Prototyping Multi-platform Software Using the UsiXML4ALL Tool

ABSTRACT

In the development of multi-platform applications, one of the most challenging problems is the prototyping of the user interface (UI), e.g. the support to rapidly build different final look and feel possibilities among the available platforms. This paper presents UsiXML4ALL, a software tool developed to facilitate the creation of multi-platform applications prototypes. UsiXML4LL acts as a renderer, mapping concrete UI's described in UsiXML to multiple platforms, and also as a connector, linking the rendered UI to application logic code developed possibly in multiple programming languages. The goal is to allow a consistent look and feel and full functionality of an application over various different platforms. UsiXML4ALL is intended to support not only the prototyping of new (multi-platform) applications but also the migration of existent applications to a multi-platform environment.

Categories and Subject Descriptors

H.5.2 [**Information Interfaces and Presentation**]: User Interfaces - *Evaluation/methodology, Prototyping.*

General Terms

Design, Human Factors.

Keywords

Multi-platform application, UsiXML, User Interface Rendering, Rendering Tool.

1. INTRODUCTION

Computer software development has nowadays as an important requirement the possibility of execution in more than one platform, either through desktop computer, handhelds or mobile phones.

This situation creates a new challenge to interactive software developers, including the need for application prototyping and interaction testing in many available platforms. This way, user interface developers want to be able to validate their work in a simple way, using supporting tools to rapidly build different final look and feel possibilities among the available platforms. In order to solve this problem, the possibility of multi-platform UI prototypes generation with minimal code alteration, making possible the interaction validation, would be a great benefit to UI designers.

Our approach to address this problem is the utilization of plastic user interfaces, capable of adapting themselves to different use contexts, in the creation of multi-platform UI prototypes. Specifically, we use High-level User Interface Descriptions (HLUID), which enable the definition of UI's in a platform independent form. Among the available HLUID's (see section 2), UsiXML [13] is based on the *Cameleon reference framework* [6], allowing the description of UIs for multiple use contexts.

This paper presents UsiXML4ALL, a software tool developed to facilitate the creation of multi-platform applications prototypes. UsiXML4LL acts as a renderer, mapping concrete UI's described in UsiXML to multiple platforms, and also as a connector, linking the rendered UI to application logic code developed possibly in multiple programming languages. The goal is to allow a consistent look and feel and full functionality of an application over various different platforms. UsiXML4ALL is intended to support not only the prototyping of new (multi-platform) applications but also the migration of existent applications to a multi-platform environment.

The paper is structured as follows: firstly, we discuss the challenges and problems of multi-platform software development, and some approaches to resolve this problem. Then, we describe the main concepts of UsiXML4LL (e.g. its architecture and some implementation details), discussing its features and benefits, and how to use it. An actual multi-platform prototype example illustrates the process of multi-platform UI rendering and multilanguage application logic connection. Some concluding remarks and future work are presented in the final section.

2. MULTI-PLATFORM SOFTWARE DEVELOPMENT

Recent years have seen the evolution of computational technology, which has allowed the development of many devices, providing users with access to processing power in different situations. This context has transformed the possibility of software execution in multiple platforms in an important requirement, proposing a new challenge for developers of interactive applications [19].

Within this challenge software developers face yet another problem: how to create multi-platform application prototypes, for testing purposes, without a great programming effort?

This situation has become a major issue, because applications prototypes are an important step in the development process, specially in a multi-platform environment, where the interaction technique must be also tested, in addition to the application itself. Indeed, the creation of multi-platform prototypes is almost impracticable without any support tool, due to both necessary work and technological knowledge of a developer in order to create applications for each device.

To address this problem, one approach is the usage of multi-platform UI development techniques in the UI prototyping creation process. A straightforward solution is the usage of plastic user interfaces, capable of being executed in multiple use contexts. Specifically, in this paper the term *multi-platform* corresponds to the definition of *context of usage* within the scope of plastic user interfaces, enclosing multiple devices, although some works consider also multiple modalities (e.g., graphical interface, voice interface and so on) and multiple environmental attributes existing when the software is being executed (e.g., light conditions, user profile). The main goal of plasticity is to accomplish these requirements, preserving the usability of the application.

