CAP3: context-sensitive abstract user interface specification

Peer-reviewed author version

VAN DEN BERGH, Jan; LUYTEN, Kris & CONINX, Karin (2011) CAP3: context-sensitive abstract user interface specification. In: Proceedings of the 3rd ACM SIGCHI symposium on Engineering interactive computing systems. p. 31-40..

DOI: 10.1145/1996461.1996491
Handle: http://hdl.handle.net/1942/12148

CAP3: Context-Sensitive Abstract User Interface Specification

Jan Van den Bergh Kris Luyten Karin Coninx
Hasselt University - tUL - IBBT
Expertise Centre for Digital Media
Wetenschapspark 2
3590 Diepenbeek, Belgium
{Jan.VandenBergh,Kris.Luyten,Karin.Coninx}@uhasselt.be

ABSTRACT
Despite the fact many proposals have been made for abstract user interface models it was not given a *detailed* context in which it should or could be used in a user-centered design process. This paper presents a clear role for the abstract user interface model in user-centered and model-based development, provides an overview of the stakeholders that may create and/or use abstract user interface models and presents a modular abstract user interface modeling language, CAP3, that makes relations with other models explicit and builds on the foundation of existing abstract user interface models. The proposed modeling notation is supported by a tool and applied to some case studies from literature and in some projects. It is also validated based on state-of-the-art knowledge on domain-specific modeling languages and visual notations and some case studies.

Author Keywords
User interface design, modeling language, abstract user interface, CAP3, graphical notation.

ACM Classification Keywords
D.2.2 Design Tools and Techniques: User interfaces.

General Terms
Design, Documentation, Human Factors, Languages.

INTRODUCTION
Abstract user interfaces have been discussed in literature for almost 20 years, although the definition of what constitutes an abstract user interface (AUI) has evolved over time together with advances in technology and extension of the scope. This evolution is illustrated by the fact that Limbourg [13, chapter 2] gives two different definitions for abstract user interface model based on the kind of abstraction that is used; abstraction from toolkit or abstraction from interactor type.

EICS'11, June 13–16, 2011, Pisa, Italy.

In this paper, we will focus on the latter, most recent interpretation of abstract user interface. The former is nowadays usually referenced as concrete user interface.

Despite this long usage of abstract user interface models, there does not seem to be convergence on the language of choice. The reasons for this may be diverse. One reason may be that the needs in different domains may differ significantly. One domain may benefit from a large vocabulary (e.g. interactive web applications), while another domain may prefer a small set of abstractions (e.g. participatory television [25]). Another reason may be that the focus of many research efforts was not the model itself, but the potential benefits for forward or backward engineering (in a specific domain), leading to incomplete descriptions of the modeling language itself in the publicly available literature. Standards such as XForms [5] and ISO 24752:2008 [10] on the other hand are complete but very specific for their domain (websites and universal remote controls).

In this paper we propose the modeling notation (concrete syntax) of CAP3, a new abstract user interface modeling language that builds on previous work. It integrates structural and behavioral specification (similar to what Denim [14] does for low-fidelity user interface sketches). CAP3 includes explicit references to models related to the abstract user interface model, such as domain model, user model or context model. CAP3 refers to the fact that CAP can be interpreted in three different ways that are all appropriate for the modeling language. The first is Canonical Abstract Prototypes; the abstract user interface notation from which most of its concrete syntax is derived. The second is Context-sensitive Abstract Prototypes, which refers to the fact that it can be used to express context-sensitive user interfaces on an abstract level. The third is Configurable Abstract Presentation, highlighting the goal of a configurable notation composed of a core set of symbols and a library of default symbols that can be used when appropriate.

Before introducing the details of CAP3 itself, we present related work and how CAP3 can be used in a user-centered software engineering approach using the MuiCSer process framework [9]. CAP3 is validated using knowledge about domain-specific modeling and visual languages and by discussing how it was applied or could have been applied in some projects.

RELATED WORK

MARIA [23] is an XML-based user interface description language that can be used to describe task models, abstract user interface models and concrete user interface models for service-oriented applications. The description language can be used both at design-time (through a custom editor) and at runtime. The abstract user interface model allows a specification of the dialog (dynamic behavior), the presentation and references to the manipulated data (datatypes are defined ad-hoc through XML Schema [4].) The dynamic behavior is expressed using events, ConcurTaskTrees [18] temporal operators such as concurrency, mutual exclusive choice or sequentiality and event handlers. The presentation of a user interface is described using a composition of interactors. Most of the interactor types are further refined depending on their role or the type of data they are manipulating[1]. The MARIA editor uses an enhanced tree control to modify abstract user interface models and the interactors are abstractions of widgets (as opposed to activities).

UsiXML [13, chapter 3] is also an XML-based user interface description language that can describe user interface models at different levels of abstraction. One of these models is the abstract user interface model. It only discerns two types of abstract interaction objects (*aio*): *abstractContainer* and *abstractIndividualComponent* (aic). Instead of defining different subtypes of aic, it can contain multiple facets but discerns four different types of facets exist (see Table 1). Five different types of relations can exist between the abstract interaction objects: *abstractContainment*, *abstractAdjacency*, *spatioTemporal*, *auiDialogControl* and *mutualEmphasis*. The UsiXML AUI model has a graphical notation [16] that is limited to the structural aspects of the abstract user interface.

