A Model-Based Approach for Developing Vectorial User Interfaces

Jean Vanderdonckt, Josefina Guerrero-Garcia, and Juan Manuel González-Calleros
Université catholique de Louvain, Louvain School of Management, Information Systems Unit
Place des Doyens, 1 – B-1348 Louvain-la-Neuve (Belgium)
{jean.vanderdonckt, josefina.guerrero, juan.m.gonzalez}@uclouvain.be

***Abstract*—This paper presents a model-based approach for developing vectorial user interfaces to an interactive applications, whether it is a web or a stand-alone applications. A vectorial user interface exhibits the capability of being rescaled in any dimension without any loss of information, while taking advantage of the screen real estate offered by the computing platform on which the interactive application is running. A model describes the vectorial user interface in order to capture its presentation and behavior in a way that is independent of any context of use. Implemented as a browser plug-in, a rendering engine parses this model at run-time so as to render the user interface bounded with the domain, thus producing together a running application. This facilitates platform adaptation, since the interface scales up or down depending on the screen resolution and user adaptation since the model can change from one session to another. The interface is then re-rendered with adaptation for the benefit of the end user. Both platform and user adaptations contribute to making the web application accessible in a ubiquitous way.**

***Keywords:* Context of use, Liquid design, Scalability, Ubiquitous computing, User interfaces, User Interface Description Language, Vectorial interface, Web application.**

I. Introduction

Developing a User Interface (UI) of an interactive application poses multiple challenges to the designer and the developer, especially those of development complexity, diversity of existing Integrated Development Environments (IDEs), and the significant amount of programming skills and usability knowledge required to produce a usable UI [2][6][13]: markup languages (e.g., HTML), programming languages (e.g., C, C++, or Java), communication skills, usability engineering, among others, are all important. These difficulties are multiplied when a UI should be developed for multiple contexts of use [4] such as multiple categories of users (e.g., having different preferences, speaking different native languages), different computing platforms (e.g., a mobile phone, a Pocket PC, a kiosk, a laptop, a wall screen), and various working environments (e.g., stationary, mobile [20]). We therefore define a context of use as a triple: $C = (U, P, E)$ where U denotes a user model, P, a platform model, and E, an environment model respectively [4].

We hereby refer to a *Vectorial User Interface* as any UI that exhibits the capability of being rescaled along any dimension (horizontally and/or vertically) without any loss of information, while taking advantage of the characteristics offered by the context of use. A vectorial user interface is equivalent to a vector-based UI since it is graphically rendered using vector graphics (such as vectorial shapes in drawing applications) as opposed to raster graphics (such as bitmaps in painting applications). The potential advantages of a vectorial UI for a web user are [7][11]:

- *Platform independence*: the same UI can be rendered indifferently on any platform since its definition does not refer to any platform-specific peculiarity such as resolution, absolute positioning of widgets, etc. Consequently, the resulting UI is never subject to pixelation, a process where interpolation is applied in order to determine a rendering between two raster graphics.
- *User interface scalability*: the same UI can be rescaled along any dimension, which is particularly appropriate for horizontally- or vertically-oriented platforms or platforms equipped with an accelerometer (e.g., the Apple iPhone) that rotates the UI depending on its orientation.

Developing a vectorial UI, e.g. for a web application, could imply several challenges today:

- *For the developer*: unless a toolkit is used for this purpose (e.g. Adobe Flash), it is very difficult to develop vectorial UI since every widget should be individually drawn in terms of lines, shapes. In addition, the software development life cycle of such a vectorial UI remains mostly an ad-hoc implementation. Instead, it should go for a step-wise development method, as recommended in [6]. It is also hard to achieve active support for dynamic content [15].
- *For the designer*: the design step is often forgotten for such UIs since the trend is to “rush to the code” before designing anything. It is rather difficult for a designer to design a UI for multiple contexts of use while avoiding to reproduce multiple UIs for multiple contexts of use. Some designers do pay attention to “liquid design”, a reduced form of a graphical UI where the UI is accommodated depending on the screen resolution, usually relying on Cascading Style Sheets.
- *For the end user*: the rendering may be slow, resource-demanding, usability is not guaranteed [17], badly or incorrectly produced, or illegible (e.g., the UI is running on a small screen, but impossible to use because of its tiny size).

