GrafiXML, A Multi-Target User Interface Builder based on UsiXML

Benjamin Michotte, Jean Vanderdonckt
Université catholique de Louvain, Louvain School of Management
Place des Doyens, 1 – B-1348 Louvain-la-Neuve (Belgium)
michotte@isys.ucl.ac.be, jean.vanderdonckt@uclouvain.be

Abstract

We have developed GrafiXML, an original user interface builder in that it enables designers and developers to design several UIs simultaneously for multiple contexts of use, i.e. for many users, platforms, and environments. For this purpose, it maintains coordination between three representations: an internal representation consisting of specifications in USer Interface eXtensible Markup Language (UsiXML), an external representation consisting of the interface preview, and a conceptual representation consisting of a user interface model. GrafiXML is an intelligent UI builder in that it maintains model consistency between these representations through a set of mappings based on a user interface ontology. Thanks to this mechanism, GrafiXML provides a unique set of features for supporting designing interfaces for multiple targets. These features are defined, motivated, discussed, and exemplified on a simple interface. Then, it is explained how the UI resulting from this design can support one or many levels of independence with respect to the underlying context of use.

1. Introduction

Many powerful tools now exist for developing Graphical User Interfaces (GUIs) [4,16,19,25]. These toolsets typically include a builder tool which is a visual editor for developing the GUI graphically for each corresponding operating system or environment, such as Aqua [1] for MacOS, Xf for Tcl/Tk, V4All for Eclipse [2], or UIM/X for OSF/Motif. A developer designs a GUI using a palette of interface interaction objects. When the appearance of the GUI is satisfactory, the developer directs the tool to generate code for the newly constructed User Interface (UI). Some GUI builders go a step further and allow the developer to associate algorithmic code with user interface components. These tools are classified as User Interface Management Systems (UIMSs) [19]. Such tools could drastically speed up the GUI development process because much of the code can be generated automatically [17], which is important since the GUI may occupy a significant portion of the total code [19]. However, an inherent limitation of these tools is that they only provide a subset of the options available in a GUI toolkit. Sometimes the abstractions provided are not sufficient to develop complex parts of the GUI, which means that the developer must then modify the generated code to fine-tune the interface. Migration from another user interface implementation is done manually, requiring a developer to make decisions about mappings and translations from the GUI in the source platform to the GUI in the target platform. Therefore mappings between GUI components may not be consistent across multiple contexts of use such as different computing platforms. The lack of support for multiple contexts of use (and not only multiple computing platforms) process makes it tedious, error-prone, and time-consuming. There are some exceptions that confirm the rule: Galaxy [9] and Simple UI Toolkit [21] embed such abstractions so that each UI developed for a particular computing platform (say for instance MacOS) is automatically translated into exactly the same UI for the other computing platform (say for instance, Linux). But the UI remains basically the same in terms of both components and layout and does not take advantage of the new platform.

There is no genuine support for building UIs in a coordinated way for multiple contexts of use where the context of use is defined as a triple (*U, P, E*) where *U* represents any user stereotype, *P*, any computing platform, and *E*, the physical environment in which the user is carrying out her task with the designated platform. There are some tools that model the UI and that generate UI code from these models [19], but no UI builder for multiple contexts of use. A *target* is defined by a particular UI tailored for a given context of use. Therefore, we believe that today, we do not have any multi-target UI builder.

In the next section, the related work will be structured around three UI representations. Then, GrafiXML, a unique multi-target UI builder will be presented based on identified requirements. All its unique facilities will be exemplified on a running example. The paper will end with a summary of those features and some avenues for future research.

0-7695-3093-1/08 $25.00 © 2008 IEEE
DOI 10.1109/ICAS.2008.29

IEEE computer society

2. Related work

Three main starting points in building UIs are typically found in UI tools (Fig. 1) [3,19]:

1. The *internal representation* is the programmers view, consists in the description of the implementation aspects of the application.
2. The *external view* consists in a view of the interface appearance and basic behavior.
3. The *conceptual view* provides an insight on the logical structure underlying a user interface. A conceptual view provides the designer with a set of abstract concepts facilitating reasoning on the artifact that is being built (e.g., a finite state machine, a class diagram or rule-based systems [13]).

These three views define three possible points to initiate the process of UI development life cycle. By defining transitions between these representations, nine theoretical approaches for UI building exist (Fig. 1).

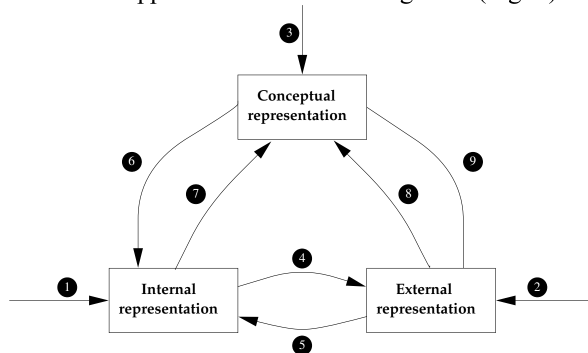

Figure 1. A classification of UI development practices.