Canonical abstract prototypes [6], CAP, is a graphical abstract user interface description language that was created based on practical experience in industry to ease the transition from task models to concrete user interface prototypes. Instead of focusing on making an abstraction of widgets independent from platform and modality, CAP abstracts user activities. Since it is positioned as an alternative to or refinement of paper prototypes during the user interface design process it does not have any formal meta-model, the semantics of each component is defined clearly, but leaves room for some semantic variation. CAP is restricted to the presentation submodel. One can make annotations, to define some behavioral aspects of the user interface.

The MetaSketch editor [20] demonstrated graphical notations for the two major components of the abstract user interface model: the abstract presentation model and the dialog model. A meta-model for these two submodels was realized by extending the UML metamodel [22]. The graphical notation for the presentation model was derived from the Canonical Abstract Prototypes [6] notation, while a custom notation was created for the dialog model. The semantics for the dialog model were derived from the ConcurTaskTrees (CTT) notation [18] while the symbols in the graphical notation were inspired by the UML activity diagram but retained the hierarchical, tree-like structure of CTT.

UMLi [7] specifies an extension of UML (both abstract and concrete syntax) for the specification of user interfaces. It uses a specific new notation for abstract presentations and an extended version of UML 1.x state machines for the specification of the dialog model. UMLi contains six types of abstract interactors, shown in Table 1. It thus supports a set of interactors that is similar to those used in Teallach [1]. The UMLi dialog specification contains specific control flow constructs and explicitly links interactors (and domain objects) to actions and *FreeContainer*s to activities. The types of links include presentation (*FreeContainer*), interaction (interactors), confirmation, cancellation and activation (*ActionInvoker*).

CUP 2.0 [26] extends the UML 2 meta-model through a set of stereotypes. It uses similar set of abstractions as UMLi but only contains one type of container and the detailed semantics and syntax are significantly different. It also allows to encode richer relations between interactors, such as *update* and *activate*.

ArtStudio [24] has a limited abstract user interface model that consists of *abstract workspaces* that only defines structural properties.

XForms [5] is an official recommendation of W3C that has a similar level of abstraction as abstract user interface models and was created to embed specification of manipulation and presentation of (complex) system state to XML-documents using the Model-View-Controller architecture. It has a rich feature-set with seven different interactors (see Table 2). These interactors can be logically arranged in *group*s. Specific structures for conditional or repeated display of components are also provided. Fifteen different kinds of actions are provided to handle the effect of specific events on the user interface.

ISO 24752 [10] has a much more limited scope (universal remote controls) and defines a set of interactors that resembles that of XForms. Similar to MARIA, there is also a number of interactors that are aimed to manipulate a certain type of data.

Other approaches, such as PUC [19] or Supple [8], focus more on a description of the state of the application and the commands that can be given to it, complemented with the appropriate labels, to generate appropriate user interfaces and thus differ significantly from the above-mentioned abstract user interface descriptions, although they also describe the user interface on an abstract level.

AUI IN A USER-CENTERED PROCESS

The role of abstract user interfaces in a development process has been discussed in the context of the unifying reference framework by Calvary et al. [3] or as part of usage-centered design [6]. Calvary et al. mention that AUI are an intermediary model to develop multi-device user interfaces. In many

[1] A detailed overview of the interactors can be found in Table 1.

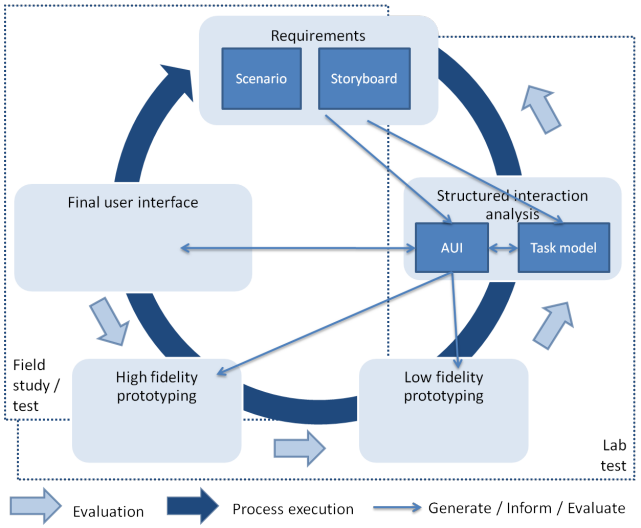


Figure 1. The role of AUI using the MuiCSer process framework.

approaches it is used as a step between the task model (and related models) and the final user interface as discussed by Limbourg [13, chapter2].

In usage-centered design, Canonical Abstract Prototypes (CAP) are used to ease the transition from very abstract specifications to concrete user interface designs. They mention that CAP's graphical notation is useful because it allows to specify the layout as well as the functionality provided by the user interface. This does not mean however that concrete designs must completely follow the layout of the CAP diagrams, but it shows some possibility to use this property to generate pleasing default layouts for the final user interface.

