USIXML: A User Interface Description Language for Context-Sensitive User Interfaces

Quentin Limbourg, Jean Vanderdonckt, Benjamin Michotte, Laurent Bouillon, Murielle Florins, and Daniela Trevisan
Université catholique de Louvain (UCL) – School of Management (IAG)
Information Systems Research Unit (ISYS) – Belgian Lab. of Computer-Human Interaction (BCHI)
B-1348 Louvain-la-Neuve, Belgium
Phone: +32-10/478525 – Fax : +32-10/478324
{limbourg, vanderdonckt, michotte, bouillon, florins, trevisan}@isys.ucl.ac.be

ABSTRACT
This paper presents USIXML (USer Interface eXtensible Markup Language), a User Interface Description Language aimed at describing user interfaces with various levels of details and abstractions, depending on the context of use. USIXML supports a family of user interfaces such as, but not limited to: device-independent, platform-independent, modality independent, and ultimately context-independent. This paper consequently details how context-sensitive user interfaces may be specified and produced from the USIXML specifications. USIXML allows specifying multiple models involved in user interface design such as: task, domain, presentation, dialog, and context of use, which is in turn decomposed into user, platform, and environment. These models are structured according to the four layers of the Cameleon framework: task & concepts, abstract user interface, concrete user interface, and final user interface. To support relationships between these models, a model for inter-model mapping is also introduced that cover forward and reverse engineering as well as translation from one context of use to another.

Categories and Subject Descriptors
D.2.1 [**Software Engineering**]: Requirements/Specifications – *elicitation methods (e.g., rapid prototyping, interviews, JAD).* D.2.2 [**Software Engineering**]: Design Tools and Techniques – *user interfaces.* H.2.4 [**Database Management**]: Systems – *transaction processing*. I.3.6 [**Computer Graphics**] Methodology and Techniques – i*nteraction techniques.*

General Terms: Design, Languages, Human Factors.

Keywords: Computing platform, context-aware adaptation, device independent, model-based approach, multi-platform, user interface description language, user interface engineering.

1. INTRODUCTION
Since the dawn of the discipline of Human-Computer Interaction (HCI), people have attempted to define many languages addressing different aspects of user interfaces (UIs). In particular, a lot of effort has been devoted to introduce various UI Description Languages (UIDLs) with various objectives in mind:

- To introduce a UIDL as a specification language.
- To introduce a UIDL as a communication format.
- To introduce a UIDL to express portability in virtual toolkits.
- To support adaptation.
- To support computing-platform independence.
- To support context-sensitivity.
- To support different families of UIs such as multimodal UIs, multimedia, hypermedia, etc.

The main goal of this paper is to define a UIDL that cumulates the previous requirements into one single language. Such UIDLs may pursue various goals:

- Ensuring portability of UIs from one computing platform to another while preserving some consistency between [4] or with the target computing platform [1,18].
- Capturing UI requirements for an abstract definition that remains stable over time [1,5,19].
- Improving the reusability of UI design [11,25].
- Making one UI design for multiple devices, platforms, or appliances. This goal is often referred to as the device-, platform- , or appliance-independence rendering [1,3,8,9,15,16].
- Supporting extensibility and adaptability of UI [10].
- Using a UI description to enable automated generation of UI code [1,3,5,8,9,10,11,15,16,18,19,25].

On top of these goals are added to more goals that are considered uncovered by ongoing initiatives:

- Making one UI design independently of any modality of interaction (e.g., graphical UI, vocal UI, virtual UI, multimodal,…) so that a design a this level may initiate more concrete designs once a particular modality has been selected.
- Supporting the integration of any model used in the UI development process, such as, but not limited to: the context of use, the user, the platform, the environment, the devices used,…
- Expressing explicitly mappings between models and elements when appropriate to address the mapping problem [8].
- Supporting the continuous and seamless manipulation of models from the abstract level to the concrete levels, such as in the Model-Driven Architecture (MDA) of the OMG Group.
- Expressing UIs at a given instant of usage so as to capture relevant information to ensure runtime migration.

Reaching a UIDL that fully addresses all these requirements and encompasses all the properties of interest of all these types of UIs is certainly neither possible nor desirable. Therefore, this UIDL pursues the goal of capturing the essential properties of interest that turn out to be vital for specifying, describing, designing, and developing such UIs. Consequently, this paper will present and motivate the choices that have been made to drive the definition of USIXML, a UIDL addressing the above requirements by defining models involved in this process. The remainder of this paper is structured as follows: section 2 provides a state of the art in the domain of UIDLs addressing partially or totally the above requirements. Section 3 presents the structure of the USIXML language by showing its scope that is wider than some existing UIDLs. Section 4 defines the different models and mappings that constitute USIXML. The original part of USIXML is emphasized when appropriate. Section 5 concludes the paper by bringing up the main benefits of USIXML with respect to other UIDLs.

2. RELATED WORK

It is worth to notice that many initiatives addressing the design of UIs for multiple platforms almost resuscitated the problem of UIDL that was for a time left forgotten after the question of portable UI has been achieved. Consequently, many initiatives for solving the design of UIs for multiple computing platforms or multiple contexts of use simultaneously consider a UIDL and software based on this UIDL to produce various types of UIs.

The PIMA Project [11] aims at producing applications that are device independent. A Platform Independent Application can be created either by a design tool or by abstracting a concrete UI thanks to the generalization process. Generalization is done by reverse engineering [6] the code of the UI. This process starts with the detection of interaction elements. Secondly, the properties and semantic information of these elements can be inferred. A specialized engine with a device profile then creates another application specialized for a particular device.