In a *programmatic approach* (transition 1), the internal representation is obtained by a direct UI coding. Theoretically, a UI may be coded with any programming, markup or scripting language. Practically, some languages do a better job than others in proposing to designers sets of pre-defined components especially tailored for UI construction. Several transitions may be defined from an internal representation:

- An *internal-external generation* approach (transition 4) derives an external representation from an internal representation (e.g., a web page coded in HTML is rendered in a browser). For example, a XUL UI [20] can be rendered by an engine.
- An *internal-conceptual derivation* approach (transition 7) derives from the internal representation a conceptual representation (e.g., reverse engineering HTML code in order to obtain an abstract view).

In an *exploratory approach* (transition 2), an external representation is firstly provided (i.e., with a visual editor or a mock-up tool) that then initiates:

- An *external-internal representation* approach (transition 5) derives an internal representation from an external representation (e.g., code generation from visually built forms in Visual Basic [17]). Most UI builders, such as MOG UI builder [7], GUIB [10], GTK+ [11], Tilcon UI [23], TrollTech [24], Unidraw [28] fall in this category. SUIT [21] also belongs to this category with the advantage that any UI built for a particular platform is rendered equally in others. The Galaxy Visual Resource Builder [9] is a powerful visual tool for constructing a multi-platform GUI. The resources created by the Visual Resource Builder are stored in a totally portable binary format. Springs and Struts describe the size and positioning constraints of GUI widgets so that all geometry management is done automatically at runtime. Graphical styles [12] are specified independently of the GUI to render them at runtime.
- An *external conceptual derivation* (transition 8) derives a conceptual representation from an external representation. For instance, CanonSketch enables the designer to sketch a UI first and to generate an underlying model behind [6].

In a *specification-based approach* (transition 3), one starts with an abstract UI representation (e.g., a description, a model, or UI specifications) to pursue with:

- A *conceptual-external generation* approach (transition 6) derives an external representation from the conceptual representation. For instance, XXL [14] enables the designer to build a hierarchical UI model that is straightforwardly represented. FormsVBT [3] is unique in that it combines a conceptual UI representation expressed in TeX language and an external representation to produce code. Any change in one view is automatically propagated in the other.
- A *conceptual-internal generation* approach (transition 9) derives an internal representation from the conceptual representation. For instance, The Cameleon reference framework [5] reports on various approaches that start from a task and domain model and terminate with a final UI matching them. In [8], an ontology of the concepts are used to progressively derive a corresponding UI. One single model could be used for this purpose or multiple [26].

The next section will demonstrate that GrafiXML supports all the UI development practices depicted in Fig. 1. Indeed, it relies on all these transitions since it simultaneously combines all three views (i.e., internal = UsiXML specifications, external = graphical representation, and conceptual = concrete UI model) and that each change applied on one representation is straightforwardly propagated in the others so as to maintain one-to-one mapping between the representations. In this way, the designer is free to apply her own preferred UI development practice.

3. A Multi-target UI builder

This section will progressively introduce, motivate, detail all original features of GrafiXML and exemplify them on a simple running example to facilitate the understanding.

3.1 Running example

Let us consider a simple GUI consisting of a login and a password for connecting to a remote system independently of the platform used by the end user. If the combination login+password does not match those recorded in a database, an error message is produced depending on the results. Fig. 2 reproduces GrafiXML's composer where this simple UI is drawn by dragging widgets from the palette and dropping them onto the working area. For this part, GrafiXML is similar to any other UI builder except that more properties can be specified for each widget since it is modeled through a concrete UI model. The left pane of Fig. 2 depicts the list of current projects.

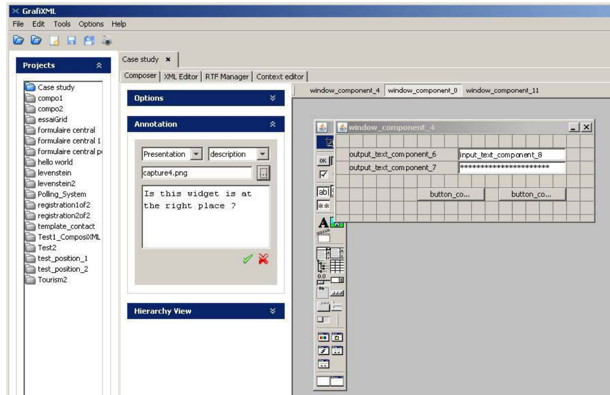

Figure 2. The composer of GrafiXML.