Figure 1 further clarifies the role that an AUI plays in different approaches described in literature by positioning into the MuiCSer process framework [9] for user-centered software engineering. This process framework emphasizes both aspects from user-centered design, such as user evaluations and prototypes at different scales of fidelity, and aspects from (model-based) software engineering, such as structured specifications in e.g. the form of models. An AUI is used as a means to do structured analysis, during the generation, creation or evaluation of prototypes and the final user interface. Some approaches described in literature also generate an AUI directly from an existing final user interface and then use the resulting model to create a user interface for a different platform or interaction paradigm.

Figure 1 also shows that (potential) users of the AUI can have diverse profiles. The professionals involved in *creating* AUI can be interaction designers, information architects, software engineers or developers. Other *users* of the models include designers, who establish concrete designs, and finally also clients and managers, who review the models. This means that the concrete syntax of an AUI should be accessible and readable to people from very diverse backgrounds.

CAP3

Structural Specification

Our CAP3 notation and meta-model builds upon CAP [6] for the specification of the structural aspects. One of the reasons for this decision is motivated that usability of the notation was important from the earliest design of the language, which is important when considering the diverse backgrounds from the potential users of the language as discussed in the previous section.

Another reason to choose for CAP as an AUI is its expressiveness. Table 2 and Table 1 show a comparison of the abstract interactors supported by the different languages. Each row in the Table refers to another type of abstract interactor. For all languages that have a (partial) graphical notation, the symbols for the different kinds of abstract interactors (and facets) are shown in Table 1. The icons for CAP and CUP 2.0 are added in the top-left corner of a rectangle that indicates the actual abstract interactor. The icons for the abstract interactors and facets of UsiXML are shown in the lower-left corner of rounded rectangles. The outline of the rounded rectangle of abstract interactors and facets differs in texture. The symbols for UMLi are the full representation of the abstract interactors; they are not contained in any bounding box.

The columns with vertical text for CAP and MARIA indicate more generic types that are available. In CAP, these more generic types (*tool*, *container*, *active material*) can be used directly in a diagram. It is however unclear whether MARIA allows the more generic types (such as *only_output*) to be used directly in a model. This unclarity and other unclarities in the support for an abstract interactor are indicated by a gray background in the Table. The vertical text *abstractIndividualComponent* in Table 1 is added for UsiXML to indicate that all facets are part of the *abstractIndividualComponent*.

One limitation of CAP is that it has no real knowledge about datatypes, although these can be indicated using appropriate naming. Furthermore, CAP offers no dedicated support for a *secret* interactor (used to model e.g. a password field) nor does it make a distinction between single selection and multiple selection. All these issues can be resolved when transforming CAP from a diagramming notation into a modeling language by adding the necessary attributes to some meta-classes. The latter two issues can be resolved by adding an attribute *isSecret* to the meta-class *Input* and an attribute *maxSelectedElements* to *SelectableCollection*). Knowledge about datatypes is added in a similar fashion. It is also important to note that one cannot choose freely between *submit* and *trigger* for XForms in Table 2, when considering equivalents for a *tool* in CAP.

Although our proposed language, CAP3, follows the conventions of CAP in most cases, there are a couple of deviations:

Repetition Repetition is indicated in CAP by a downwards pointing triple chevron in an *interactor* that contains other

CAP		MARIA		UsiXML		CUP 2.0	UMLi
tool		*control*		AuiInteractor			
	start/goTo		navigator		Navigator	actionComponent	ActionInvoker
	stop/end/complete						
	perform(&return)						
	view						
	select		activator		Command		
	create						
	delete/erase						
	modify						
	move						
	duplicate						
	toggle						
conceptual group			grouping		AuiContainer	groupComponent	Container, FreeContainer
container				AuiInteractor			
	element	*only_output*			Output	outputComponent	Displayer
			text				
			object				
			description				
	notification		alarm				
			feedback			outputComponent	
	collection		*		*	outputComponent, *	
active material			interactive_ description		*	*	*
	input/accepter						Inputter
	editable element	edit	position_edit		Input	inputComponent	Editor
			text_edit				
			object_edit				
			numerical_edit				
			numerical_edit_ full				
			numerical_edit_ in_range				
	editable collection		*				
	selectable collection	*selection*			Selection		
			single_choice				
			multiple_choice				
	selectable action set		*		Commands or Navigators	*	*
	selectable view set		*		Navigators	*	*

Table 1. Comparison of abstract interactors among languages. * denotes that a construct is supported through a combination of abstract interactors. A gray background denotes an unclear support.