In order to address these issues, this paper presents a model-based approach for developing a vectorial user interface of a web application. Thanks to this approach, a model of the vectorial UI is produced (in UsiXML format) that is rendered at run-time thanks to a browser plug-in, called FlashiXML, thus providing a proof-of-concept.

The reminder of this paper is structured as follows: in the 'state of the art' Section, existing works are discussed comparing advantages and disadvantages of current solutions. Section 3 outlines the model-based approach (models, language, approach) and details the conceptual elements used to model a vectorial UI. Section 4 introduces, describes, and motivates FlashiXML, a browser plug-in that renders a vectorial UI from a UI model specified in UsiXML. Section 5 exemplifies this approach by demonstrating some case study. Section 6 concludes the paper by presenting some avenues of this research.

II. State of The Art

There have been several attempts to disseminate Integrated Development Environments (IDEs) that support vectorial UIs where the UI is mostly developed by hand. **SVG** is a W3C standard language for describing vectorial graphics that is itself a delivery target. Most of existing browsers provide native support for SVG [20]. The variations in the scripting interfaces in current implementations remains a major challenge. Therefore, SVG has been used as a target language for vectorial graphics [20].

The ubiquitous Adobe **Flash** environment, given that it is now on nearly all platforms, remained a scripting language. Several programming languages have been offered on top of Flash in order to provide developers with a higher order in programming Flash applications.

The open source compiler and programming language **Haxe** [12] is proving to be a very effective solution for developing Flash applications. **OpenLazlo** (www.openlaszlo.org) [18] and Macromedia Flex (www. macromedia.com/go/flex) are the two development environments for vectorial UIs. Both approaches have common characteristics: the authoring language used was created by them, they used a XML format, the target language is Macromedia Flash, dynamic aspects are defined using scripts, both were server oriented and evolved to be client oriented building *Rich Internet Applications* (RIAs). But the common drawback is that every UI is developed by hand.

The Flex server offers a declarative programming methodology for delivering rich Internet applications. Relying on a *n*-tier architecture a layer with executable code can be added to the client machine. Operations and code is stored in the client machine those reducing server time response for simple operations. Flex is compatible with other standards, including: HTML, HTTP(S), XML, SOAP/web services, CSS, SVG, J2EE, and the .NET platform. A big advantage of using Flex over other IDEs is that new application modules can be easily integrated to the *n*-tier architecture. The compatibility to the resulting code, flash, created by the same society, Macromedia. The script langue is also compatible with the one used in Flash applications. Even though, some drawback exist on Flex runs just in Java application servers, even the simplest example need a server configuration.

OpenLaszlo applications are made available on the web either: on a server is stored locally in the client computer or a compiler. The OpenLaszlo architecture is designed to support multiple device types. Its dynamic layout mechanisms enable simple modifications to such properties as an application's overall size to be applied by the platform. This simplifies adapting an application to work on screens and devices of different size.

An interesting aspect of both approaches is the language used as *User Interface Description Language* (UIDL), later used for the language selection. On the one hand, The Macromedia Flex Markup Language (MXML) [1] is a declarative XML-based language used to describe UI layout and behaviors, and ActionScript for the Flex Framework. MXML is used to create client logic for rich extensible UI components for creating RIAs, as well as interactive applications. On the other hand, OpenLaszlo applications [10] uses a XML based specification plus a java script file. By comparing these two approaches some conclusions arise: Flex offers a bigger set of graphical components, OpenLaszlo server is less performance than java servers, both are solutions depending on their own, and both need server configuration.

PlastiXML [5] consists of an environment that enable designers to produce UIs for multiple platform by exemplifying variations from one platform to another, but the resulting UI is not vectorial. Instead, different GUIs are produced that are switched depending on the platform. In [7], examples of vectorial UIs are delivered and a method to define them precisely. As opposed to this approach, we propose a model-based approach that captures these aspects. Pleuß [22,23] also proposes a model-driven approach for multimedia UIs that supports Flex as output.