3.2 Platform-independent UI design

It is possible to design a GUI independently of any computing platform and any look & feel by drawing it in the composer. Since GrafiXML is implemented in Java, it will adopt the look and feel of the computing platform on which it is running, but this does not mean that the GUI being designed is targeted towards this platform. The GUI being designed is stored in UsiXML 1.8.0 [27]. Export plug-in's automatically generate code corresponding to various computing platforms such as Java, XUL, or XHTML. If there is a need to preview a GUI for a specific platform, the preview can be obtained according to various schemes such as MS Windows, OSF Motif, and Java (Fig. 3). Rendering engines exist in two forms: code generators (which could be internal plug-in or external transformation engines) and UsiXML interpreters (which opens a UsiXML file and renders it in the environment). For the moment, interpreters exist for Java, Adobe Flash, and Tcl/Tk.

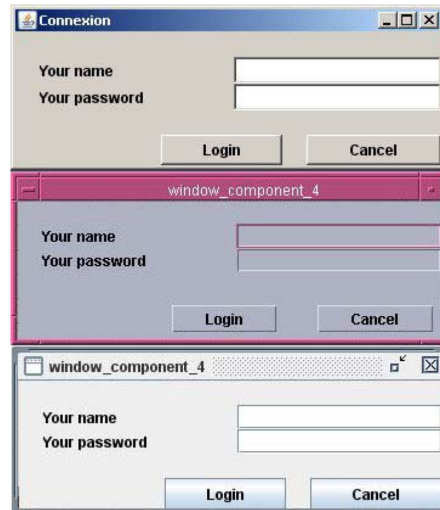

Figure 3. Three renderings of the same GUI.

3.3 Specification of a context of use (target)

Once a GUI is designed in the composer, a particular context of use (or target [5]) can be attached to express that this GUI is relevant for that target. A target is composed of at most three aspects (which do not necessarily be specified): a *user* (which is characterized by attributes such as task experience, disabilities, motivation, experience with interaction devices, preferences), a *platform* (which is specified according to a subset of CC/PP recommendation from W3C), and an *environment* (which is specified by a set of attributes such as level of noise, location, neighborhood, stress level). The context editor (Fig. 4) used to specify these three models is invoked from the composer and automatically generates UsiXML specifications corresponding to the target specified. It is important to maintain specifications in the same User Interface Description Language (UIDL) so as to export the file in one shot and to allow easy transformations. All inputs are achieved by direct manipulation of the concepts involved in the models. A property sheet is then available for those aspects which cannot be specified graphically.

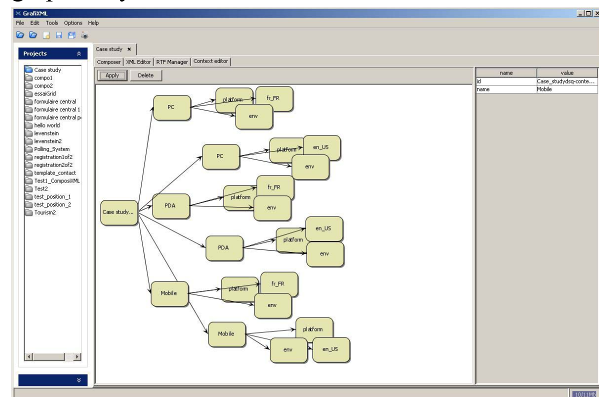

Figure 4. The context editor.

3.4 Multi-target UI design

If a GUI should run in many different contexts of use, then alternative GUI designs should be specified and added for each new context of use, thus leading to specifying a multi-target UI. In this case, each time a target changes, the corresponding UI changes. If the same GUI should be expressed to work on different contexts of use (e.g., the same GUI on different platforms), it is not required to reproduce the same GUI: only the alternative contexts (more precisely here, the alternative platforms) should be specified. Contexts of use can be arranged according to an inheritance hierarchy. In this way, a sub-platform could be specified as a child of an existing platform model. Multiple sub-models of a single model could be specified equally to support multi-model and multi-level UI modeling [26].

Let us imagine in our example that once the login and password are entered, the end user is being asked the platform on which she would like to continue and that the UI will change accordingly. Fig. 4 shows that a GUI is specified for three contexts of use: a stationary context with a desktop PC, a transient context with a PDA, and a mobile context with a mobile phone. In our example, if we do not want to use different UI specifications for PDA and mobile phone because all the widgets we use are available on those platforms, we can factor out the common parts. We only specify that our application is designed to run on those platforms. The rendering engine will adapt the UI for PDA and mobile phone accordingly. Fig. 5 reproduces the new situation in the composer with the corresponding UI variations. The "Options" frame contains the properties of each UI object, whether it is composed or individual. Fig. 6 shows the UI exported in XUL [20] thanks to the "Export to XUL" plug-in [27] and rendered in the XUL-compatible Mozilla browser. This UI can also be run on a mobile phone (Fig. 7) with a XHTML browser.