	CAP	Standards: Xforms	Standards: ISO/IEC 24752:2008
tool		submit or trigger+ Action	TriggerInteractor TimedTriggerInteractor
	start/goTo	send	
	stop/end/complete		
	perform(&return)		
	view		
	select	setindex	
	create	insert setvalue	
	delete/erase	delete	
	modify	setvalue	
	move		
	duplicate	copy	
	toggle	toggle	
	conceptual group	group	Group
container			Group
	element	output	OutputInteractor
	notification	Action + message	ModalDialogInteractor
	collection	*	
active material			
	input/accepter	input secret textarea range	InputInteractor SecretInteractor [type]Interactor RangeInteractor
	editable element		
	editable collection	*	
	selectable collection	select	SelectInteractor
		select1	Select1Interactor
	selectable action set	*	
	selectable view set		

Table 2. Comparison of abstract interactors between CAP and two standardized languages: XForms and ISO 24752-3:2008. * denotes that a construct is supported through a combination of abstract interactors. A gray background denotes an unclear support.

interactors. It is however not clear whether the repetition applies to all contained interactors. In Figure 7(a), for example, it is not clear whether only the information about the film should be repeated or also the *tools* that allow to add or remove film clips. To make it easier to discern that certain elements are repeated they are all contained in a *RepeatedConceptualGroup*, which is depicted by a dashed rectangle (in common with a CAP conceptual group) with a triple downpointing chevron below it. Figure 7 shows this difference between the CAP notation and the CAP3 notation in the interactor *Film Clips*.

Presentation units A presentation unit, which corresponds to a window in a graphical user interface, is depicted in CAP with a rectangle with a flipped corner. In CAP3, every interactor that is not contained in another interactor can indicate a presentation unit. In most cases, this top-level interactor will be a container (as is the *Film Clip Viewer* in Figure 7), but is not necessarily the case. User interfaces on e.g. smart phones may at times only display a single interactor at a time. In CAP3, this single interactor can be a top-level interactor. An example of this is shown in Figure 8.

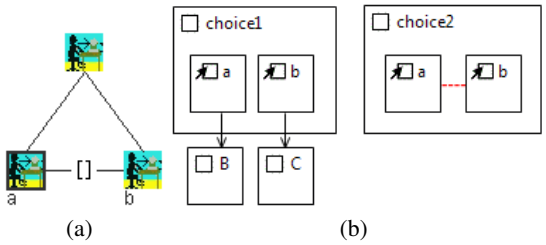


Figure 2. Choice representation using CTT (a) and CAP3 (b).

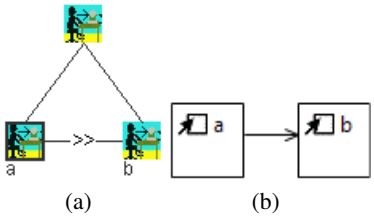


Figure 3. Enabling representation using CTT (a) and CAP3 (b).

In order to keep the language as lean as possible, only the three base interactors (*Tool*, *Container* and *ActiveMaterial*) are considered as required interactors. All other interactors are optional and can be considered part of a standard library rather than part of the language itself. This choice does not conflict with ideas from CAP, which also state that these three interactors can replace all other interactors (see Table 1). Furthermore, Gayos et al. argue that the resulting lack of a selection interactor, which is also the case in their modeling language, can even be beneficial [8].

Similarly additional symbols, such as the *ParticipantElement* and *ActiveParticipantCollection* as used in SPieLan [25] could be placed in a specific library complementing the interactors provided within CAP3 itself. Given appropriate tool support, this enables the creation of reusable modeling language extensions similar to the way reusable extensions are created for programming languages; through the creation of additional libraries.

Behavior Specification

Before adding a dynamic behavior specification to CAP, we considered it good practice to first consider how the dynamic behavior is specified in other abstract user interface languages. UsiXML [13], MARIA [23], and Nobrega et al. [20] all use a similar set of temporal operators. This set is based on, but different from the temporal operators of LOTOS [2]. Differences with LOTOS include, but are not limited to the introduction of new operators such as suspend/resume ($|>$) and deterministic choice (π). UMLi [7] defines its own set of temporal operators but is explicitly mapped to LOTOS specifications.

In this section we introduce how the behavior of the most commonly used temporal operators can be specified in the proposed notation. Each proposal is compared with a minimal task specification with LOTOS operators. We do not

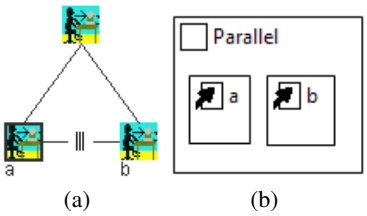


Figure 4. Parallel representation using CTT (a) and CAP3 (b).

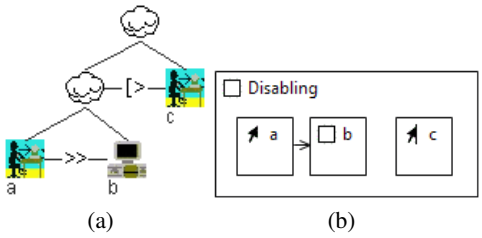


Figure 5. Disabling representation using CTT (a) and CAP3 (b).

discuss the unary optional operator from CTT as an optional interaction task is probably always represented in the user interface, unless a specific constraint is attached. At this moment, we do not consider a specific graphical notation for this operator. The repetition operator is only represented in the graphical notation when it results in a repetition of the interactors, which is not always the case. For example, in the AUI represented in Figure 7, the *Find* interactor may be modeled in a task model as a repeatable task, but it will only be represented using a single interactor.

