE Transactions on Software Engineering*, 30(8):507–520, August 2004.
13. Nuno Jardim Nunes. *Object Modeling for User-Centered Development and User Interface Design: The Wisdom Approach*. PhD thesis, Univ. da Madeira, 2001.
14. Object Management Group. *UML 2.0 Superstructure Specification*, October 8 2004.
15. Fabio Paternò. *Model-Based Design and Evaluation of Interactive Applications*. Springer, 2000.
16. Andreas Pleuss. MML: A Language for Modeling Interactive Multimedia Applications. In *Proceedings of Symposium on Multimedia*, pages 465–473, December 12–14 2005.
17. Kurt Stirewalt. *MDA Guide Version 1.0.1*. World Wide Web, <http://www.omg.org/docs/omg/03-06-01.pdf>, 2003.
18. 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.
19. Jan Van den Bergh and Karin Coninx. Towards Modeling Context-Sensitive Interactive Applications: the Context-Sensitive User Interface Profile (CUP). In *Proceedings of SoftVis '05*, pages 87–94, New York, NY, USA, 2005. ACM Press.