TERESA [18] produces different UIs for multiple computing platform from a general task model which is progressively refined for the different platforms. Then, various presentation and dialogues techniques are used to map the general specifications expressed into XHTML code for each platform such as web, PocketPC, and mobile phones. The TERESA (Transformation Environment for inteRactivE Systems representAtions) exploits a UIDL called TERESAXML that supports several types of transformations such as: task model into presentation task sets, task model into abstract UI, abstract UI to concrete UI, generation of the final UI. In [25], a very interesting algorithm is provided that maps a hierarchical task model to a presentation model that explicitly takes into account platform characteristics such as screen resolution.

UIML [1] is also a UIDL intended to support the development of UIs for multiple computing platforms by introducing a description that is platform-independent that will be further expanded with peers once a target platform has been chosen. Recently, the TIDE tool introduced transformations from a basic task model.

XIML [8,19] is a UIDL containing mechanisms for specifying any type of model, of model element, and relationships between. Although some predefined models and relationships exist, one can expand the existing set to fit a particular context of use. XIML has been used in MANNA for platform adaptation [8], in VAQUITA to support reverse engineering [3], and in Envir3D to transform a graphical UI into a virtual Ui thanks to mapping tables [3].

WSXL [9] covers UI and Web services that are attached to these UIs. eNode (http://www.enode.com) is also an interesting UIDL in the sense that the dialog part is precisely described, especially at the widget level where abstract events allow a precise definition of any widget behavior. SeescoaXML [21] is the base UIDL exploited in the Seescoa project to support the production of UIs for multiple platforms and the run-time migration of the full UI at run-time. From the same specifications, multiple UIs can be generated for multiple computing platforms.

Many UIDLs exist today that cover similar and different requirements, some of them having been reviewed in [20]. USIXML, the UIDL that is presented in this paper, is similar to the above UIDLs in the sense that is also cover multiples aspects supported by these languages, but it is especially intended for context-sensitive UIs.

3. STRUCTURE OF USIXML

USIXML is structured according to the four basic levels of abstractions defined in the Cameleon reference framework [4] that is intended to express the UI development life cycle for context-sensitive interactive applications (Fig. 1).

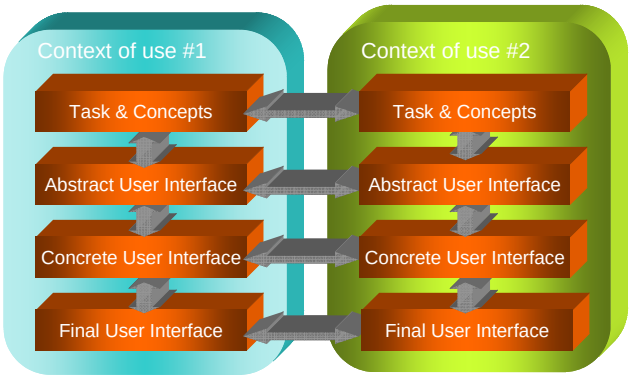

Figure 1. The four basic levels of the Cameleon reference framework [8].

At the bottom is the *Final User Interface* (FUI) related to any UI running on a particular computing platform either by interpretation (e.g., through a Web browser) or by execution (e.g., after compilation of code in an interactive development environment).

A *Concrete User Interface* (CUI) abstracts a FUI into a UI definition that is independent of any computing platform. Although a CUI makes explicit the final look and feel of a FUI, it is still a mockup that runs only within a particular environment. A CUI can also be considered as a reification of a AUI at the upper level and an abstraction of the FUI with respect to the platform.

An *Abstract User Interface* (AUI) abstract a CUI into a UI definition that is independent of any modality of interaction (e.g., graphical interaction, vocal interaction, speech synthesis and recognition, video-based interaction, virtual, augmented or mixed reality). An AUI can also be considered as a canonical expression of the rendering of the domain concepts and tasks in a way that is independent from any modality of interaction. For example, in SEGUIA [24], an AUI is a collection of related presentation units. The relations between the presentation units are inferred from the task relationships expressed at the upper level (task and concepts). An AUI is considered as an abstraction of a CUI with respect to modality.

At the top of the framework is the *Task & Concepts* level where the interactive task to be carried out by the end user is defined according to her viewpoint, along with the various objects that are manipulated by these tasks. These objects are considered as instances of classes representing the concepts manipulated.

A *transient model* [4] is an intermediate model that is required only momentarily during the development life cycle of a FUI. Task-oriented descriptions, AUI and CUI are typical examples of transient models.

On the other hand, *ontological models* [4] are meta-models that are independent of any domain of human activity (e.g., medical domain, surgery, and accounting) and any interactive system. Roughly speaking, they identify key dimensions for addressing a given UI design problem. When instantiated, they give rise to *archetypal models* [4] that are models dependent of an interactive system for a given domain of human activity.

There are three ontological models for context-sensitivity [4]:

- Domain models that support the description of the concepts and user tasks relative to a domain;
- Context models that characterize the context of use in terms of user, platform, and environment. The context model is consequently further decomposed into a user model, a platform model, and an environment model. At least one of these sub-models should be present to build a context model.
- Adaptation models that specify how a UI can be adapted after a change of the context of use that is significant enough to trigger

4. CONTENTS OF USIXML

4.1 Task

A *task model* describes the various tasks to be carried out by a user in interaction with an interactive system. After a comparison of several task modeling technique, an extended version of ConcurTaskTree (CTT) [17] has been chosen to represent user's tasks and their logical and temporal ordering. A task model is therefore composed of *tasks* and *task relationships*.