Figure 2 shows how the choice operator is represented in ConcurTaskTrees notation (CTT) and CAP3. Figure 2(b) shows two options to represent a choice, depending on whether a transition between states is invoked by *a* and *b* or there is no dialog transition as a consequence of an invocation of task *a* or *b* as in Figure 2(a). The former can happen when the parent task of *a* and *b* enables another task or when *a* and *b* are further decomposed into two or more subtasks connected by an enabling operator.

Notice that for both cases we use a notation inspired on UML [22]. The activation of another abstract interactor is based on the UML notation for state transitions, while the notation used in *choice2* is based on the xor-relation, which is used in the UML version of CTT used by Nunes [21]. Instead of relying on a textual notation, we enhance the difference with the other relations through the usage of color (red). We thus increase the number of visual variables we use in the notation, which should make it easier to recognize the relation.

The *Activate* relation (Figure 3(b)) corresponds to the enabling operator in CTT (see Figure 3(a)). Notice that this relation can also be used within a *Container* to denote intra-dialog transitions (see Figure 5(b)) or from a *Tool* or *Active-Material* contained in a *Container* to an abstract interactor outside that *Container* (see Figure 2(b)).

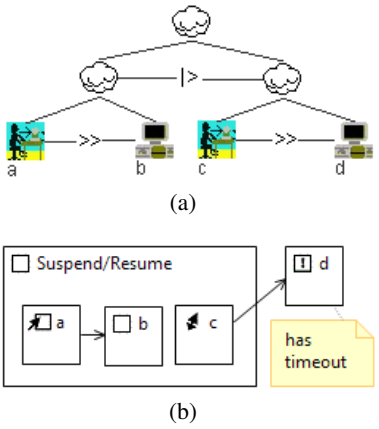


Figure 6. SuspendResume representation using CTT (a) and CAP3 (b).

The parallel temporal operator from CTT (Figure 4(a)), as well as the order independence operator are translated into a single relation CAP3 (Figure 4(b)). As this relation is naturally expressed by containment in the same *Container* in CAP, we keep this notation.

The *End* tool from CAP is especially useful when specifying a disabling relationship as shown in Figure 5(b). This tool disables the complete container in which it is directly contained.

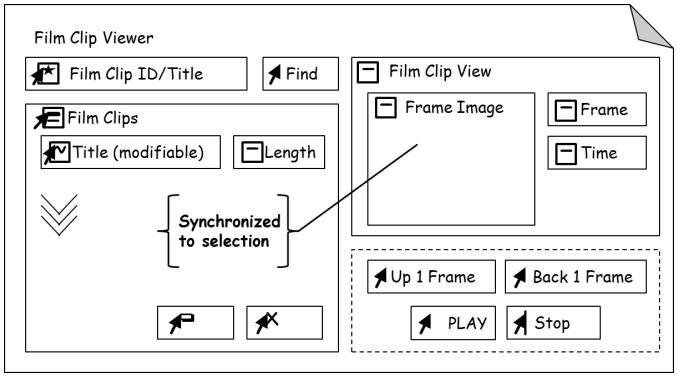
The *PerformReturn* tool is similarly helpful (Figure 6(b)) to specify the suspend/resume temporal operator (Figure 6(a)). Note that reverse direction does not need to be explicitly defined. In Figure 6(b), a notification is shown, which automatically disappears after a certain period. When the notification disappears, the *Suspend/Resume* presentation unit becomes active again.

Inspired by the UML statecharts specification and CAP itself, we decided to use separate notations for information exchange. We still use to convention from CAP to embed a tool in a *Container* to denote that the function in the application core, triggered by the tool *Use* this information. For indicating the other direction, we use an *Update* relation. The notation is inspired on, but different from the UML notation for object flow. We opt for a thicker dotted line and a full arrow head instead of a dashed line and an open arrow head to make the conceptual difference also visually clearer. Figure 7(b) shows several examples of information flow. There is a relationship from the *Find Tool* to the *Film Clips* to indicate the former *update* the latter.