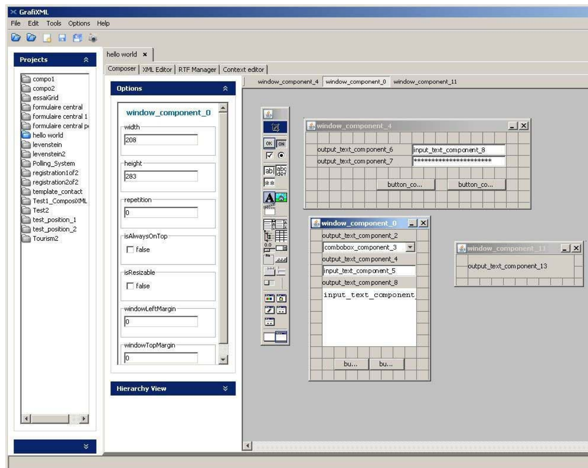

Figure 5. A multi-target UI.

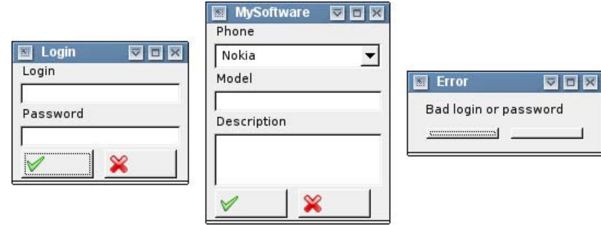

Figure 6. UI rendered in XUL.

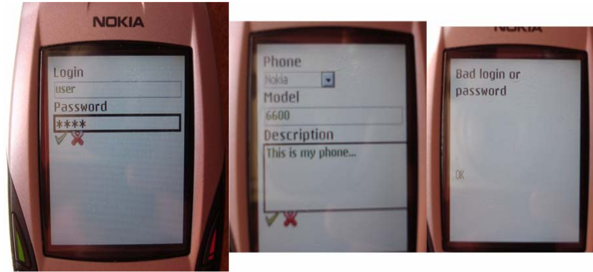

Figure 7. UI rendered on a mobile phone.

Thanks to the "Export to Java" plug-in [27], once your .java file compiled, you can run it on a any Java-compatible platform, such as a MacOsX (Fig. 8). We can use InterpiXML, a UsiXML V.1.8.0 interpreter implemented in Java 1.5, whose rendering on a Linux platform is reproduced in Fig. 9. Last but not least, thanks to the "Export to XHTML" plug-in [27], our application can run on a browser for disabled users, such as a text-based browser (Fig. 10).

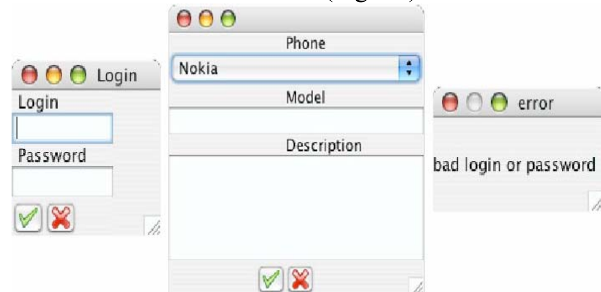

Figure 8. UI rendered on a mobile phone.

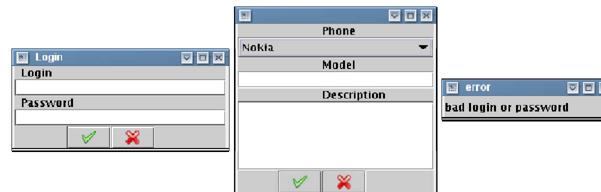

Figure 9. UI rendered by InterpiXML.

3.5 Multi-target UI Localization

Localizing a UI often means a UI specialization (or generalization) for a particular culture, set of users, or population. In this way, all parameters related to user stereotypes are captured in the user model, one of the three dimensions of a target. Therefore, it is possible to support multiple localization of UI, such as for different natural languages at any time. When a new language is added in GrafiXML, it creates automatically a

new context for this language copying the data of the previous context so that you can modify this context yourself as shown previously. In our running example, six contexts will be finally incorporated (Fig. 4): PDA in French, PDA in English, Mobile in French, Mobile in English, PC in French, and PC in English. GrafiXML adds a contextual menu on all widgets to allow the designer to translate their contents for any language. All components have a lot of content you can localize such as text content, icon, shortcut, and tooltip. Fig. 11 shows two definitions of the Cancel push button, one in French and one in English.