Tasks are, notably, described with a name, a type (user's, interactive, system and abstract [17]), a task frequency (relative frequency of execution of a task. *Task frequency* is evaluated on a scale from 1 to 5. A value of 1 meaning that a task has a low frequency, 5 meaning that a task is very frequent), a task importance (relative importance of a task with respect to main user's goals. As task frequency, task importance is evaluated on a scale from 1 to 5. A value of 1 means that a task has a low frequency, 5 means that a task is very frequent). Frequency and importance are interesting attributes when it comes to adapt a UI to a constraining context imposing a UI system to be pruned. Finally, an *action type* is based on a taxonomy introduced to better qualify tasks the leave of a task tree. This taxonomy, strongly inspired by [12] (Table 1), is twofold: a verb describes the type of activity at hand; an expression designates the type of object on which the action is operated. By combining these two dimensions a fine derivation of interaction objects supposed to support a task becomes possible.

Task relationships are of two main types: *decomposition* enables to represent the hierarchical structure a task tree hierarchical structure, *temporal* allows specifying a temporal relationship between sibling tasks of a task tree. LOTOS operators are used here.

Action	Item
Start/go, stop/exit, select, choose, create, delete, modify, move, duplicate, toggle, view, monitor	Operation, container, collection, element

Table 1. Taxonomy of action types for tasks.

4.2 Domain

A *domain model* describes the real-world concepts and their interactions as understood by users. Many formalisms have been introduced to represent systems of concepts: frames, semantic networks, entity relationship schemas, class diagrams,… USIXML domain model has the form of a UML class diagram. Concepts contained in USIXML domain model are at a certain point manipulated by users. By manipulated, it is meant that either attribute values are rendered through the UI or that methods attached to classes of objects are used by a user (i.e., triggered by a user event).

Domain model concepts are classes, attributes, methods and domain relationships. A *class* describes the characteristics of a set of objects sharing a set of common properties. A class may contain several attributes and methods. *Attributes* are described with their type, cardinality. Extensive specification of enumerated domains is possible. An original typology allows to characterize the type of domain of an attribute. Indeed, `attribute_domain_characterization` takes the value of: interval, continuous interval, discrete interval, linear interval, circular interval, set[*n*] (where *n* is the number of possible values in an attribute domain). Used in combination with a task model, this typology helps to map domain attributes to a type of interaction object by which it will be rendered. For instance, a choose element task on an attribute with a circular interval enable the derivation of a (multi-state) toggle button. *Methods* are described with their signature i.e., with their name, type, and parameters. A set of predefined method name inspired from OO patterns are used to facilitate the definition of generic design heuristics. For instance, the CRUD pattern is used any method realizing a Create, Read, Update or Delete operation [12]. Finally, *domain relationships* describe various types of relationships between classes. They can be classified in three types: *generalization*, *aggregation*, and *ad hoc*. Class relationships are described with several attributes enabling the specification of role names and cardinalities.

4.3 Context model

A *context model* describes all the entities that may influence carrying out the interactive task of user with the intended UI. It is assumed to capture any relevant attribute of the context of use, in which the user is. A context model consists of:

- A *user model* that recursively decomposes the user population into stereotypes (or profiles) and sub-stereotypes, each stereotype sharing a same series of attributes and associated values.
- A *platform model* captures relevant attributes for each couple software-hardware platform that may significantly influence the context-sensitivity. For instance, screen resolution of the platform is a major property taken into account in adaptation [8] and graceful degradation of UIs [10] when the UI designed for a normal screen is reduced for a more constrained screen. An interesting initiative related to platform modelling is the W3C CC/PP profiles (Composite Capabilities/Preferences Profile). A CC/PP profile is a description of device capabilities and user preferences that can be used to guide the adaptation of content presented to that device. Although CC/PP is not a vocabulary that would permit us to describe a platform, it is a generic XML-based language that allows to write vocabularies peculiar to various platforms. For the purpose of USIXML, we integrated a subset of CC/PP into platform families that are recursively decomposed.
- An *environment model* describes any property of interest of the physical environment where the user is using the UI on the computing platform to accomplish her interactive tasks. Such attributes may be physical (e.g., lighting conditions), psychological (e.g., level of stress), and organization (e.g., location and role definition in the organization chart).

4.4 Abstract User Interface (AUI)

A *AUI model* is a UI model that represents a canonical expression of the renderings and manipulation of the domain concepts and functions in a way that is independent from any modality and computing platform. An AUI is populated by *abstract interaction*

objects and *abstract user interface relationship.*

4.4.1 Abstract Interaction Object (AIO)

An AIO consists of any element populating an *AUI model* consisting in an abstraction of widgets found in most toolkits like windows, buttons but, also, vocal output widget in auditory interface. An AIO is supposed to be independent of any modality of interaction and any platform. AUI types are presented in hierarchy. The more a specification is precise the more the mapping to a concrete object will be precise. AIO are composed of multiple facets. We call them *multi-faceted***.** Each facet describes a particular function an AIO may assume. Four main facets are identified:

1. An **input facet** describes the input type accepted by an AIO.
2. An **output facet** describes what data may be presented to the user by an AIO.
3. A **navigation facet** describes the possible container transition a particular AIO may enable.
4. A **control facet** describes possible methods of the functional core that may be triggered from a particular widget.