III. Model-Based Approach for Vectorial User Interfaces

Although many different approaches have been defined and used, the community has reached to a relatively global consensus [14] about the components of a User Interface (UI) development method. The variations are mainly in the way and level of details, but not very much in the goals:

- *Models*. A series of models pertaining to various facets of the UI such as: domain, presentation, dialog.
- *Language*. In order to specify different aspects and related models, a specification language is needed that allows designers and developers to exchange, communicate, and share fragments of specifications and that enables tools to operate on these specifications. These models are uniformly and univocally expressed according to a single Specification Language.
- *Approach*. An approach refers to the research line or paradigm to be followed by the method.
- *Tools*. A suite of software engineering tools that supports the designer and the developer during the development life cycle according to the method.

The following subsections respectively address each item by proposing a solution to this tem, along with its justification.

A. Approach

In software engineering, specification-based (or model-driven) approach relies in the power of models to construct and reason about software systems. This approach is based on models. The goal of model-based approach, for user interface development is to propose a set of abstractions, development processes and tools enabling a engineering approach of user interface development. The characteristics of an engineering approach are its systematic (development based of rational principles), its reproducibility, its orientation towards quality criteria.

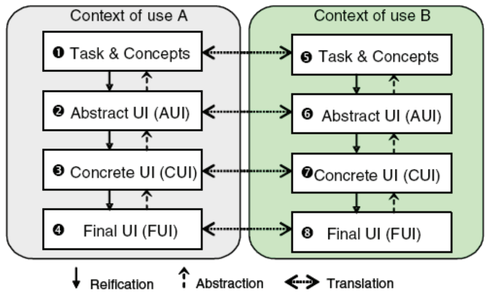


Figure 1. The Simplified Cameleon Reference Framework [4].

Our methodology (Figure 1) structures development processes into four development steps:

1) *Task & Concepts (T&C):* describe the various user's tasks to be carried out and the domain-oriented concepts as they are required by these tasks to be performed.
2) *Abstract UI (AUI):* defines abstract containers (AC) and individual components (AIC) two forms of Abstract Interaction Objects (AIO) [2] by grouping subtasks according to various criteria (e.g., task model structural patterns, cognitive load analysis, semantic relationships identification), a navigation scheme between the container and selects abstract individual component for each concept so that they are independent of any modality.
3) *Concrete UI (CUI):* concretizes an abstract UI for a given context of use into Concrete Interaction Objects (CIOs) [4] so as to define widgets layout and interface navigation. It abstracts a FUI into a UI definition that is independent of any computing platform.
4) *Final UI (FUI):* is the operational 3DUI i.e. any 3DUI 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.

Model-Driven Development (MDD) approach has been selected for its characteristics, including but not limited to: Modifiability of the models, Complexity support, Rigorousness, some reasoning is possible to be understandable for humans.

B. Models

The present work just affects and extends two levels of the Framework introduced in previous section, which are: The CUI and FUI layers. Nevertheless, it benefits from the whole framework. The CUI model is assumed to be described without any reference to any particular computing platform or toolkit of that platform, thus contributing to the platform independence characteristic needed for our solution. For this purpose, a CUI model consists of a hierarchical decomposition of CIOs. A *Concrete Interaction Object* (CIO) is defined as any UI entity that users can perceive such as text, image, animation and/or manipulate such as a push button, a list box, or a check box. A CIO is characterized by various attributes such as, but not limited to: *id*, *name*, *icon*, *content*, *defaultContent*, *defaultValue*, *isVisible, isEnabled, fgColor* and *bgColor.* Each *CIO* is then sub-typed into one of the two categories: *graphicalContainer* for all widgets containing other widgets such as page, window, frame, dialog box, table, box and their related decomposition or *graphicalIndividualComponent* for all other traditional widgets that are typically found in such containers. A graphicalIndividualComponent cannot be further decomposed. The model supports a series of widgets defined as graphicalIndividualComponents such as: *textComponent*, *videoComponent*, *imageComponent*, *imageZone*, *radioButton*, *toggleButton*, *icon*, *checkbox*, *item*, *comboBox*, *button*, *tree*, *menu*, *menuItem*, *drawingCanvas*, *colorPicker*, *hourPicker*, *datePicker*, *filePicker*, *progressionBar*, *slider*, and *cursor*.