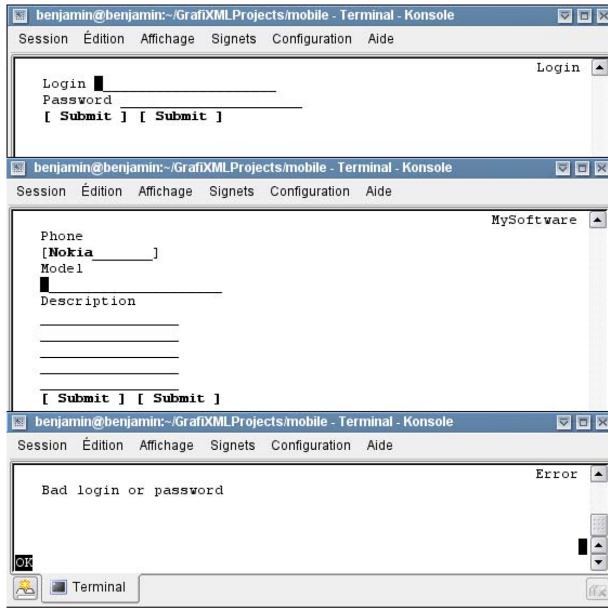

Figure 10. UI rendered in a text-only browser.

Figure 11. Localization of the Cancel push button.

3.6 Three-representation UI Design

Depending on the designer's preference, the UI can be designed in any of the three representations (Fig. 1):

1. The *internal representation* consists of UI specifications expressed in UsiXML V1.8.0 (www.usixml. org), a XML-compliant UIDL for multi-target, multi-model [26], and multimodal UIs. On the upper left of the XML Editor (Fig. 12), we have a tree-based view of the UsiXML specifications. A single click on a node of this tree will select the corresponding lines in the editor. When a node is selected, the bottom left pane shows all the attributes available for this node. You can modify them by selecting the value in a drop down list if values are static, checking a combo box if the value is a Boolean or edit them if it is composed text. You can also edit the UsiXML specification directly in the XML editor if you really want to do so. Expert designers who are familiar with the language may refine the specifications directly in this window.
2. The *external representation* consists of a view provided in the composer. It is a synthetic and simplified view of every widget (as in Fig. 5). 32 widgets are today supported to cover a wide range of platforms. A preview facility allows the designer to see the resulting UI for a particular Look & Feel for a particular platform that has been specified, thus reaching the level where a true external representation is brought to the designer's eye.
3. The *conceptual representation* consists of a Concrete User Interface (CUI) model, made of objects that are independent of any context of use. At any time, the corresponding Abstract User Interface (AUI) model is also generated to augment translation capabilities, as in the Cameleon framework [5].

Any editing applied to any particular representation is immediately propagated in the other two representations, as in FormsVBT [3]. For this purpose, an ontology of the CUI and AUI are exploited to maintain a set of mappings between the representations [26].

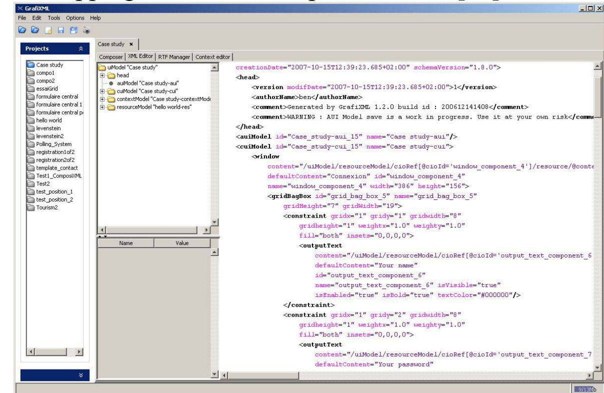

Figure 12. UsiXML editor for the UI being designed.

3.7 Annotation-based UI design

Not all information related to the UI objects can be captured in any existing UI builder that fits all the purposes. This is also applicable to GrafiXML: although a conceptual representation is maintained for both a CUI and a AUI, possibly along with a context model, it cannot capture all design aspects through the underlying model. Therefore, there is a need to provide some support for annotation-based design. An annotation is defined as any information captured at UI design-time that needs to be further exploited in the remainder of the UI development life cycle. It could be a guideline for a model-to-code generator, a model-to-model transformation engine, or simply for human purposes. Several types of annotations are defined: *Presentation* (any guideline related to presenting information such as a metric, a convention), *Specification* (any guideline related to the connection with the data base, such as the data type), *Verification* (any syntactical or semantic constraint to be verified, such as a mask, a profile, or a regular Perl expression), *Discussion* (any design consideration that requires further attention and refinement) and *Tools* (any guideline that will be exploited later on by other software for automatic processing). All these annotation types have options such as task, domain for Specification, description for Presentation, etc. For instance, SketchiXML (http://www.usixml.org) is a multi-fidelity software for sketching a UI which can export a UI into a UsiXML file. This file can then be in turn imported in GrafiXML and refined. Or in the other way around. When multiple designers collaborate in the design case, an annotation can be refined with a sub-type such as “decision”, “proposition” or “argumentation” to capture at design-time multiple or alternative UI design considerations and facilitate the decision. An annotation can be augmented by text, image (e.g. a drawing), sound or voice (e.g., a vocal comment). Annotations are saved in the UsiXML de-scription, such as (Fig. 2, central frame):

