USIXML: a User Interface Description Language for Specifying Multimodal User Interfaces

Jean Vanderdonckt, Quentin Limbourg, Benjamin Michotte, Laurent Bouillon, Daniela Trevisan, Murielle Florins

[1] Université catholique de Louvain, School of Management (IAG), ISYS-BCHI
Place des Doyens, 1 – B-1348 Louvain-la-Neuve, Belgium
{vanderdonckt,limbourg, michotte, bouillon, trevisan, florins}@isys.ucl.ac.be
http://www.isys.ucl.ac.be/bchi

Abstract. USer Interface eXtensible Markup Language (USIXML) consists of a User Interface Description Language (UIDL) allowing designers to apply a multi-directional development of user interfaces at multiple levels on independence, and not only device independence. In this development paradigm, a user interface can be specified and produced at and from different, and possibly multiple, levels of abstraction while maintaining the mappings between these levels if required. Thus, the development process can be initiated from any level of abstraction and proceed towards obtaining one or many final user interfaces for various contexts of use at other levels of abstraction. In this way, the model-to-model transformation which is the cornerstone of Model-Driven Architecture (MDA) can be supported in multiple configurations, based on composition of three basic transformation types: abstraction, reification, and translation.

Keywords: context-sensitive user interface, device independence, modality independence, model-driven architecture, model-to-model transformation, rendering independence, user interface description language.

1 Introduction

Many User Interface Description Languages (UIDLs) have been introduced so far that addresses different aspects of a User Interface (UI): AUIML, UIML, XAML, XIML, XUL,…. Depending on the perspective, the UIDL may address portability, device independence, support of multiple computing platforms, user-centered design, iterative and incremental development to name a few. This proliferation results in many XML-compliant dialects that are not (yet) largely used and that do not allow interoperability between tools that have been developed around the UIDL. In addition, there has been little or no integration of the various models that not only addresses the needs of traditional Graphical User Interfaces (GUIs) and Multimodal User Interfaces (MUIs) simultaneously. Some UIDLs have been defined for one of them, but not for both. To address the above challenges and others, we have defined USIXML (User Interface eXtensible Markup Language – see www.usixml.org), a UIDL that is characterised by the following original principles:

- *Expressiveness of UI*: any UI is expressed depending on the context of use thanks to a suite of models analysable, editable, and manipulable by a software.
- *Central storage of models*: each model is stored in a model repository where all UI models are expressed according to the same UI Description Language (UIDL).
- *Transformational approach*: each model stored in the model repository may be subject to one or many transformations supporting various development steps.
- *Multiple development paths*: development steps can be combined together to form developments paths that are compatible with the organisation's constraints, conventions, and context of use.
- *Flexible development approaches*: development approaches (e.g., top-down, bottom-up, wide spreading, and middle-out) are supported by flexibly following alternate development paths and enable designers to freely shift between these paths depending on the changes imposed by the context of use.

2 The Reference Framework used for Multi-Directional UI Development

Multi-directional UI development is based on the Cameleon Reference Framework which defines UI development steps for multi-context interactive applications. Its simplified version, reproduced in Fig. 1, structures development processes for two contexts of use into four development steps (each development step being able to manipulate any specific artefact of interest as a model or a UI representation):

1. *Final UI* (FUI): is the operational UI i.e. 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).
2. *Concrete UI* (CUI): concretises an abstract UI for a given context of use into Concrete Interaction Objects (CIOs) so as to define widgets layout and interface navigation. abstracts a FUI into a UI definition that is independent of any computing platform. Although a CUI makes explicit the final Look & Feel of a FUI, it is still a mock-up 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.
3. *Abstract UI* (AUI): defines interaction spaces (or presentation units) by grouping subtasks according to various criteria (e.g., task model structural patterns, cognitive load analysis, semantic relationships identification), a navigation scheme between the interaction spaces and selects Abstract Interaction Objects (AIOs) for each concept so that they are independent of any context of use. An AUI abstracts a CUI into a UI definition that is independent of any modality of interaction.
4. *Task & Concepts* (T&C): describe the various tasks to be carried out and the domain-oriented concepts as they are required by these tasks to be performed. These objects are considered as instances of classes representing the concepts manipulated.