In order to develop plastic user interfaces, different techniques have been proposed. These propositions can be classified according to the World Wide Web (W3C) note on authoring technique for device independence [3], which identifies three classifications for authoring techniques: *single authoring, multiple authoring* and *flexible authoring*. A brief description of each one of these categories is [21]:

- Multiple authoring: The developer creates a specific version of the application for each device or device category. This situation, which includes (re)creation and maintenance for each platform, is extremely costly, and could result in users having many different versions of applications on different devices [7], but also provides the maximum control over the results.
- Single authoring: In this category, only one interface implementation is created, which is adapted to a specific device before being presented to the user. Single authoring techniques can be subdivided in techniques that use **platform independent vocabularies or toolkits**, like AUIML [4] or UIML [1], techniques that **extend established markup languages**, as RIML (developed as part of the Consensus Project [8]) or techniques which use **model-based user interface development**, as XIML [20] or UsiXML [13].
- Flexible Authoring: Situation where the developer combines single authoring and multiple authoring techniques.

In this work, our solution approach is a model-based user interface development single authoring technique, using a High-Level User Interface Description Language to allow the design of multi-platform user interfaces.

A solution using HLUID's was chosen because UI description languages have been widely used in multi-platform UI development, mainly because they abstract the user interface description, providing a uniform way to develop multi-platform and even multimodal user interfaces. Besides that, the characteristic shared by many HLUID's, which is to be a XML-based declarative language, makes these languages easy to be learned and understood, having potential to be adopted by a large developer community.

We adopt a model-based approach because it allows us to work in many different abstraction levels (e.g., task, abstract user interface, concrete user interface) of the same description, having the choice between different approaches in the creation of multi-platform user interfaces.

3. USER INTERFACE EXTENSIBLE MARKUP LANGUAGE - UsiXML

Different High Level User Interface Description Languages have been proposed in order to design multi-platform user interfaces, like TERESA [19], UIML [1], XIML [20] and WSXML [11]. It is not the purpose of this paper to investigate these languages, and a deeper analysis can be found in [13] and [22].

Among the existent HLUID's, we have chosen UsiXML because it is a language specially intended for context sensitive UI's [13], having potential to become a w3c standard and being supported by an active and international research community.

The User Interface eXtensible Markup Language (UsiXML) is a UI description language which pursues the goal of capturing the essential properties of interest that turn out to be vital for specifying, describing, designing and developing such UI's [13].

In order to achieve these goals, UsiXML is based on the four abstraction levels of the *Cameleon reference framework* [6], which allows the description of a context-sensitive UI design cycle, as described by Figure 1.

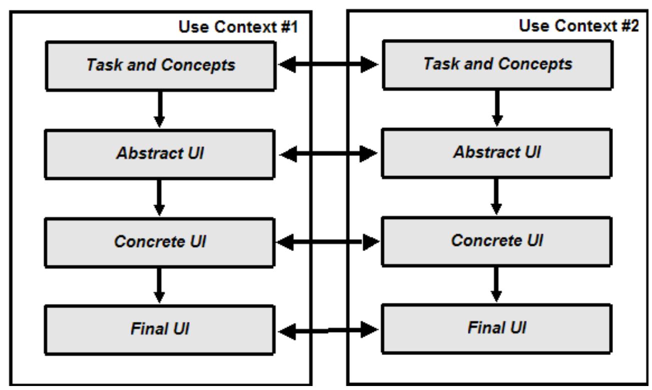

Figure 1. The four basic levels of the Cameleon reference framework [13].

The four basic levels existent in this framework are [13]:

- Final User Interface (FUI): UI running on a particular platform either by interpretation or by execution.
- Concrete User Interface (CUI): abstracts the FUI into a UI definition that is independent of any computing platform.
- Abstract User Interface (AUI): abstracts the CUI into a UI definition that is interaction modality independent (e.g., graphical/vocal interaction).
- Task and Concepts: highest level, where the user task is defined based on his viewpoint, along with the various objects that are manipulated by it.