An AIO may assume several facets in the same time. For instance, an AIO may display an output while accepting an input from a user, ensure a transition between windows and trigger a method from the functional core.

In order to group AIOs together, the *Interaction Space* is a type of *AIO* that support the execution of a set of logically/semantically connected tasks. An interaction space contains other interaction or other AIO's (see *grouping relationship).* It may be reified into one or more graphical containers like windows, dialog boxes or time slot in the case of auditory user interfaces. It is very important to note that an interaction space in not necessarily reified into a visible object. For instance, an *outputer* (a textbox) and *inputer* (a label) may be grouped together, their possible materialisations i.e., respectively a textbox and an inputer will only be bound together by an internal constraint (not perceivable by the user as is).

4.4.2 Abstract User Interface Relationship (AUI relationship)

An *AUI relationship* is an abstract relationship among *AUI objects* that indicate the existence of some spatio-temporal setting among them (e.g., a navigation between two interaction spaces). Relationships may have multiple source and multiple targets. Two main types of AUI relationships are therefore distinguished: dialog transitions and spatio-temporal relationships.

Dialog Transition is a type of *AUI relationship* that enables to specify a navigation transition between one interaction space and on another or several others with the following possibilities:

- **Suspend**: is a type of *AUI relationship* that enables to specify that the source interaction space is suspended to enable the target interaction space. A (reverse) resume relationship between these interaction spaces must exist for the coherence of the model.
- **Resume**: is a type of *AUI relationship* that enables to specify that the target window is re-enabled after having been suspended by a prior *suspend* relationship.
- **Disables**: is a type of *AUI relationship* that enables to specify that the source interaction space disables the target interaction spaces.
- **Enables**: is a type of *AUI relationship* that enables to specify that the source interaction space enables the target interaction spaces.

Grouping is a type of *AUI relationship* that enables to specify a collection of grouped *AIOs.* The source of a grouping relationship is always an interaction space. Additional information can be specified to precise the nature of the grouping relationship. For instance some ordering may be specified between grouped elements. This ordering can be based on an alphabetical order or a numerical order. Furthermore, it may be specified that grouped element must be specifically differentiated with each other (e.g., by using different colours or dissimilar tone of voices).

At the AUI level, the designer is interested in expressing only high-level relationships between AIOs, if any, without expressing low-level details of the relationships, such as specific distance or time. **Spatio-temporal relationships** characterise the physical constraints between AIOs as they are presented in time and space. Since an AUI does not preclude the usage of any particular modality, we do not know whether a particular AUI will be further reified into a CUI that is graphical, vocal, multimodal, or virtual. Therefore, spatio-temporal relationships should be expressed in a way that is independent of any modality.

For this purpose, the thirteen possible temporal relationships from Allen are considered. Basically, there are two types of temporal relationships (Table 2): sequential (*before* relationship) and simultaneous (that can be *equal*, *meets*, *overlaps*, *during*, *starts*, or *finishes* relationships). Each basic relationship has an inverse relationship, except the *equal* relationship which is symmetric. Although Allen relationships have been introduced to characterise temporal intervals, they are suitable for expressing constraints for space and time thanks to a space-time value. All simultaneous relationships (such as *overlaps*, *during*, *starts*, and *finishes)* can be generalised a the *equal* relationship by inserting some delay time when needed. For example, in the *x before y* relationship, there is a space-time value greater than zero between *x* and *y* while in the *x meets y* relationship the space-time value is equal zero between *x* and *y*. As relationships are abstract at the AUI level, the space-time value is left unspecified until needed at the CUI level. The spatial relationship between *A* and *B* is defined as follows:

Spatial_Composition (*A,B*) = (*Ri* , *Rj)*, where i, j ∈ {1,…,13}, *Ri* is the identifier of the spatial relationship between *A* and *B* according to the *X* axis and *Rj* is the identifier of the spatial relationship between *A* and *B* according to the *Y* axis in the matrix reproduced in Fig. 3. When a spatial arrangement is expressed only according to one dimension, *Ri* = ∅ ou *Rj* = ∅.

The temporal relationship between the *A* and *B* is defined as follows: *Temporal_Composition (A,B)* = (*Ri* , *Rj*), where i, j ∈ {1,…,13} as defined in Fig. 3.

4.5 Concrete User Interface

A CUI is a UI model allowing a specification of an appearance and behavior of a UI with elements that can be perceived by users. A CUI consists of:

- *Modality dependent* i.e., an instance of a CUI addresses a single modality at a time. Two modalities lie in the intended scope of USIXML: graphical and auditory.
- *Platform independent* i.e., elements populating a CUI realize an abstraction of common languages used to program UIs.
  - Concrete Interaction Objects realize an abstraction of widget sets found in popular graphical toolkits (Java AWT/Swing, HTML 4.O, Flash DRK6). A CIO is defined as an entity that users can perceive and/or manipulate (e.g., a push button, a

list box, a check box). Orthogonally to AIOs, CIOs are devided into two types graphicalContainers (e.g., window, panel, table, cell, dialog box,…) and graphicalIndividualComponents (e.g., a button, a text component, an video component, a menu, a spin button,…). In SEGUIA [38], a CUI consists of a hierarchy of CIOs resulting from a transformation of AIOs [37].