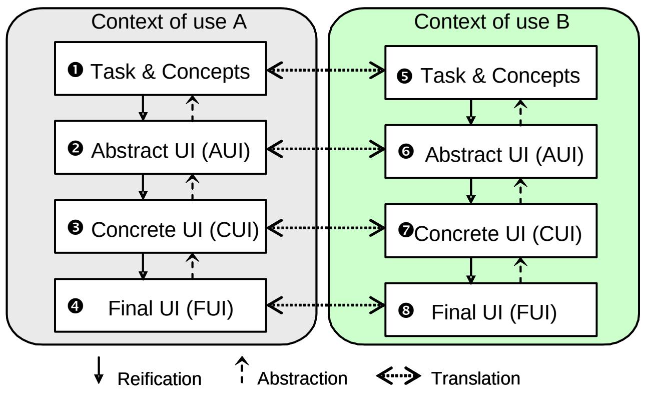


Figure 1: The Cameleon Reference Framework.

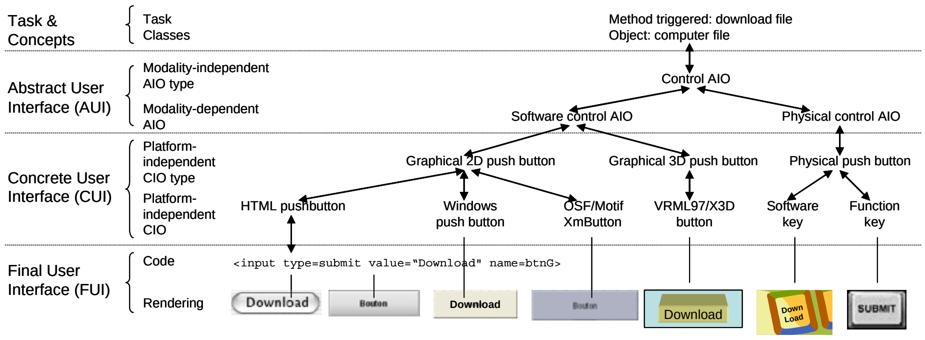
This framework exhibits three types of *basic transformation types*: (1,2) *Abstraction* (respectively, *Reification*) is a process of elicitation of artefacts that are more abstract (respectively, concrete) than the artefacts that serve as input to this process. Abstraction is the opposite of reification. (3) *Translation* is a process that elicits artefacts intended for a particular context of use from artefacts of a similar development step but aimed at a different context of use. With respect to this framework, *multi-directional UI development* refers to a UI engineering method and tool that enables a designer to (1) start a development activity from any entry point of the reference framework (Fig. 1), (2) get substantial support in the performance of all basic transformation types and their combinations of Fig. 1. To enable such a development, the two most important requirements gathered from observations are:

1. A language that enables the expression and the manipulation (e.g., creation, modification, deletion) of the model at each development steps and for each context of use. For this purpose, USIXML is introduced and defined.
2. A mechanism to express design knowledge that would provide a substantial support to the designer in the realisation of transformations. For this purpose, a GT techniques is introduced and defined based on USIXML.

3 Transforming Specifications with USIXML

Graph transformation techniques were chosen to formalise USIXML, the language designed to support multi-directional UI development, because it is (1) **Visual**: every element within a GT based language has a graphical syntax; (2) **Formal**: GT is based on a sound mathematical formalism (algebraic definition of graphs and category theory) and enables verifying formal properties on represented artefacts; (3) **Seamless**: it allows representing manipulated artefacts and rules within a single formalism. Furthermore, the formalism applies equally to all levels of abstraction of USIXML (Fig. 2). USIXML model collection is structured according to the four basic levels of abstractions defined in the Cameleon Reference Framework that is intended to express the UI development life cycle for context-sensitive interactive applications. Each level of Fig. 1 can be itself further decomposed into two sub-levels (Fig. 2):

- At the FUI level, the rendering materialises how a particular UI coded in one language (markup, programming or declarative) is rendered depending on the UI toolkit, the window manager and the presentation manager. For example, a push button programmed in HTML at the code sub-level can be rendered differently, here on MacOS X and Java Swing. Therefore, the code sub-level is materialised onto the rendering sub-level.
- Since the CUI level is assumed to abstract the FUI independently of any computing platform, this level can be further decomposed into two sub-levels: platform-independent CIO and CIO type. For example, a HTML push-button belongs to the type “Graphical 2D push button”. Other members of this category include a Windows push button and XmButton, the OSF/Motif counterpart.



```
<input type=submit value="Download" name=btnG>
```

Figure 2: Example of transformations in USIXML.

- Since the AUI level is assumed to abstract the CUI independently of any modality of interaction, this level can be further decomposed into two sub-levels: modality-independent AIO and AIO type. For example, a software control (whether in 2D or in 3D) and a physical control (e.g., a physical button on a control panel or a function key) both belong to the category of control AIO.
- At the T&C level, a task of a certain type (here, download a file) is specified that naturally leads to AIO for controlling the downloading.