<annotation annotation-Type="Discussion#proposition" file="capture4.png"> Is this widget is at the right place? </annotation>

3.8 Visual UI (de)composition

A GrafiXML plug-in, called ComposiXML, has been developed in order to compose and decompose existing GUIs. In UI builders, UI recomposition is traditionally performed by copying and pasting UI controls of interest from one UI to another one, thus requiring many manual adjustments such as alignment, resizing, reshuffling. These operations, although simple, are often perceived as tedious [25]. To overcome these shortcomings, the Operator allows the designer to select one or two GrafiXML projects, that is one or two UsiXML files, and make some composition or decomposition operations on these UI, which are as follows:

- *Unary Operators*: these operators are used to operate on a single UI at a time. They are used to filter, remove widgets or change a kind of widget by another.
- *Binary Operators*: these operators are used to compose a single UI from different UIs. You can choose to remove duplicated items or select only those items. In our running example, we decided to merge the three windows of Fig. 5 into a single one. ComposiXML provides the fusion binary operator (Fig. 13) for obtaining the final UI reproduced in Fig. 14.

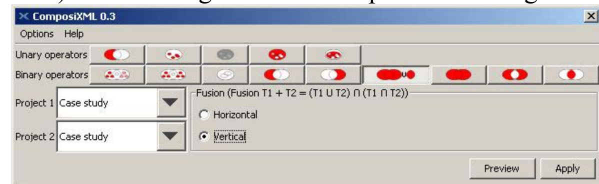

Figure 13. Interface of composition plug-in.

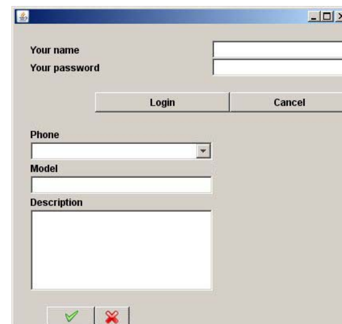

Figure 14. Fusion of three windows into a single one.

3.9 Graceful degradation of UI

An important requirement identified for multi-target UI [5,16] is the ability to easily transform a UI existing for a source context into a new one that is tailored to a target context. For example, a GUI designed for a desktop PC may not fit in the constraints imposed by a smaller platform, such as an Internet ScreenPhone or a PDA. Therefore, instead of starting designing a new UI from scratch, it is desirable to apply a series of transformations to the initial GUI to adapt it to the final context. The “Graceful degradation” plug-in has been developed for changing a GUI to fit it for another platform in a logical way for an entire UI, and not just a window of it. For instance, we can develop a PC UI and transform it into a PDA one. The plug-in offers five families of transformation rules which can be specified and triggered at once or separately (Fig. 15): *resizing rules*, *moving rules*, *interactor transformations* (e.g., a radio button is reduced to a combo box as in Fig. 16), *image transformations*, and *splitting rules*. The primary advantage of this approach is that transformations are applied logically on the conceptual representation, thus updating the other views accordingly.

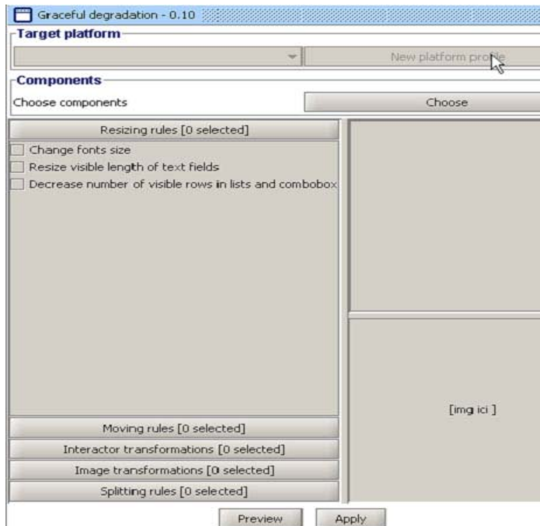

Figure 15. UI of the “Graceful degradation” plug-in.

Figure 16. Example of a widget substitution applied.