- The layout of the CUI is defined without any absolute coordinates. A box embedding mechanisms is used to specify a layout. Alignments between CIOs are defined with a special relationship called alignment.

Fig. 6 shows a simple declaration of a window containing a top-centered label and an OK button.

```
<window id="W1" name="Main Window">
      <box … type = "main" splittable=true detachable=false… >
      <box … type = "horizontal" >
      <textComponent id="TX1" name="Text1" offsetVertical="top" offsetHorizontal="center" defaultContent="Hello World!"/>
      </box>
      <box type="horizontal">
      <button id="B1" name="OkButton" defaultContent="OK" />
      </box>
      </box>
</window>
```

Figure 6. USIXML specification of a window containing widgets.

A CUI is equipped with a mechanism, called dialog, allowing the specification of the dynamic behavior of a concrete user interface. This mechanism covers a navigation definition language and a powerful event/action language. For clarity purpose, we isolated explanations on this aspect. To better understand the differences that exist between AIOs and CIOs in the context of USIXML, Fig. 2 shows that the FUI level is populated by the true final widget in the target platform, e.g., a Download pushbutton written in HTML and rendered on a MacOS X platform (bottom left of Fig. 2). At the FUI level, the HTML source code of this button may remain the same, but can be rendered differently depending on the browser, the platform and other parameters. At the CIO level, the different physical widgets are abstracted from their platform and classified into CIO types. At the AIO level, these objects are successively abstracted from their modality of interaction.

Fig. 2 shows that thanks to these different levels, it is possible to find out alternate CIO or AIO in case of change of the context of use, especially in graceful degradation [16]. Fig. 7 shows alternate CIOs for a menu AIO in the same graphical modality, while Fig. 8 shows alternate AIOs with different modalities.

4.6 Inter-model Mapping

Model integration is a well-known issue in model-based approach of UI development. Rather than proposing a collection of unrelated models and model elements, USIXML provides the designer with a set of pre-defined relationships allowing to map elements from heterogeneous models. This may be useful, for instance, for architecture derivation (mappings between domain and CUI/AUI models), for traceability in the development cycle (reification, abstraction and translation), for addressing context sensitive issues, for improving the preciseness of model derivation heuristics. The mappings between the different models are of several types:

- *Manipulates* maps a task onto a domain concept i.e., a class, an attribute, an operation or any combination of these types. This relationship has an attribute 'centrality' which specifies the relative importance of a domain element to the execution of its corresponding task. This item is evaluated on a scale of 1 to 5. 1 meaning that domain concepts is not central, 5 that is completely necessary (i.e., essential to the execution of the task).
- *Is Rendered By* maps a domain concept onto a presentation element either that a domain concept is subject to user input or that it is only presented to a user. An attribute of this relationship specifies if the values of the mapped attribute may be updated from the UI or not. If not, values are only visualized.
- *Is Executed In* maps a task onto an AUI or CUI element. It indicates that a task is performed through this (set of) AUI(s) or CUI(s) element(s).
- *Is Abstracted Into* and *Is Reified Into* map AUI and CUI elements. This relationship indicates that an element has been derived, through reification or abstraction (see framework of Fig. 1), from another.
- *Has Context* maps any model element to one or several contexts of use.
- *Corresponds To* maps a task temporal relationship with a navigation relationship as defined in a AUI or a CUI.

4.7 Dynamic aspects in USIXML

Dynamics of USIXML cover several aspects:

- **Requirements derivation (**dubbed **reification)** along our development cycle structure (Fig. XX). By requirement derivation it is meant the changing or translation of a high-level requirement into a form that is appropriate for low-level analysis or design.
- **Reverse engineering (**dubbed **abstraction)** along our development cycle (see Fig. 1). By reverse engineering it is meant the extraction of high-level requirement from a set of low-level requirements artifacts or from code.
- **Context (of use) adaptation (**dubbed **translation).** The context of use is defined as a triple of the form *(e, p, u)* where *e* is an possible or actual environments considered for a software system, *p* is a possible or actual target platform, *u* is a user category. Context adaptation is a process of modifying a user interface in consequence of a change of one or several element of the triple described above.
- **Dialog specification** of the user interface. Dialog can be defined as a description of user interface state change along with the event/action specification resulting in a state changes.

The three first items are referred with the generic term of **transformation**. Both transformation and dialog are specified using transformation systems. Transformation systems rely on the theory of graph grammars [20]. We first explain what a transformation system is. Transformations and dialog specification are, then, further explained.

4.7.1 Transformation systems

The proposed formalism to represent model transformation and dialog in USIXML is graph transformation. This formalism has been discussed in [19]). USIXML has been designed with an underlying graph structure. Consequently any graph transformation rule can be applied to a USIXML specification. This formalism conveniently applies to model transformation and dialog specification.

A transformation system is composed of several transformation rules. Technically, a rule is graph rewriting rule equipped with negative application conditions and attribute conditions [20].

Fig. 9 illustrates how a transformation system applies to a USIXML specification: let G be a USIXML specification, when 1) a Left Hand Side (LHS) matches into G and 2) a Negative Appplication Condition (NAC) does not matches into G (note that several NAC may be associated with a rule) 3) the LHS is replaced by a Right Hand Side (RHS). G is resultantly transformed into G', a resultant USIXML specification. All elements of G not covered by the match are considered as unchanged. To add to the expressive power of transformation rules, variables may be associated to attributes within a LHS. An expression may compare this variable with a constant or with another variable. This mechanism is called 'attribute condition'.