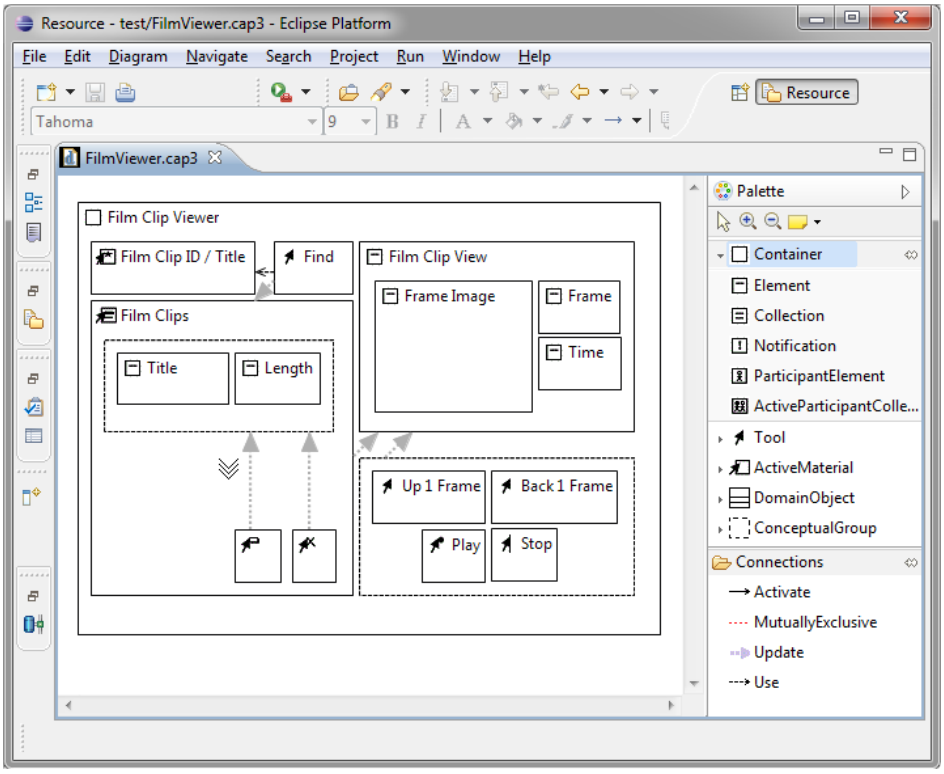
When we compare Figure 7(b) with its original specification in CAP, shown in Figure 7(a)[2], we notice that there is also a difference for the behavior specification: The rather informal notation between parentheses is replaced by an *Update* relationship.

Figure 7(b) also shows another use of a conceptual group; it provides some syntactic sugar to reduce the number of

[2]This diagram is recreated after [6, Figure 2].



(a) Canonical Abstract Prototypes



(b) CAP3 specification in Eclipse-based supporting tool

Figure 7. Film viewer example as specified by Constantine (a) and using our tool support (b).

interactor	data display	data modification	navigation
tool		X	X
start/goTo			X
stop/end/complete			X
perform(&return)			X
view			X
select		X	
create		X	
delete/erase		X	
modify		X	
move		X	
duplicate		X	
toggle		X	
container	X		
element	X		
notification			
collection	X		
active material	X	X	
input/accepter	X	X	
editable element	X	X	
editable collection	X	X	
selectable collection	X		
selectable action set			
selectable view set			

Table 3. Capabilities of the CAP3 interactors.

Update relations. An *Update* relation leaving a conceptual group could be replaced by *Update* relations from each of the tools or active materials it contains to the target of the *Update* relation. This convention is used to avoid that all tools in the lower-right corner would have an *Update* relation with the *Film Clip View*.

Table 4 summarizes the discussion of the relations (except containment) in CAP3. Empty cells in the Table indicate that no constraints are defined. Table 3 lists the CAP3 interactors and their capabilities (data display, data modification and navigation).

These enumerations are the complete subset of elements as defined in CAP. Other interactors that are not part of CAP, can also belong to these groups of interactors. The *ParticipantElement* and *ActiveParticipantCollection* as defined in SPieLan [25], for example, also belong to the interactors that show data; they are specialisations of respectively the *Element* and *Collection* interactors.

Relations to Other Models

With the increasing adoption of context-sensitive user interfaces, multi-user interfaces (such as for social applications) and mixed initiative interfaces, one should take into account that the state of the user interface can not only respond to input by its user, but also by input from sensors. Since this influence is independent of the kind of interactor, it can also be included at the abstract user interface (AUI) level.

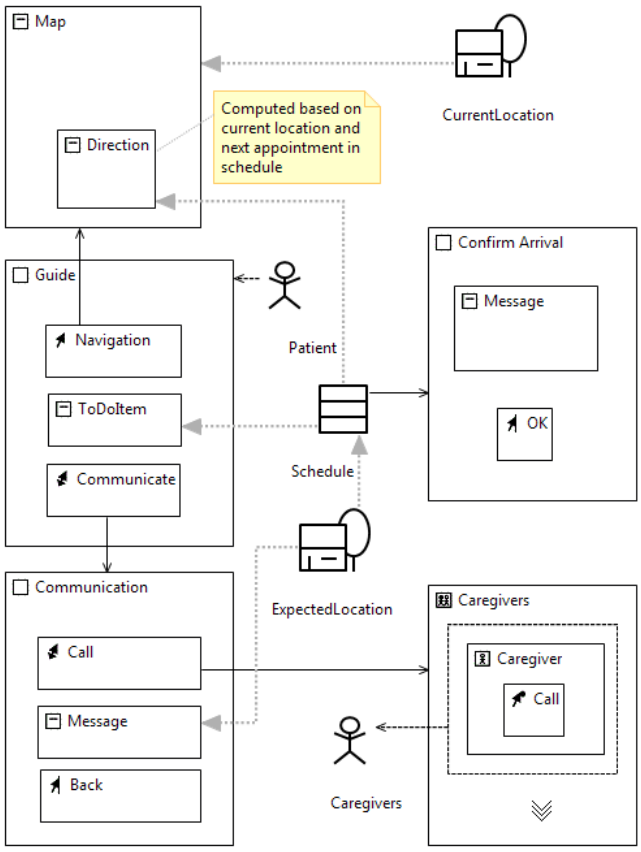


Figure 8. Partial specification of a dementia patient's assistance tool using CAP3.