C. Language

A User interface description language (UIDL) is needed to support the MDD method. In the state of the art section, MXML and OpenLaszlo were identified to support vectorial UIs development. However, they are not generic to be compliant with the requirements of a MDD. In [26] a number of XML-compliant languages for defining user interfaces, including vectorial UIs, served for selecting the UIDL. From this review it was found some limitations of existing UIDLs, including: the set of widgets cannot be expanded; an interpreter is needed in any case for supporting other languages; context dependent; technology dependent; and platform dependent. To these shortcomings a more general one is added: whenever we would like to submit an extension of an existing language there is no guarantee that the Consortium in charge with that language will consider it. After identifying the shortcomings on existing UIDL languages, the selection went to UsiXML (USer Interface eXtensible Markup Language) [13]. The selection of UsiXML for that purpose includes, among other, the following advantages: Coverage of elements and models for Model-Driven UI development; Quality of the semantic definition of the language; and support for tools. The advantages include:

- *Multilingual UI* [19], offering the user to select the language before launching the application.
- *Event handling* triggering behavior defined internally (windows transitions) and externally (scripts)

- The possibility to *execute programs locally* (client oriented) or *remotely* (server oriented) [16], for instance by having a UsiXML interpreter either locally or remotely.
- The *portability* due to the use of a standard target language. This includes multiplatform (Flax plug-ins are available for almost all operating systems) and multi-device (Flash plug-in exist for mobile devices as well as desktop computers).
- The selected UIDL that is part in a broader Framework allowing UsiXML specifications to be rendered in other UIDLs, such as UIML (www.uiml.org) or XIML (www.ximl.org) provided that the concepts manipulated have a counterpart in those languages. More about UsiXML can be found at: www.usixml.org

There are still some disadvantages in this proposed solution, which can be listed: the non-correspondence between the source and target language which means that any change in the target language need an adaptation of the system ; once rendered, there is no way to adapt the UI but just relying on the supported adaptation to screen size provided by Flash.

IV. A RENDERING ENGINE FOR VECTOR-BASED UIS

A rendering engine for vector-based UIs is presented in this section that is compliant with the model-based approach discussed in the previous section. Any UsiXML-compliant user interface definition can be opened and rendered in this interpreter so as to create the truly working interface with presentation and dialog. In this running environment, the UI can be resized at any time to address some constraints imposed by the computing platforms and to support some properties of *Graceful Degradation* of UIs [7], a sub-property of the *Plasticity property* [4]. In this way, any UsiXML-compliant UI can be rendered on any computing platform equipped with a SVG or Flash plug-in or player. For this purpose, an underlying mini-toolkit has been developed in ActionScript in the Macromedia Flash environment so as to render basic widgets which were not available natively in the Macromedia Flash environment. We have decided to implement a rendering engine in the FlashMX environment [1] that would render a vectorial UI by interpreting its XML description remotely, such as in [16][24]. Flash is a widely used tool for creating multimedia elements. Flash can generate interactive animations that deliver considerable visual impact with relatively small sized files. Flash content is browser independent and looks the same on all graphical browsers that are equipped with the necessary Flash plug-in or reader.

Before the simultaneous release of the Flash MX authoring tool and the Flash Player 6, Flash generated content was inaccessible to many disabled Web users. It was not possible to add alternative text equivalents to the visual content for users of screen readers or caption audio content for users with impaired hearing.

Although Flash presented accessibility barriers for many people with physical disabilities, in some cases the use of Flash enhanced accessibility for people with cognitive and learning disabilities. A concept or process is sometimes considerably easier to understand when it is presented in a simple, elegant animation rather than explained in words.

The dynamic generation of vectorial UIs works as follows: a configuration file is used, containing a series of parameters describing the application set up, including:

System parameters. They are used to locate the resources needed to generate the UI. When the parameter standalone is set to false, the server will be in changer to generate the file dynamically, i.e. those xml files found in the local folder and/or in the subfolders (UsiXML and config in Figure 2). Otherwise, when the parameter is set to true, the file must be edited manually for each aggregated or suppressed element from the list.

Language parameter allows the user to choose the language for the UI. Two parameters are included: language selection to indicate whether language selection is a service provided to the final user or not; and language navigation to indicate user privileges to change or not the language at run time.