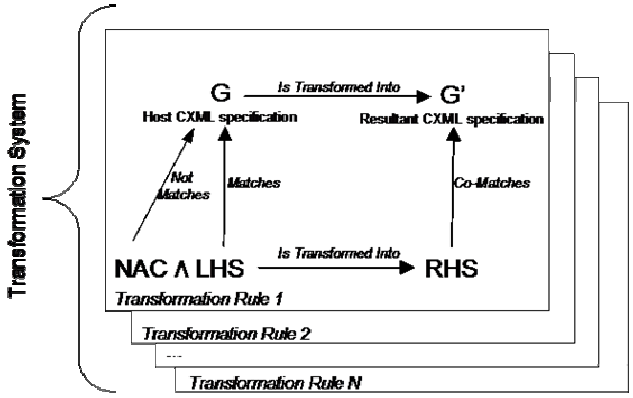

Figure 9. Characterisation of a transformation in USIXML.

4.7.2 Dialog specification

USIXML is equipped with a concrete dialog model. This dialog model is integrated into the concrete user interface model. For illustration purpose we have isolated the dialog parts from the CUI model. The basis of our dialog is an event/action language. Every concrete interaction object may be associated with one or several behavior specification. A behavior is a couple event/action.

- Events may be composite (composed by other events) allowing the expression of complex expressions.
  - Events may be chosen within a predefined event language (Table 3).
  - Events may be composed with temporal operators. Each operator is represented with its symbol: >> is a sequence, ||| is order independence, OR is a disjunction, (n) is an iteration where n is the iteration factor.
- Actions are performed by transformation systems. Transformation systems are sets of transformation rules operating on a specification. Actions have the expressive power of graph grammars. Graph grammars have been proved very powerful (as powerful as Petri nets) represent the behavior of dynamic systems. Concretely, the result of an action may be any change in the CIO model including the triggering of methods from the domain model. An example is given in Fig. 10.

CIO	Events
All graphical CIOs	movePointer(X,device), pointerOver(X,device), moveOutPointer(X,device), click(X,device), doubleClick(X,device), depress(X,device), release (X,device), dragOver(X,Y,device), dragDrop(X, Y,device), hasFocus(X), lostFocus(X)
graphicalContainer	resize(xFactor,yFactor)
textComponent	change
slider	move(cursor,x)
spin	spinUp, spinDown

Table 3. Events of CIOs.

```
<button …name="ClearButton1"…>
 <behavior>
<event>doubleClick(self,Mouse1)</event>
<action>
<transformationSystem …>
<lhs>
<window … mapId = "M1" name="registerWindow"…>
<textComponent … mapId ="M2" isEditable=true/></window>
</lhs>
<rhs>
<window … mapId = "M1" name="registerWindow"…>
<textComponent…mapId="M2" isEditable=true content=""/></window>
</rhs>
</transformationSystem> </action> </behavior>
…
</button>
```

Figure 10. Event Language in USIXML at the CIO level: Clicking on button 1 erases all editable textComponents of registerWindow

Navigation is part of a dialog specification; consequently it is easily described with dialog elements exposed above. Nonetheless, from previous works [36], we consider navigation definition as a pattern-based activity. USIXML provides an ad hoc relationship to define navigation in a straightforward way: `graphicalContainerTransition`. This relationship type enables to specify an open/close, suspend/resume, minimize/maximize relationship among containers populating an application.

4.7.3 Transformation model

A transformation model has been introduced to represent the possible transformations as defined in the framework of Fig. 1 (i.e., abstraction, reification, translation). Like actions in the behavior specification, transformations are performed via graph transformation rules as introduced in [20]. A model transformation is performed by one or several transformation system. Each transformation system realizes an identifiable design goal (i.e., widget selection, layout, navigation definition,…) in the transformation process. Fig. 12 shows a simple transformation (a translation) consisting in one single rule aligning vertically all widgets of a container. This rule has been design with the graph grammar editor AGG (http://tfs.cs.tu-berlin.de/agg/). Its textual equivalent in USIXML is shown in Fig. 11.

```
…
<translation id= "TL1" name = "squeezeDisplay" description= "this translation
vertically aligns all widgets of a container" >
<sourceModel> cui<sourceModel>
<targetModel>cui<targetModel>
<transformationSystem id = "TR1" name = "Transfo1" … >
<transformationRule id = "rule1" name "squeeze1" >
      <lhs>
            <box  mapId ="M1">
            <graphicalIndividualComponent ruleSpecificId"gi1" mapId =M2>
            </graphicalIndividualComponent>
            </box>
      </lhs>
      <rhs>
            <box  mapId ="M1">
            <graphicalIndividualComponent ruleSpecificId"gi1" mapId =M2
            glueHoriz="left" >
            </graphicalIndividualComponent>
            </box>
      </rhs>
      <nac>
            <box  mapId ="M1">
            <graphicalIndividualComponent ruleSpecificId"gi1" mapId =M2
            glueHoriz="left" >
            </graphicalIndividualComponent>
            </box>
      </nac>
</transformationRule>
</transformationSystem>
</translation>
…
```

Figure 11. Translation expressed in USIXML.