In the *Cameleon Reference Framework*, development steps are obtained in vertical or horizontal transformations. Vertical transformations (reifications) define processes in that the output is a UI model which is more concrete than the model used as input. In horizontal transformations (translations), the model received as output has the same level of abstraction than the one used as input, but is defined for another use context.

In order to represent the abstraction levels defined on the framework, UsiXML defines the following concepts [13].

The *Task Model* describes the user tasks that can be performed in an interaction with the computational system. These tasks are modeled in UsiXML with an extended version of the *ConcurTaskTree* (CTT) notation [18]. Within this representation, a task model is composed by tasks and relationships, where the task can be described by a name, a type and its frequency, and the relationships can be of two kinds, *decomposition* and *temporal*.

The *Domain Model* defines the real world concepts and its associations, as understood by the users. In UsiXML, the existing concepts of the domain model are described in the same way as a UML class diagram. Among the existing concepts are classes, attributes, methods and relationships between the domain objects.

To allow the specification of content independent user interfaces, UsiXML defines a *Resource Model*, which specifies the UI content, like messages and images, facilitating the UI adaptation to different use contexts.

A *Context Model* describes the entities which may influence the execution of an interactive task with the user interface. The context model is supposed to capture any relevant information of the use context where the application is being executed, being composed by an *user model*, which classifies the existing users into stereotypes, a *platform model*, which captures attributes of the platform/hardware combination, and a *ambient model*, which defines interesting properties of the current environment.

The *Abstract User Interface* (AUI) represents a canonical expression of the user interface, in a way that is independent of any modality or computing platform. An AUI is composed of abstract interaction objects (AIO), which abstract components that are present in most platforms, like windows and buttons for graphic platforms.

The *Concrete User Interface* (CUI) allows the specification of an user interface definition in a modality dependent, platform independent way, making possible the UI rendering to multiple platforms. A CUI is populated by *Concrete Interaction Objects* (CIO), which contain in its definition information about its characteristics. A CUI layout is defined without any absolute position information, but in a hierarchical manner, allowing the specification of position relations between elements. In addition to the components definition, a CUI allows the specification of the UI dynamic behavior, through a navigation and a event/action definition language [13].

In order to clarify the CUI concept, an example is shown in Figure 2, which is the specification of a CUI containing a window with two elements, a text field (*textComponent*) and a button. It can be observed that each component definition contains also its attributes, like *id* and *name* for the 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 2. UsiXML CUI specification [13].

4. RELATED WORK

The accomplishment of multi-platform UIs is also the goal of some related works in the literature, which can be classified in two categories: a) tools working with UsiXML UI descriptions and b) UI rendering tools, for UsiXML or other UI models.

Among the projects which use UsiXML, SketchiXML [9] can generate a UsiXML Concrete UI (CUI), receiving as input hand sketched UI descriptions, having as main goal the creation of evolutionary UI prototypes. Working with another kind of input, GrafiXML [12] is a visual designer which allows the creation of CUI specifications.

In the category of UI rendering tools, QTKiXML [10] can map UsiXML description to the Tcl-Tk language. With the same objective, FlashiXML [5] can also map UsiXML descriptions, but to UI's described in vectorial mode, being interpreted by Flash or SVG plug-ins. InterpiXML [16] performs the mapping of UsiXML CUI descriptions using *Java Swing* UI components. Using another UI languages, Uiml.NET [14] and TIDE [2] map user interfaces specified in UIML [1] to the .Net and Java platform respectively. TERESA (Transformation Environment for InteRactivE System representations) [15] allows the design of multi-platform UI with the utilization of TeresaXML UI descriptions. Also in this category, the MONA project [21] has developed a single authoring tool which can be used to create multimodal user interfaces.

UsiXML4ALL is similar to the works presented above, because it explores the design of multi-platform applications using UsiXML. As a differential, UsiXML4ALL has also as a goal the possibility of the rendered user interface connection with application logics developed in multiple programming languages.