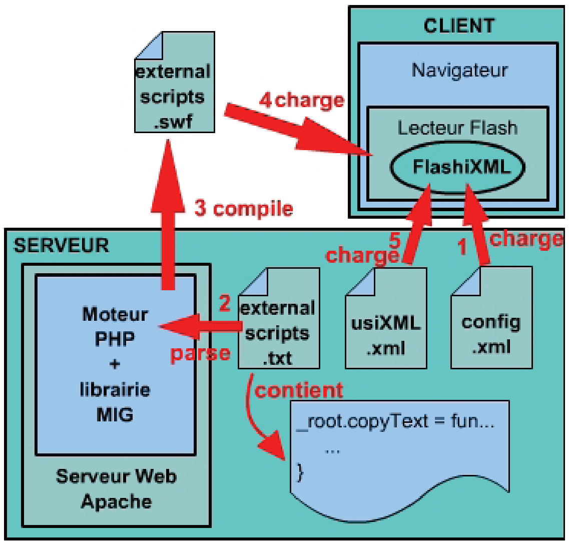


Figure 2. Dynamic generation of Vectorial User Interfaces

Navigation Manager. Depending on the application type and its objectives sometimes is relevant to have applications with some functionality associated for navigation. The renderer uses a set of parameters to address ergonomic aspects of the application (such as allowing or not full screen presentation). Other parameters serve just to provide feedback about the system (error messages when rendering), this is more for debugging purposes.

Parameters of the graphical elements of the User Interface. Some rules and parameters are needed in order to guarantee the way widgets are rendered, while some widgets might be specified in terms of number of pixels, for instance, an image component, some other use a different value, for instance, *number of characters* is used in a text component. Some heuristics are needed in order to decide how to display the widgets, for instance, using the average size of images to display all of them with the same size, or the average number of characters to display harmoniously a set of text components.

TABLE I. SYSTEM PARAMETERS

Parameter	Type	Description
standalone	Boolean	Indicates if the application run in serve mode (false) or standalone mode (true)
usiXML files location	String	File path including the UsiXML to be loaded
external scripts file location	String	File path containing the scripts that can be associated to objects

TABLE II. PARAMETERS FOR NAVIGATION

Parameter	Type	Description
displayWindowBar	Boolean	Indicate if the icons bar is displayed or not on minimized or closed windows
displayMenu	Boolean	Indicate if the menu option is displayed or not
displayUsiXMLInformation	Boolean	Indicate if the information contained in the UsiXML is show or not in the menu
displayFlashiXMLInformation	Boolean	Indicate if FlashiXML about information is displayed or not
displayErrorLog	Boolean	Indicate if a menu item link to show errors caused on generation are added or not to the menu
allowFileReloading	Boolean	Indicate if the UsiXML files are added or not to the menu
displayFullScreen	Boolean	Indicate if the application is displayed full screen or not
allowFullScreen	Boolean	Indicate if the application can switch from full screen to normal or not

A. Generation of Graphical Elements

So far, widget models were discussed and some basic rules for their presentation. To assure a correct mapping between the model specification (stored as a UsiXML file) and the target language, in this case flash, an interpreter was built. The first step is to build flash widgets corresponding to UsiXML widgets. The list of Flash components that are supported includes: button, combo box, check box, image component, radio button, text component, window, and box. The properties of flash elements have an impact on the final rendering. For instance the combo box, Figure 3, is rendered differently when the attribute *isDropDown* is set to true (combo box) or false (list box).

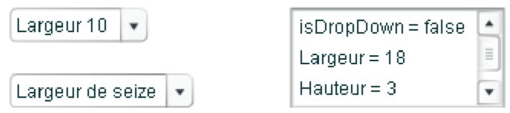


Figure 3. Illustration of the graphical elements: Combo Box

The text component (Figure 4) is a particular widget. It corresponds to the different variation of graphical elements that handles text (label, textfield, textarea, passwordfield). Depending on the attributes selected in the UsiXML model the corresponding rendering is generated by the interpreter. on (Figure 6).

The box (Figure 5) serves as a container for the different widgets, this objects is not part of any other vectorial UI language. It relies, but is not limited, to those used in Java for layout and containers. Similarly the window concept is also not present. Next section explains how these elements gather the different graphical elements. The attribute border width is illustrated in Figure 5, notice how the size for the content is reduced.