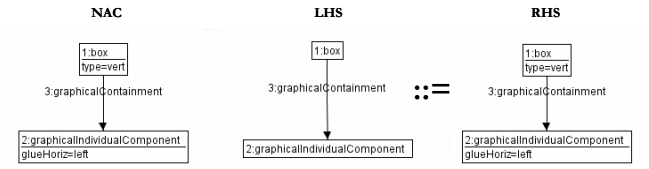

Figure 12. Aligning rule expressed in AGG in terms of a transformation.

5. CONCLUSION

In this paper, we have presented USIXML, a UIDL that addresses various requirements in UI design such as coverage of multiple models, relationships, and levels of abstraction. Graph transformations are explicitly used to define an executable mapping mechanism between these fragments so as to support continuous and seamless development of UIs from multiple entry points. USIXML is original and different with respect to existing UIDL regarding the following aspects:

- USIXML is precisely structured into four levels of abstraction that do not need all to be specified to obtain a UI.
- USIXML can be used to specify a platform-independent, a context-independent, and a modality-independent UI. For instance, a UI that is defined at the AUI level is assumed to be independent of any modality and platform. Therefore, it can be reified into different situations. Conversely, a UI that is defined at the CUI level can be abstracted into the AUI level so as to be transformed for another context of use.
- USIXML allows the simultaneous specification of multiple facets for each AIO, independently of any modality.
- USIXML encompasses a detailed model for specifying the dynamic aspects of UI based on productions (right-hand side, left-hand side, and negative conditions) and graph transformations. These aspects are considered as the basic blocks of a dialog model that is directly attached to the CIOs of interest, thus facilitating the local specification.
- Thanks to these dynamic aspects, virtually any type of adaptation can be explicitly specified. In particular, a transformation model consisting of a series of adaptation rules can be specified equally in an integrated way with the rest of the UI.
- USIXML contains a simplified abstraction for navigation based on windows transitions, that is compatible with dynamics.
- USIXML is based on Allen relationships for specifying constraints in time and space at the AUI level, that can be in turn mapped onto more precise relationships at the CUI level. These relationships are applicable to graphical UIs, vocal UIs, multimodal UIs, and virtual reality UIs.
- Similarly, a progressively more precise specification of the CIO layout can be introduced locally to concretize the Allen constraints imposed at the AUI level.
- USIXML defines a wide range of CIOs in different modalities of use so as not to be limited only to graphical CIOs.
- USIXML already introduced a catalogue of predefined, canonical inter-model mapping that can be expanded and a taxonomy of task types that facilitate the identification and selection of concepts at both the AUI and CUI levels.

From these advances, we can conclude that USIXML is probably one of the mostly integrated UIDL that addresses platform-, modality-, and context-independence and sensitivity. Depending on the kind of UI that is envisioned, USIXML can be used to specify only those parts that are required for a specific case.

ACKNOWLEDGEMENTS

The authors would like to thank Cameleon partners who contributed USIXML V1.2: Lionel Balme, Gaëlle Calvary, Cristina Chesta, Alexandre Demeure, Joëlle Coutaz, Jean-Thierry Lechein, Fabio Paternò, Stéphane Raymond, Carmen Santoro, and Youri Vanden Berghe. This paper presents USIXML V1.4, an extension of USIXML V1.2 with dialog model, more inter-model mappings, a context model made up of user, platform, and environment, and the concrete user interface level. Laurent Bouillon is supported by the "Cameleon" research project (http://giove.cnuce.cnr.it/ cameleon.html) under the umbrella of the European Fifth Framework Programme (FP5-IST2). Murielle Florins is supported by "Salamandre" research project (http://www.isys.ucl.ac.be/research/salamandre.html) under convention n°001/4511 of "Initiatives II" research program, Walloon Region (Belgium). Benjamin Michotte is supported by the SIMILAR network of excellence (http://www.similar.cc), the European research task force creating human-machine interfaces similar to human-human communication of the European Sixth Framework Programme (FP6-2002-IST1-507609). Daniela Trevisan is supported by the Mercator project.

REFERENCES

1. Ali, M.F., Pérez-Quiñones M.A., Abrams M., *Building Multi-Platform User Interfaces With UIML*, in A. Seffah & H. Javahery (eds.) Multiple User Interfaces: Engineering and Application Framework, John Wiley and Sons, 2003.
2. Allen, J.F., *Maintaining Knowledge about Temporal Intervals*, Communications of the ACM, Vol. 26, No. 11, November 1983, pp. 832-843.
3. Bouillon, L., Vanderdonckt, J., Chow, K.C., *Flexible Reengineering of Web Sites*, Proc. of 8th ACM Int. Conf. on Intelligent User Interfaces IUI'2004 (Funchal, 13-16 January 2004), ACM Press, New York, 2004, pp. 132-139.
4. Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., Vanderdonckt, J., *A Unifying Reference Framework for Multi-Target User Interfaces*, Interacting with Computers, Vol. 15, No. 3, June 2003, pp. 289-308.
5. Chamberlain, D., Angel Diaz, Dan Gisolfi, Ravi Konuru, John Lucassen, Julie Macnaught, Stephane Maes, Roland Merrick, David Mundel, TV Raman, Shankar Ramaswamy, Thomas Schaeck, Rich Thompson, and Charles Wiecha, *WSXL: a web services language for integrating end-user experience*, in Proc. of 3rd Conf. on Computer-Aided Design of User Interfaces CADUI'2002, Kluwer Ac., Dordrecht, 2002, pp. 35-50.
6. Chikofsky, E.J. and Cross, J.H., *Reverse Engineering and Design Recovery: A Taxonomy*, IEEE Software, Vol. 1, No. 7, January 1990, pp. 13-17.
7. Constantine, L., *Canonical Abstract Prototypes for Abstract Visual and Interaction Design*, in Proc. of 10th Int. Workshop on Design, Specification, and Verification of Interactive Systems DSVIS'2003, LNCS, Springer-Verlag, 2003.
8. Eisenstein, J., Vanderdonckt, J., Puerta, A., *Model-Based User-Interface Development Techniques for Mobile Computing*, Proc. of 5th ACM Int. Conf. on Intelligent User Interfaces IUI'2001 (Santa Fe, 14-17 January 2001), Lester, J. (Ed.), ACM Press, New York, 2001, pp. 69-76.
9. Elting, Ch., Zwickel, J.and Malaka, R., *Device-Dependent Modality Selection for User Interfaces – An Empirical Study*, in Proceedings of 6th Int. Conf. on Intelligent User Interfaces IUI'2002 (January 13-16, 2002, San Francisco), ACM Press,