5. UsiXML4ALL

The rendering tools presented in the last section have some limitations. They don't address the problem of the connection between the rendered user interface and the application logic, allowing, in the majority of cases, only the connection to application logic in a specific programming language.

Due to the limitation of these tools to a specific technology, its utilization in real world software prototyping is made difficult, since interactive tests are less effective because of the absence of a connection to the application logic. In this situation, the developer would benefit from a tool that could permit UI connection with application logic developed in multiple programming languages, because this feature would enable the creation of multi-platform prototypes independent from the language being used in the application development.

For example, when an application developed in C# must be expanded to support new platforms, like a mobile platform, the UI developers have to learn how to implement user interfaces in this specific platform in order to create the first interactive prototypes. Another solution would be the utilization of a UI rendering support tool, but then a specific tool that supports C# would have to be found.

UsiXML4ALL is a rendering tool that outcomes these problems, making possible UI rendering in multiple platforms. More, UsiXML4ALL supplies as differential the possibility of application logic connection in multiple programming languages.

The main goals of UsiXML4ALL are to make possible CUI UsiXML descriptions rendering to multiple platforms, to allow logic application connection to multiple programming languages and to have an extensible architecture, being able to be extended to new use contexts.

The proposed architecture is shown in Figure 3. In this representation, dotted lines describe the user interface rendering process, and normal lines the logic application connection.

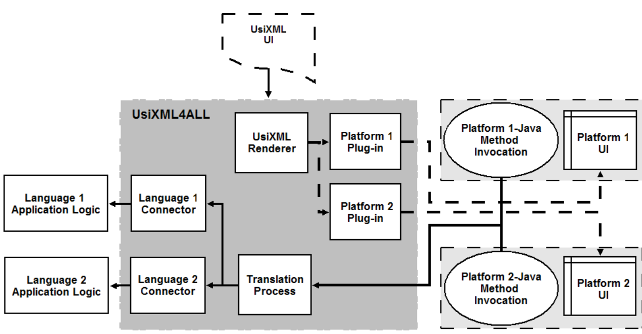

Figure 3. UsiXML4ALL Architecture.

In order to perform the UI rendering, UsiXML4ALL receives as input a CUI UsiXML description (*UsiXML UI* in Figure 3), and forwards it after validation to the target platform rendering plug-in (*Platform 1 Plug-in* in Figure 3). The rendering plug-in is responsible for the UI components instantiation, being the application logic method calls redirected to the UsiXML kernel (*Translation Process* in Figure 3).

To connect the UI to its application logic, the UsiXML4ALL kernel (*Translation Process* in Figure 3) receives methods invocations and translates them to a language independent format. This description is forwarded to a plug-in for the target programming language (*Language 1 Connector* in Figure 3), which calls the method in the application logic being executed.

The rendering plug-ins (*Platform Plug-in 1 and* 2 in Figure 3) have the task of mapping the user interface UsiXML CUI description to a final user interface in a specific platform.

In order to accomplish this task, the rendering algorithm hierarchically covers the CUI description contained in the UsiXML file, instantiating the contained components with its specified characteristics.

During the rendering process, the user interface content is obtained in the UI resource model, and the necessary application logic methods are obtained in the UI domain model.

To connect with the logic application, UsiXML4ALL has the definition of logic connectors (*Language 1 and 2 Connector* in Figure 3). The connectors are responsible for receiving the description of the called method, translate it to the logic application programming language, and invoke it.

In order to perform this task, the connector is subdivided in two different parts, the first one developed in Java, and the second developed in the target programming language. These two parts are connected by calls made through the *Java Native Interface* (JNI), which makes possible the connection between Java and other programming languages applications.

In this process, the method description is passed through JNI, and reflection techniques are then used to realize the method call in the target programming language, based on its name and attributes.