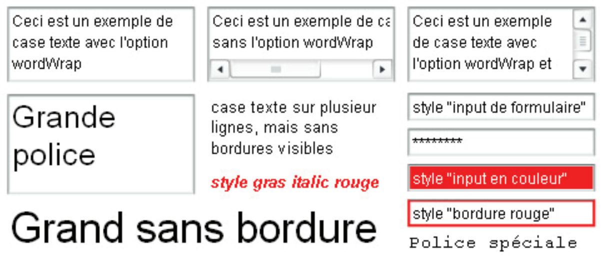

Ceci est un exemple de case texte avec l'option wordWrap
Ceci est un exemple de ca sans l'option wordWrap
Ceci est un exemple de case texte avec l'option wordWrap et
Grande police
case texte sur plusieur lignes, mais sans bordures visibles
style gras italic rouge
style "input de formulaire"
****
style "input en couleur"
style "bordure rouge"
Grand sans bordure
Police spéciale

Figure 4. Variations of the text component rendering

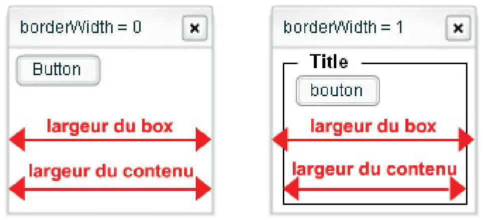


Figure 5. A window with the box element

B. Positioning Graphical Elements

So far, objects instantiation has been discussed. This includes the size of the containers (window and box) the type of layout desired (for instance, average of their pixel size). The next step is to create the layout of the UI, one of the major challenges that were faced in this work. An example is used to illustrate this step. Considering the UsiXML code, shown below, representing two windows with a same structure, a box with a text component and a button, the sequence diagram is shown in Figure 7.

```
<window id="W0">
    <box id="B0">
        <textComponent id="T0"/>
        <button id="BT0"/>
    </box>
</window>
<window id="W1">
    <box id="B1">
        <textComponent id="T1"/>
        <button id="BT1"/>
    </box>
</window>
```

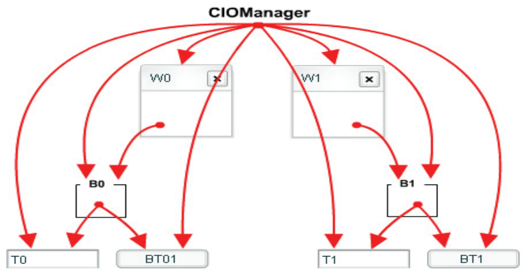


Figure 6. References to the widgets.

Once a UsiXML is loaded (1), all the windows elements are stored in a table (2). The *CIOManager* is in charge of instantiating windows (3). The *FUIwindow* request a Box instantiation (4), the *CIOManager* generates a Box instance (5) and a message to the *FUIWindow* (6) to add the Box. The FUIBox requires a text component (7). The *CIOManager* generates a Text Component instance (8). This process continues until no more elements exist in the file (12). The order in which windows are created is depicted in Figure 6. That corresponds to the hierarchical structure of the source file. Notice how all elements keep a reference to the CIOManager.

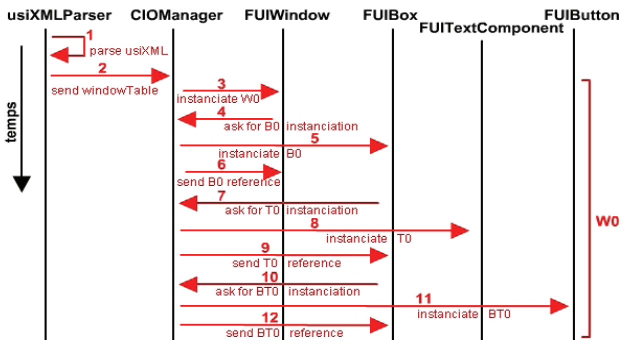


Figure 7 Sequence diagram of the render engine.

Figure 10 and 14 show the result of an application online for a Declaration of a new company to financial administration. The objective is to test the possibility of associating transitions using "tabs" to go from one page to another one. For this example windows transitions, Figure 8, were used.