3.10 Usability evaluation of UI at design-time

Since a UI conceptual representation is continuously maintained throughout the development life cycle, it is an appropriate candidate to apply model-checking techniques to verify properties of interest, such as usability guidelines, physical properties, heuristics, rules of thumbs or any other type of evaluation principle [4]. For this purpose, UsabilityAdvisor (www.usixml. org) is a GrafiXML plug-in that performs logical evaluation of properties expressed in a XML-language on the corresponding UsiXML specifications of the GUI of interest. The evaluation could be performed on a *single UI* at a time (e.g., checking the alignment of controls) or *multiple UIs* simultaneously (for example, one can check the consistency between windows across multiple targets by asking the plug-in to compare the model definitions across the multiple targets). This is particularly appreciated when several versions of the same UI should be maintained in a coordinated way, such as in the multi-target situation. If the designer wishes to check another guideline, she may enter the guideline in the XML language and see its evaluation incorporated without changing the evaluation engine.

3.11 Multiple levels of independence

Thanks to the different models involved in GrafiXML, it is possible to specify a GUI dependently or independently of various concerns on demand. Five levels of independence are depicted in Fig. 17:

- *Device independence*: the CUI level allows expressing a UI without any reference to any term belonging to a particular input/output device. In particular, there is no physical coordinates for any widget constituting the UI, either absolute or relative. In this way, the description of the UI is independent of any screen resolution, any window manager or toolkit.
- *Platform independence*: when a CUI does not preclude any reference to a particular computing platform, it is said to be platform independent, which is the case for a CUI. However, if a link between a CUI and a particular platform needs to be established, a mapping between this CUI and a platform model could be maintained as long as this is relevant.
- *‘Modality of interaction’ independence*: the AUI level allows expressing a UI without any reference to any term belonging to a particular modality of interaction (e.g., graphical interaction as in GUIs, sonic interaction in auditory interfaces, speech synthesis/recognition in speech interfaces, haptic for touch-sensitive interfaces). Since this level is independent of any modality of interaction, only a recursive decomposition of actions is produced. A same platform may combine one or many interaction modalities.
- *Channel independence*: a channel is defined as a particular computing platform, along with a selected set of modalities of interaction in a given physical environment. For instance, using an interactive kiosk with an Internet navigator that is HTML-enabled and connected to a T1-network consists of a particular channel of interaction. Another channel could be for instance the production of structured PDF documents from the same con-tents to be delivered through Internet web sites. When a AUI does not preclude any reference to any channel, it is said to be channel independent. When a need arises to create a mapping between a AUI and its relevance for a particular channel, a mapping between this AUI and a corresponding platform can be established and maintained.
- *‘Context of use’ independence*: ultimately, any AUI is said to be context-independent when there is no reference to any term relevant to a context of use [5]. When such a reference exists deliberately, the AUI is mapped to a context model stating that this AUI is relevant to this context of use, although the AUI does not contain any descriptor.

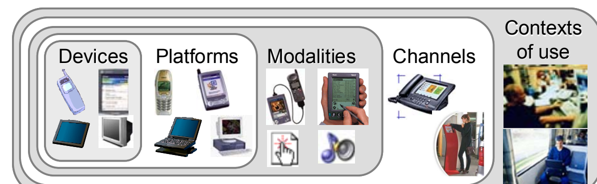

Figure 17. Multiple levels of independence supported.

5. Conclusion

In this paper, we have introduced, motivated and discussed GRAFIXML, a software that supports multi-target design of UIs thanks to a set of several facilities that are often cited as requirements for selecting an appropriate UI builder [4,16,25]: platform independent design, multi-target, localization, context editing, usability evaluation are among them. In the near future, we will deploy a system for fostering plug-in deployment over the Web in order to allow any interested party to make a new plug-in largely available.

10. References

[1] Apple Corp., Aqua Interface Builder, 2007, http://developer.apple.com/tools/interfacebuilder/

[2] Assisi, R., "V4ALL GUI Designer for Eclipse Manual V.1.0", Visual Builder for Eclipse, Eclipse, 2004, http://v4all.sourceforge.net/index_start.html

[3] Avrahami, G., Brooks, K.P., and Brown, M.H., "A Two-view Approach to Constructing User Interfaces", Proc. of SIGGRAPH'89, pp. 137–146.

[4] Bass, L., Abowd, G., and Kazman, R., "Issues in the Evaluation of User Interface Tools", in Proc. of Workshop on Software Engineering & Computer-Human Interaction, Lecture Notes in Computer Science, Vol. 896, Springer, Berlin, 1995, pp. 17-27.

[5] Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., and Vanderdonckt, J., "A Unifying Reference Framework for Multi-Target User Interfaces, *Interacting with Computers*, 15(3), 2003, pp. 289–308.

[6] Campos, P. and Nunes, N.J., "Towards useful and usable interaction design tools: CanonSketch", *Interacting with Computers*, 19(5-6), 2007, pp. 597–613.

[7] Colebourne, A., Sawyer, P., and Sommerville, I., "MOG User Interface Builder: A Mechanism for Integrating Application and User Interface", *Interacting with Computers*, 5(3), 1993, pp. 315–331.