6. CASE STUDY: MULTI-PLATFORM CALCULATOR

To evaluate the first implementation of UsiXML4ALL, an example prototype was developed: a multi-platform calculator. The goal was the UI rendering in three different platforms, *Java Swing* and *Winforms* for a desktop version of the application, and *Java Swing* also for its mobile version, using the J2ME *Connected Device Configuration* (CDC). In addition to that, all UI's should be able to connect to two different application logics, in Java and C#.

To this, a UsiXML CUI description of the calculator UI was created. The application logics in Java and C# were developed, both implementing the same interface, with one single method, called *buttonPressed()*. In the execution of the application, when the user presses one button in the calculator UI, a method call is transmitted to the UsiXML4ALL kernel, which translates it at runtime to the target application logic, returning its result to the UI.

In order to describe the user interface, a UsiXML file containing the calculator CUI description was created. This file contains not only the UI components definition, but also its content and dynamic behavior specification. A sample of the calculator's user interface UsiXML code is presented in Figure 4.

In Figure 4, a part of the calculator CUI description can be seen. In this description there is a window (*window* element), which uses a flowbox layout manager (*flowbox* element), and is composed by a display (*inputText* element) and buttons (*button* element).

```
<cuiModel id="5-cui_20" name="5-cui">
   <window id="window_1" name="window_1"
      width="192" height="200">

      <flowBox id="flow_box_2" name="flow_box_2" alignment="left">

         <inputText id="input_1"
            name="input_1" isVisible="true"
            isEnabled="true" textColor="#000000"
            maxLength="50" numberOfColumns="15" isEditable="true"/>

         <button id="button_1"
            content="/uiModel/resourceModel/
              cioRef[@cioId='button_1']/resource/@content"

            name="button_1" isVisible="true"
            isEnabled="true" textColor="#000000"/>

         <button id="button_2"
            content="/uiModel/resourceModel/
              cioRef[@cioId='button_2']/resource/@content"

            name="button_2" isVisible="true"
            isEnabled="true" textColor="#000000"/>

         <button id="button_3"
            content="/uiModel/resourceModel/
              cioRef[@cioId='button_3']/resource/@content"

            name="button_3" isVisible="true"
            isEnabled="true" textColor="#000000"/>

                  ...
      </flowbox>
   </window>
</cuiModel>
```

Figure 4. Calculator UsiXML description.

In the calculator application, the specified events refer to the operation of clicking in one of the calculator's buttons. In this situation, the method *buttonPressed()* must be invoked in the logic application, receiving as return value the value that must be displayed in the calculator.

However, to maintain programming language independence, the CUI UI description directs its methods invocations to UsiXML4ALL, informing its name and parameters. This definition is translated to the application logic programming language being executed, and the method is invoked. Figure 5 shows an example of the source code necessary to perform a method call in a language independent manner. In Figure 5, the method *buttonPressed* declared in the class *br.inf.ufrgs.calc.logic.CalculatorLogic* is prepared and invoked, using for that the *logicConector* class, which is based on the *Command* GoF design pattern.

```
logicConector.initializeMethod();
logicConector.prepareMethodClass(
        "br.inf.ufrgs.calc.logic.CalculatorLogic");
logicConector.prepareMethodName("buttonPressed");
logicConector.prepareMethodIntegerParameter(1, 0);
logicConector.prepareMethodStringReturnParameter();
logicConector.executeMethod();
```

Figure 5. Method invocation source code.

In this way, a prototype for the three different platforms can be created, and it can be tested with logic application developed in programming languages. Having as example Figure 6, 6 different UI-application logic combinations could be created (A,1; B,1; A,2; B,2; A,3; B,3). In this case, the mobile version of the user interface could be connected to a C# source code only if the device used to display the user interface could execute C# applications.

In practice, to perform this operation, the application logic code has to be developed, providing the interface with the methods to be called by the UI. In the situations presented in the example, this is done by the creation of a *.jar* file for the Java application logic, and a *.dll* library for the C# version.

With the application logic interface defined, the user interface can be created in a UsiXML file, which will declare the UI structure, behavior and the signature of the application methods to be called.