Thanks to the four abstraction levels, it is possible to establish mappings between instances and objects found at the different levels and to develop transformations that find abstractions or reifications or combinations. For example, if a Graphical User Interface (GUI) needs to be virtualised, a series of abstractions is applied until the sub-level “Software control AIO” sub-level is reached. Then, a series of reification can be applied to come back to the FUI level to find out another object satisfying the same constraints, but in 3D. If the GUI needs to be transformed for a UI for augmented reality for instance, the next sub-level can be reached with an additional abstraction and so forth. The combinations of the transformations allow establishing development paths. Here, some first examples are given of multi-directional UI development. To face multi-directional development of UIs in general, USIXML is equipped with a collection of basic UI models (i.e., domain model, task model, AUI model, CUI model, context model and mapping model) (Fig. 4) and a so-called *transformation model* (Fig. 3) [13].

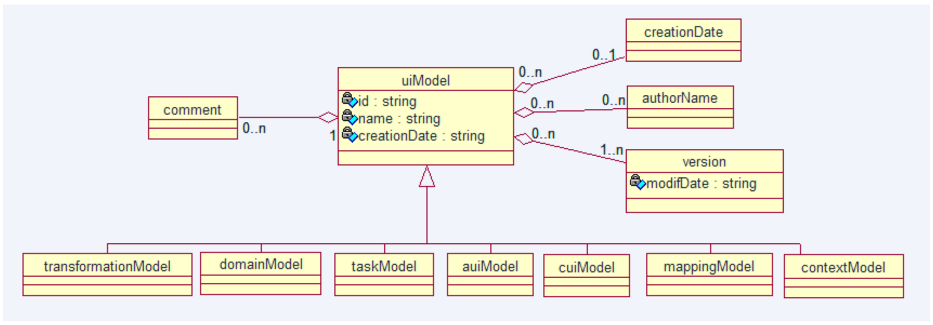


Figure 3: USIXML Model Collection

Beyond the AUI and CUI models that reflect the AUI and CUI levels, the other UI models are defined as follows:

- taskModel: is a model describing the interactive task as viewed by the end user interacting with the system. A task model represents a decomposition of tasks into sub-tasks linked with task relationships. Therefore, the decomposition relationship is the privileged relationship to express this hierarchy, while temporal relationships express the temporal constraints between sub-tasks of a same parent task.
- domainModel: is a description of the classes of objectsmanipulated by a user while interacting with a system.
- mappingModel: is a model containing a series of related mappings between models or elements of models. A mapping model serves to gather a set of inter-model relationships that are semantically related.
- contextModel: is a model describing the three aspects of a context of use in which a end user is carrying out an interactive task with a specific computing platform in a given surrounding environment. Consequently, a context model consists of a user model, a platform model, and an environment model.
- uiModel: is the topmost superclass containing common features shared by all component models of a UI. A uiModel may consist of a list of component model sin any order and any number, such as task model, a domain model, an abstract UI model, a concrete UI model, mapping model, and context model. A UI model does not need to include one of each model component. Moreover, there may be more than one of a particular kind of model component.

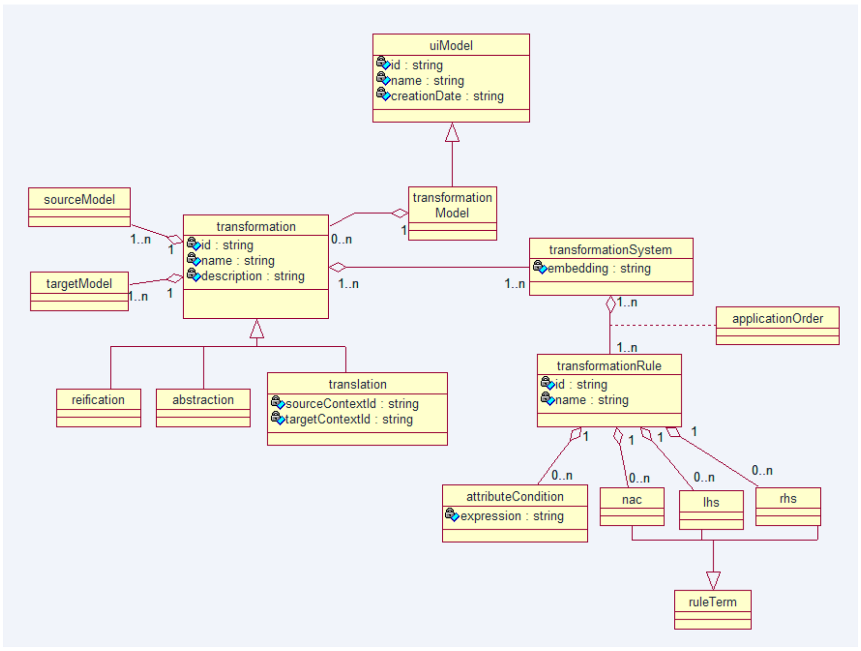


Figure 4: Transformation model as defined in USIXML

4 Tool Support

A tool support is provided at various levels of Fig. 2.