C. *Behavior definition*

Behavior is defined as a ECA (event, condition action) rule. The renderer supports two types of actions (method call and window transitions). The events list is limited to those events supported in flash. Conditions are stored as strings. Actions Methods calls are links to scripts. Special scripts were created to support window transitions, which includes: open, close, fadeIn, BoxIn, BoxOut and fadeout (Figure 8, 9).

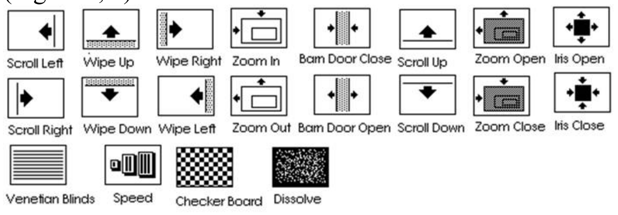


Figure 8. Set of Window transitions implemented

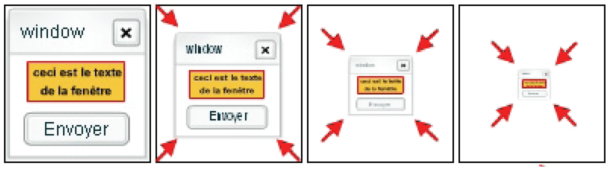
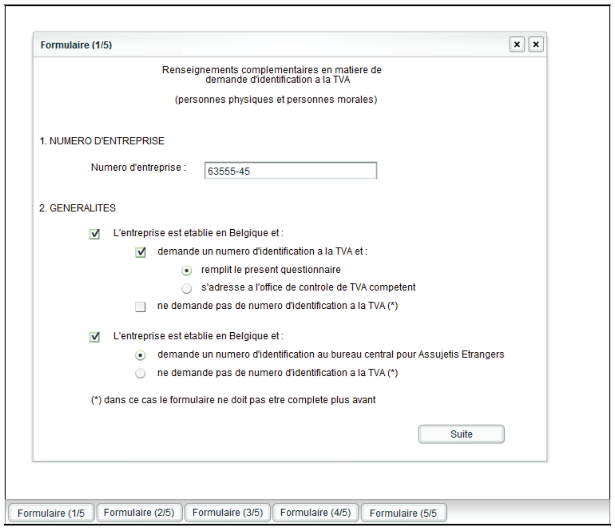


Figure 9. Fadeout Window transition


Formulaire (1/5)

Renseignements complementaires en matiere de demande d'identification a la TVA

(personnes physiques et personnes morales)

1. NUMERO D'ENTREPRISE

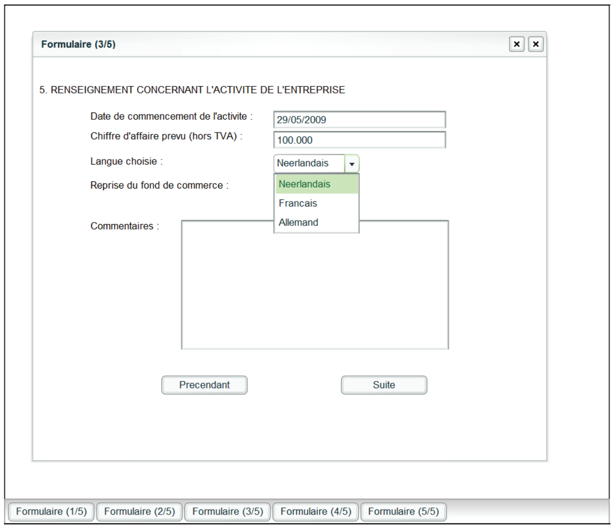
Numero d'entreprise : 63555-45

2. GENERALITES

- [✓] L'entreprise est etablie en Belgique et :
  - [✓] demande un numero d'identification a la TVA et :
    - [✓ remplit le present questionnaire]
    - s'adresse a l'office de controle de TVA competent
  - [ ] ne demande pas de numero d'identification a la TVA (*)
- [✓] L'entreprise est etablie en Belgique et :
  - [✓ demande un numero d'identification au bureau central pour Assujetis Etrangers]
  - ne demande pas de numero d'identification a la TVA (*)

(*) dans ce cas le formulaire ne doit pas etre complete plus avant

Figure 10. Rendering of a form.