In order to change the user interface platform, as in switching from A1 to A2 in Figure 6, the only operation needed is to change the execution parameters of UsiXML4ALL, specifying the renderer to be used.

In the case of changing from A1 to A3, a different tool has to be used, because the current version of UsiXML4ALL has two implementations, for desktop and for mobile platforms, as show in Figure 6.

To change the logic application being used, as in A1 to B1 in Figure 6, again the only operation needed is a parameter change, specifying the logic connector to be used, as long as the two application logic versions implement the same interface.

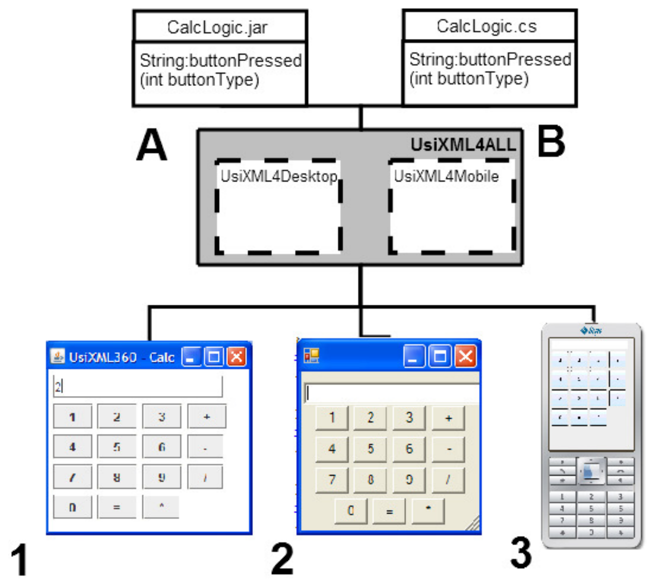

Figure 6. Example application.

7. CONCLUSIONS AND FUTURE WORK

A practical approach to the development of multi-platform applications prototypes was introduced, and the first version of UsiXML4ALL was presented. UsiXML4ALL acts as a UI renderer in multiple platforms, and also allows the UI connection to application logic developed in multiple programming languages. This tool has potential to stimulate the utilization of HLUID's in the prototyping of multi-platform applications, as much in the development of new applications, as in the migration of legacy applications to a multi-platform environment.

Future work consists in the evolution of UsiXML4ALL, allowing the creation of UI's to other (conventional or not) devices and platforms, in addition to multimodal UI's.

Examples of devices/platforms we intend to investigate are mobile phones, smartphones, PDA's, web-based interfaces, desktop interfaces and even XO laptops from the OLPC Project [17].

The final objective is to allow the creation of UsiXML-based user interfaces for a great number of platforms and devices, and also expand the number of supported programming languages. In particular, our work aims to provide a tool which can be used in actual user interface prototyping in such a diversity of contexts of usage.

Another interesting possibility is the investigation of UsiXML4ALL as a support for the prototyping of multimodal user interfaces, enabling not only device-independent presentation but also new interaction modalities like voice or gesture. In this case, the basic difference would be the adaptation of the rendering process to these new interaction modalities.

8. ACKNOWLEDGMENTS

This research is partially funded by CNPq (LIFAPOR/CNPq-Grices Project).

9. REFERENCES

[1] Abrams, M., Phanouriou, C., Batongbacal, A. L., Williams,S. M., and Shuster, J. E. "UIML: An Appliance-IndependentXML User Interface Language." Proceedings of the 8th International WWW Conference. Toronto, Canada. 11-16 May 1999. Elsevier Science Publishers.

[2] Ali, M.F., Pérez-Quiñones, M.A., Abrams, M., e Shell, E. Building Multi-Platform User Interfaces With UIML. In Proceedings of 2002 International Workshop of Computer-Aided Design of User Interfaces: CADUI'2002. Valenciennes, France.