[8] Furtado, E., Furtado, J.J.V., Silva, W.B., Rodrigues, D.W.T., Taddeo, L.S., Limbourg, Q., Vanderdonckt, J., "An Ontology-Based Method for Universal Design of User Interfaces", Proc. of Workshop on Multiple User Interfaces over the Internet: Engineering and Applications Trends MUI'2001, Lille, 10 September 2001, http://www.cs.concordia.ca/~faculty/seffah/ihm2001/papers/furtado.pdf

[9] Galaxy Application Environment, Ambiência, http://www.ambiencia.com/galaxy/galaxy.htm

[10] Graphical User Interface Builder (GUIB), 2004, http://www-k3.ijs.si/kokalj/guib/

[11] GTK+ User Interface Builder, http://glade.gnome.org/

[12] Hashimoto, O. and Myers, B.A., "Graphical Styles for Building User Interfaces by Demonstration By Example", Proc. of the ACM Symposium on User Interface Soft-ware and Technology UIST'92, ACM Press, New York, 1992, pp. 117–124.

[13] Hudson, S.E. and Yeatts, A.K., "Smoothly Integrating Rule-Based Techniques into a Direct Manipulation Interface Builder Input Techniques", Proc. of the ACM Symposium on User Interface Software and Technology UIST'91 (Hilton Head, 11-13 November 1991), ACM Press, New York, 1991, pp. 145–153.

[14] Lecolinet, E., "XXL: A Dual Approach for Building User Interfaces", Proc. of the ACM Symposium on User Interface Software and Technology UIST'96, ACM Press, New York, 1996, pp. 99–108.

[15] Lumsden, J. and Gray, P.D., "SUIT - Context Sensitive Evaluation of User Interface Development Tools", Proc. of DSV-IS'2000, Lecture Notes in Computer Science, Vol. 1946. Springer, Berlin, 2000, pp. 79–95.

[16] McKirdy, J., "Choosing the UI Tool Which Best Suits Your Needs", Proc. of 7th IFIP Int. Conf. on Human-Computer Interaction Interact'99 (Edinburgh, 30-August-3 Sept. 1999), IOS Press, Amsterdam, 1999.

[17] Milosavljević, B., Vidaković, M., Komazec, S., and Milosavljević, G., "User interface code generation for EJB-based data models using intermediate form representations", Proc. of the 2nd Int. Conf. on Principles and practice of programming in Java (Kilkenny City, June 16-18, 2003), ACM Press, NY, 2002, pp. 59–64.

[18] Myers, B.A. and Rosson, M.B., "Survey on User Interface Programming Tools and Techniques", Proc. of ACM Conf. on Human Factors in Computing Systems CHI'92 (Monterey, 3-7 May 1992), ACM Press, New York, 1992, pp. 195–202.

[19] Myers, B.A., "User Interface software Tools", http://www-2.cs.cmu.edu/afs/cs.cmu.edu/user/bam/www/toolnames.html

[20] Oeschger, I., XUL Programmer's Reference Manual, 5 April 2001, http://www.mozilla.org/xpfe/xulref/

[21] Pausch, R., Conway, M., and DeLine, R., "Lessons Learned from SUIT, the Simple User Interface Toolkit Practice and Experience", *ACM Transactions on Information Systems*, 10(4), 1992, pp. 320–344.

[22] Puerta, A.R., Cheng, E., and Ou, T., "MOBILE: User-Centered Interface Building Tools for Building Interfaces and Applications", Proc. of ACM Conf. on Human Factors in Computing Systems CHI'99, ACM Press, New York, 1999, pp.426-433.

[23] The Graphical Interface Company, Tilcon Interface Builder, http://www.tilcon.com/ graphicseditor.html

[24] Trolltech Corp., Qt Windows editor, 2004, http://www.trolltech.com/products/qt/windows.html

[25] Valaer, L.A. and Babb, R.G., "Choosing a User Interface Development Tool", *IEEE Software*, 14(4), July-August 1997, pp. 29–39.

[26] Vanderdonckt, J., Furtado, E., Furtado, V., Limbourg, Q., Silva, W., Rodrigues, D., and Taddeo, L., "Multi-model and Multi-level Development of User Interfaces", *Multiple User Interfaces - Cross-Platform Applications and Context-Aware Interfaces*, John Wiley & Sons, New York, 2003, pp. 193–216.

[27] Vanderdonckt, J., "A MDA-Compliant Environment for Developing User Interfaces of Information Systems", Proc. of CAiSE'05, Springer, Berlin, pp. 16–31.

[28] Vlissides, J.M. and Tang, S., "A Unidraw-based User Interface Builder", Proc. of the 4th Annual ACM Symposium on User Interface Software and Technology UIST'91. ACM Press, New York, 1991, pp. 201-210.