In CAP3, we model this influence through the integration of proxies to "domain objects". Each proxy contains a reference to an element of another model. Subtypes of domain objects can include concepts of other models and include the human users of the modeled user interface, things and humans in its environment, the environment itself and the data manipulated through the user interface. These subtypes of domain objects are also considered as library objects.

Figure 8 models the system discussed by Mahmud et al. [15] illustrates how CAP3 can express the influence that domain objects and the environment can have on the user interface. It shows that the user's *current location* is shown on the *map* and that the *message* in the user interface of the *communication* module of the application. It shows that a list of caregivers is shown using the *ActiveParticipantCollection* interactor, introduced in SpIeLan [25]. Figure 8 thus demonstrates how CAP3 supports context-sensitivity on two levels: it can be adapted to the modeling context by adding libraries that contain elements that specialize CAP3 elements, and through the explicit coupling with other models using "domain objects" one can model context-sensitive interactive applications. Figure 8 specifies a mobile application that is not only location-aware but also aware of the reachability of caregivers.

CTT Operator	Relation	Source	Target	Parent Interactor
a >> b	Activate	interactor (-) domain object	interactor	
a ||| b, a |=| b	no relation			interactor
a |> b	Activate	PerformReturn	interactor	
a [> b		End		
a [] b	MutuallyExclusive	interactor	interactor	
a*		interactor		RepeatedConceptualGroup (+)
a |[]| b	Update	interactor (*) domain object	interactor (**) domain object	
/				
/	Use	interactor (*) domain object	interactor (**) domain object	
(-) only interactors that can trigger navigation				
(*) only interactors that can modify data				
(**) only interactors that can show data				
(+) only applicable when the interactor should be repeated in the user interface				

Table 4. Relations supported by CAP3, the constraints that apply on the source and/or target of these relations, or on the parent interactor.

The authors believe that this very explicit integration of elements of other models should be limited to cases in which these elements actively influence the user interface without interaction from the user, or when the type of user performing the interaction can differ and this distinction is important (such as in the case of collaborative software). In other cases, a more implicit integration through properties that are displayed as part of the labels or not displayed in the diagram at all seems more appropriate. This latter approach is the only mechanism available in modeling languages such as MARIA [23] and UsiXML [13, chapter 3].

DISCUSSION

The abstract user interface modeling language, CAP3, as proposed in this paper demonstrates the possibility to integrate both structural and behavioral aspects of abstract user interfaces into a single consistent language. To assess the language, its expressiveness is compared to that of several other abstract user interface languages (see for example Table 2 and Table 1 for interactors, and Table 4 for the behavior). CAP3 has only been used to describe graphical user interfaces. These comparisons reveal that it has a similar abstraction level as AUI languages that are also used to semi-automatically generate multi-modal and voice user interfaces, such as MARIA [23].

It was applied to some projects in which the authors were involved or that were described in literature (such as the mobile application discussed by Mahmud et al. [15]).

CAP3 (and other AUI modeling languages) were also assessed using state-of-the-art knowledge regarding the construction of domain-specific languages as discussed by Karsai et al. [11] and the perception of visual languages for software engineering[17]. The results of these assessments were overall positive for CAP3. Nonetheless also some points for improvement were identified; namely inclusion of complexity management mechanisms could still improve the language. But this point for improvement is applicable to many software engineering languages.

Some notable positive points regarding the principles discussed by Moody [17] include that all relations differ in one visual dimension and most in two different visual dimensions (Principle of Perceptual Discriminability), an explicit mechanism for including information from other diagrams (Principle of Cognitive Integration) and the symbols suggest their meaning (Principle of Perceptual Immediacy) to at least a significant subset of CAP3's potential users as identified in the discussion on the role of AUI in a user-centered process.

The knowledge about the design of domain-specific languages was applied as demonstrated by the fact that the potential uses and users of the language were identified before the design of the notation, the language is mostly a composition of existing languages (or language elements), it reuses the existing type system provided by EMF ECore, it allows comments (as shown in Figure 8), and syntactic sugar is used appropriately (only in one case, using the *ConceptualGroup*, where it can clearly reduce clutter).

CONCLUSIONS

This paper presents CAP3, a new abstract user interface modeling language to support user-centered development methods. As state-of-art knowledge on domain-specific languages and visual sofware engineering languages propose, it was not created from scratch, but rather builds on existing notations and modeling languages. It integrates the specification of structural and behavioral aspects of user interfaces into a single notation. An approach that was also taken by Denim [14], which was however focused on informal specification using sketches. Denim thus has no model integration, does not allow the integration of contextual information, and uses sketches instead of precise abstractions.

CAP3 has tool support, which is realized on the Eclipse platform using Eugenia [12]. CAP3 is actively used in research projects and validated using state-of-the art knowledge on domain-specific languages and visual software engineering languages.

Future work includes enhancing the tool support for model-transformations, especially to concrete user interfaces.