[3] Authoring Techniques for Device Independence. W3C Working Group Note 18 February 2004. http://www.w3.org/TR/2004/NOTE-di-atdi-20040218/

[4] Azevedo, P., Merrick, R., Roberts, D. "OVID to AUIML - User Oriented Interface Modeling." http://math.uma.pt/tupis00/submissions/azevedoroberts/azevedoroberts.html

[5] Berghe, Y. Etude et implémentation d'un générateur d'interfaces vectorielles à partir d'un langage de description d'interfaces utilisateur, M.Sc. thesis, Université catholique de Louvain, Louvain-la-Neuve, Belgium, September 2004.

[6] 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.

[7] Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Souchon, N., Bouillon, L., Florins, M., Vanderdonckt, J. Plasticity of User Interfaces: A Revised Reference Framework. In Pribeanu, C., Vanderdonckt, J. (Eds.), Proceedings of 1st International Workshop on Task Models and Diagrams for User Interface Design TAMODIA'2002 (Bucharest, July, 18-19, 2002), Academy of Economic Studies of Bucharest, INFOREC Printing House, Bucharest, pp. 127–134, 2002.

[8] Consensus Project. http://www.consensus-online.org/

[9] Coyette, A., Faulkner, S., Kolp, M., Limbourg, Q. SketchiXML: Towards a Multi-Agent Design Tool for Sketching User Interfaces Based on UsiXML. In Proc. of Tamodia'2004.

[10] Denis, V. Un pas vers le poste de travail unique: QTKiXML, un interpréteur d'interface utilisateur à partir de sa description, M.Sc. thesis, Université catholique de Louvain, Louvain-la-Neuve, Belgium, September 2005.

[11] 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.

[12] Lepreux, S., Vanderdonckt, J., Michotte, B. Visual Design of User Interfaces by (De)composition, Em Proc. of 13th Int. Workshop on Design, Specification, and Verification of Interactive Systems DSV-IS'2006 (Dublin, 26-28 de Julho de 2006), G. Doherty and A. Blandford (eds.), Lecture Notes in Computer Science, Vol. 4323, Springer-Verlag, Berlin, 2006, pp. 157-170.

[13] Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., Florins, M. and Trevisan, D. UsiXML: A User Interface Description Language for Context-Sensitive User Interfaces. In Proc. of the AVI'2004 Workshop "Developing User Interfaces with XML: Advances on User Interface Description Languages" UIXML'04 (Gallipoli, May 25th 2004). EDM-Luc, 55–62.

[14] Luyten, K., Thys, K., Vermeulen, J., e Coninx, K. A Generic Approach for Multi-Device User Interface Rendering with UIML. In 6th International Conference on Computer-Aided Design of User Interfaces (CADUI'2006), Bucareste, Romênia.

[15] Mori, G., Paternò, F., Santoro, C. Tool Support for Designing nomadic Applications. Em Proc. of 7th ACM Int.Conf. on Intelligent User Interfaces. ACM Press, New York, 2003, pp. 141-148.

[16] Ocal, K. Etude et développement d'un interpréteur UsiXML en Java Swing, Haute Ecole Rennequin, Liège, 2004.

[17] One Laptop Per Child (OLPC). http://www.laptop.org/index.en_US.html

[18] Paternò, F. Model-Based Design and Evaluation of Interactive Applications, Springer-Verlag, Berlin, 2000.

[19] Paternò, F., Santoro C. One model, many interfaces. In Proceedings of the Fourth International Conference on Computer-Aided Design of User Interfaces, pp 143-154. Kluwer Academics Publishers, 2002.

[20] Puerta, A. and Eisenstein, J. "XIML: A Common Representation for Interaction Data." Proceedings of IUI 2002, International Conference on Intelligent User Interfaces. San Francisco, California, USA. ACM Press.

[21] Simon, R., Wegscheider, F., Tolar, K. Tool-supported single authoring for device independence and multimodality. Proceedings of the 7th international conference on Human computer interaction with mobile devices & services

MobileHCI '05. Salzburg, Austria. Pages: 91 - 98 ISBN:1-59593-089-2

[22] 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 Sys-tems DSV-IS'2003, Lecture Notes in Computer Science, Vol. 2844, Springer-Verlag, Berlin, 2003, pp. 377-391.