- **Reverse engineering of UI code**: a specific tool, called *ReversiXML*, automatically reverse engineers the presentation model of an existing HTML Web page at both the CUI and AUI levels, with or without intra-model, inter-model mappings. This tool allows developers to recuperate an existing UI so as to incorporate it again in the development process. In this case, a re-engineering can be obtained by combining two abstractions, one translation, and two reifications. This is particularly useful for evolution of legacy systems.
- **Model edition**: as editing a new UI in USIXML directly can be considered as a tedious task, a specific editor called *GrafiXML* has been developed to face the development of USIXML models. USIXML being at first hand a textual language, an *ad hoc* XML editor was created. In this editor, the designer can draw in direct manipulation any graphical UI by directly placing CIOs where they need to be and editing their properties in the Composer, which are instantly reflected in the UI design. At any time, the designer may want to see the corresponding USIXML specifications and edit it. Selecting a USIXML tag automatically displays possible values for this tag in a contextual menu. When the tag or the elements is modified, this change is propagated in the graphical representation. In this way, a bidirectional mapping is maintained between a UI and its USIXML specifications: each time a part is modified, the other one is updated accordingly.

- **Transformation specification and application:** an environment called *TransformiXML*, based on AGG (Attributed Graph Grammars tool) is used for this experiment. AGG can be considered as a genuine programming environment based on graph transformations. It provides 1) a programming language enabling the specification of graph grammars 2) a customizable interpreter enabling graph transformations. AGG was chosen because it allows the graphical expression of directed, typed and attributed graphs (for expressing specifications and rules).
- **A tool for transformation application:** several Application Programming Interface are available to perform model-to-model transformations. We tested AGG API as this API proposes to transform model with graph transformations. An initial model along with a set of rules are transmitted to a Application Programming Interface that performs appropriate model transformations and provide a resulting model that can be edited.

Acknowledgements

USIXML has been developed in the context of the Cameleon research project (http://giove.cnuce.cnr.it/ cameleon.html) under the umbrella of the European Fifth Framework Programme (FP5-2000-IST2) and is now progressively being adopted in 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).

References

1. Ali, M.F., Pérez-Quiñones M.A., Abrams M.: Building Multi-Platform User Interfaces with UIML. In: Seffah, A., Javahery, H. (eds.): Multiple User Interfaces: Engineering and Application Framework. John Wiley and Sons, New York (2003)
2. Bouillon, L., Vanderdonckt, J., Chow, K.C.: Flexible Re-engineering of Web Sites. In: Proc. of 8th ACM Int. Conf. on Intelligent User Interfaces IUI'2004 (Funchal, January 13-16, 2004). ACM Press, New York (2004) 132–139
3. Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., Vanderdonckt, J.: A Unifying Reference Framework for Multi-Target User Interfaces. Interacting with Computers 15,3 (2003) 289–308
4. Eisenstein, J., Vanderdonckt, J., Puerta, A.: Model-Based User-Interface Development Techniques for Mobile Computing. In: Proc. of 5th ACM Int. Conf. on Intelligent User Interfaces IUI'2001 (Santa Fe, January 14-17, 2001). ACM Press, New York (2001) 69–76
5. Mori, G., Paternò, F., Santoro, C.: Tool Support for Designing Nomadic Applications. In: Proc. of 7th ACM Int. Conf. on Intelligent User Interfaces IUI'2003 (Miami, January 12-15, 2003). ACM Press, New York (2003)141–148
6. Vanderdonckt, J., Berquin, P.: Towards a Very Large Model-Based Approach for User Interface Development. In: Paton, N.W., Griffiths, T. (eds.): Proc. of 1st IEEE Int. Workshop on User Interfaces to Data Intensive Systems UIDIS'99 (Edinburgh, September 5-6, 1999). IEEE Computer Society Press, Los Alamitos (1999) 76–85
7. Wong, C., Chu, H.H., Katagiri, M.A., Single-Authoring Technique for Building Device-Independent Presentations. In: Proc. of W3C Workshop on Device Independent Authoring Techniques (St. Leon-Rot, 15-26 September 2002), accessible at http://www.w3.org/2002/07/DIAT/posn/docomo.pdf