ACKNOWLEDGMENTS

This work is supported by FWO project Transforming human interface designs via model driven engineering (G. 0296.08).

REFERENCES

1. P. J. Barclay, T. Griffiths, J. McKirdy, J. B. Kennedy, R. Cooper, N. W. Paton, and P. D. Gray. Teallach - a flexible user-interface development environment for object database applications. *J. Vis. Lang. Comput.*, 14(1):47–77, 2003.

2. T. Bolognesi and E. Brinksma. Introduction to the iso specification language lotos. *Computer Networks*, 14:25–59, 1987.

3. G. Calvary, J. Coutaz, D. Thevenin, Q. Limbourg, L. Bouillon, and J. Vanderdonckt. A unifying reference framework for multi-target user interfaces. *Interacting with Computers*, 15(3):289–308, 2003.

4. W. W. W. consortium. *XML Schema*. `http://www.w3.org/XML/Schema`.

5. W. W. W. consortium. *XForms*. `http://www.w3.org/xforms/`.

6. L. L. Constantine. Canonical Abstract Prototypes for abstract visual and interaction. In *Proc. DSV-IS 2003*, Springer (2003), 1–15.

7. P. P. da Silva and N. W. Paton. User interface modeling in UMLi. *IEEE Software*, 20(4):62–69, 2003.

8. K. Z. Gajos, D. S. Weld, and J. O. Wobbrock. Automatically generating personalized user interfaces with Supple. *Artif. Intell.*, 174(12-13):910–950, 2010.

9. M. Haesen, K. Coninx, J. Van den Bergh, and K. Luyten. Muicser: A process framework for multi-disciplinary user-centred software engineering processes. In *Proc. TAMODIA/HCSE 2008*, Springer (2008), 150–165.

10. ISO JTC 1/SC 35. ISO/IEC 24752-3:2008: Information technology – user interfaces – universal remote console – part 3: Presentation template, 2008.

11. G. Karsai, H. Krahn, C. Pinkernell, B. Rumpe, M. Schindler, and S. Völkel. Design guidelines for domain specific languages. In *Proc. DSM 2009*, 2009.

12. D. S. Kolovos, L. M. Rose, S. bin Abid, R. F. Paige, F. A. C. Polack, and G. Botterweck. Taming EMF and GMF using model transformation. In *Proc. MoDELS 2010 (1)*, Springer (2010), 211–225.

13. Q. Limbourg. *Multi-Path Development of User Interfaces*. PhD thesis, Université Catholique de Louvain, 2004.

14. J. Lin, M. W. Newman, J. I. Hong, and J. A. Landay. Denim: finding a tighter fit between tools and practice for web site design. In *Proc. CHI 2000*, ACM Press (2000), 510–517.

15. N. Mahmud, J. Voigt, K. Luyten, K. Slegers, J. Van den Bergh, and K. Coninx. Dazed and confused considered normal: An approach to create interactive systems for people with dementia. In *Proc. HCSE 2010*, Springer (2010), 119–134.

16. F. Montero and V. López-Jaquero. Idealxml: An interaction design tool. In *Proc. CADUI 2006*, Springer (2006), 245–252.

17. D. L. Moody. The "physics" of notations: Toward a scientific basis for constructing visual notations in software engineering. *IEEE Trans. Software Eng.*, 35(6):756–779, 2009.

18. G. Mori, F. Paternò, and C. Santoro. Ctte: Support for developing and analyzing task models for interactive system design. *IEEE Trans. Software Eng.*, 28(8):797–813, 2002.

19. J. Nichols, B. A. Myers, M. Higgins, J. Hughes, T. K. Harris, R. Rosenfeld, and M. Pignol. Generating remote control interfaces for complex appliances. In *Proc. UIST 2002*, ACM Press (2002), 161–170.

20. L. Nóbrega, N. J. Nunes, and H. Coelho. The meta sketch editor. In *Proc. CADUI 2006*, Springer (2006) 201–214.

21. N. J. Nunes and J. F. e Cunha. Wisdom - a UML based architecture for interactive systems. In *Proc. DSV-IS 2000*, Springer (2000), 191–205.

22. Object Management Group. *UML 2.2 Superstructure Specification*, 2009.

23. F. Paternò, C. Santoro, and L. D. Spano. Maria: A universal, declarative, multiple abstraction-level language for service-oriented applications in ubiquitous environments. *ACM Trans. Comput.-Hum. Interact.*, 16(4), 2009.

24. D. Thevenin. *Adaptation en Interaction Homme-Machine : le cas de Plasticité*. PhD thesis, Université Joseph Fourier, 2001.

25. J. Van den Bergh, B. Bruynooghe, J. Moons, S. Huypens, B. Hemmeryckx-Deleersnijder, and K. Coninx. Using high-level models for the creation of staged participatory multimedia events on tv. *Multimedia Syst.*, 14(2):89–103, 2008.

26. J. Van den Bergh and K. Coninx. Cup 2.0: High-level modeling of context-sensitive interactive applications. In *Proc. MoDELS 2006*, Springer (2006), 140–154.