New York.
10. Florins, M., Vanderdonckt, J., *Graceful Degradation of User Interfaces as a Design Method for Multiplatform Systems*, in Proc. of 8th ACM Int. Conf. on Intelligent User Interfaces IUI'2004 (Funchal, 13-16 January 2004), ACM Press, New York, 2004, pp. 140-147.
11. Gaeremynck, Y., Bergman, L.D., Lau, T., *MORE for Less: Model Recovery from Visual Interfaces for Multi-Device Application Design*, in Proc. of ACM Int. Conf. on Intelligent User Interfaces IUI'2003 (Miami, January 12-15, 2003), ACM Press, New York, pp. 69-76.
12. Larman, C., *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process*, Prentice Hall, July 2001.
13. Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, B., *TOMATOXML, a General Purpose XML Compliant User Interface Description Language*, TomatoXML V1.2.0, Working Paper n°105, IAG, Louvain-la-Neuve, 19 February 2004.
14. Limbourg, Q., Vanderdonckt, J., *Transformational Development of User Interfaces with Graph Transformations*, Proc. of 5th Int. Conf. on Computer-Aided Design of User Interfaces CADUI'2004 (Madeira, 14-16 January 2004), Kluwer Academics Pub., Dordrecht, 2004.
15. Luyten, K., Van Laerhoven, T., Coninx, K., Van Reeth, F., *Runtime Transformations for Modal Independent User Interface Migration*, Interacting with Computers, Vol. 15, No. 3, 2003, pp. 329-347.
16. Nichols, J., Myers, B.A., Higgins, M., Hughes, J., Harris, T.K., Rosenfeld, R., Pignol, M., *Generating Remote Control Interfaces for Complex Appliances*, Proc. of the 15th Annual ACM Symposium on User Interface Software and Technology UIST'2002, ACM Press, New York, 2002.
17. Paternò, F., *Model-Based Design and Evaluation of Interactive Applications*, Springer-Verlag, Berlin, 2000.
18. Paternò, F., Santoro, C., *One Model, Many Interfaces*, in Proc. of 3rd Int. Conf. on Computer-Aided Design of User Interfaces CADUI'2002, Kluwer Acad., Dordrecht, 2002, pp. 143-154.
19. Puerta, A. and Eisenstein, J., *Developing a Multiple User Interface Representation Framework for Industry*, in: A. Seffah & H. Javahery (eds.) Multiple User Interfaces: Engineering and Application Framework, John Wiley and Sons, 2003.
20. Souchon, N., Vanderdonckt, J., *A Review of XML-Compliant User Interface Description Languages*, Proc. of 10th Int. Conf. on Design, Specification, and Verification of Interactive Systems DSV-IS'2003, Lecture Notes in Computer Science, Vol. 2844, Springer-Verlag, Berlin, 2003, pp. 377-391.
21. Trevisan, D., Vanderdonckt, J., Macq, B., *Analyzing Interaction in Augmented Reality Systems*, Proc. of ACM Multimedia 2002 International Workshop on Immersive Telepresence ITP'2002 (Juan Les Pins, 6 December 2002), Pingali, G., Jain, R. (Eds.), ACM Press, New York, 2002, pp. 56-59.
22. Vanderdonckt, J., Limbourg, Q., Florins, M., *Deriving the Navigational Structure of a User Interface*, Proc. of 9th IFIP Conf. on Human-Computer Interaction INTERACT'2003 (Zurich, 1-5 September 2003), M. Rauterberg, M. Menozzi, J. Wesson (Eds.), IOS Press, Amsterdam, 2003, pp. 455-462.
23. Vanderdonckt, J., Bodart, F., *Encapsulating Knowledge for Intelligent Automatic Interaction Objects Selection*, in Proc. of the ACM Conf. on Human Factors in Computing Systems INTERCHI'93 (Amsterdam, 24-29 April 1993), ACM Press, New York, 1993, pp. 424-429.
24. Vanderdonckt, J., Berquin, P., *Towards a very large model-based approach for user interface Development*, in Proc. of 1st IEEE Int. Workshop on User Interfaces to Data Intensive Systems UIDIS'99, IEEE Computer Society Press, Los Alamitos, 1999, pp. 76-85.
25. Wong C., Chu H.H. and Katagiri M.A., *Single-Authoring Technique for Building Device-Independent Presentations*, in Proceedings of W3C Workshop on Device Independent Authoring Techniques (St. Leon-Rot, 15-26 September 2002).