Formulaire (3/5)

5. RENSEIGNEMENT CONCERNANT L'ACTIVITE DE L'ENTREPRISE

Date de commencement de l'activite : 29/05/2009
Chiffre d'affaire prevu (hors TVA) : 100.000
Langue choisie : Neerlandais
Reprise du fond de commerce :
Commentaires :

Figure 11. Rendering of a form.

D. *Transformation rules*

Transformation rules from the models to Flash code considered two scenarios: a flash element exist and it was possible to apply a one to one mapping from the CUI element and flash; or there were no correspondence between the CUI element and Flash. In the second scenario there were some attributes that were irrelevant for the interpreted and could be ignored. Those that were more relevant then were built as follows:

- The CUI element can be the result of the composition of existing flash widgets
- The CUI element cannot be the result of a composition of Flash components. Consequently an implementation is needed

More details and the software can be obtained at FlashiXML web site [8], the tool developed as a proof-of-concept.

E. Context adaptation

The multidevice capability of this work is well illustrated in Figure 12. In this case the rendering of the UI varies its size depending on the screen size available, so the window (called calculator) is fully rendered in two context of use (Smart Phone and Pocket PC). In this example just a resize of the UI is appreciated but more complex rules could be apply, notably those of graceful degradation [7] that might replace UI components more drastically when a change in the context is reported. Another example of context adaptation refers to language variations of the UI. This means that the model should be able to capture those variations and consequently support adaptation to the user needs, in this case its language.

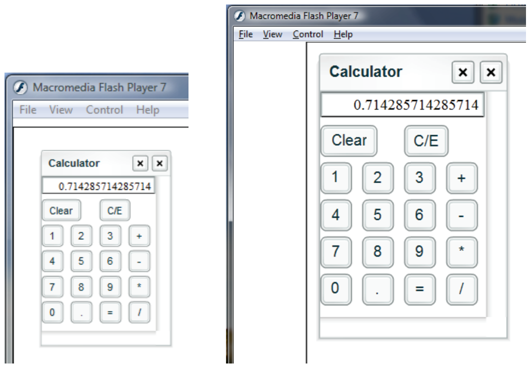


Figure 12. Rendering of a calculator on a SmartPhone and PocketPC.

V. CASE STUDY: VIRTUAL STORE

In this section a case study is presented as proof of the concept discussed in this paper. The Virtual Store is a common application used to test ergonomics of Information Systems. Than the first phase, Figure 13, the user selects a CD and drags it to the shopping chart. Is the CD is drop in the shopping chart then it became part of the shopping list. The cost of the CDs is automatically updated when a CD is added or removed. An excerpt of the corresponding code illustrates the specification of the window transition and the behavior definition.

```
<button defaultContent="Go">
 <behavior>
  <event eventType="depress" eventContext="button_go" />
  <condition/>
  <action id="actI8">
      <transition transitionIdRef="Tr1" />
  </action>
 </behavior>
</button>
…..
<graphicalTransition id="Tr1" transitionType="fadeOut">
        <source id="button_go" />
         <target id="box_all1" />
 </graphicalTransition>
```

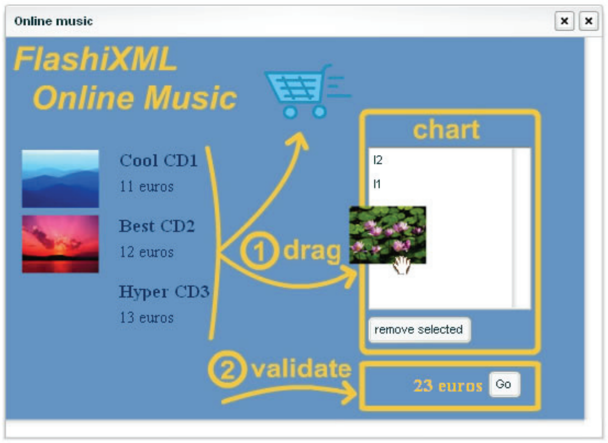


Figure 13. Virtual Store: Music selection window.

Once selected the CDs, the user pass to the checkout screen, Figure 14, where the customer introduces its personal data, credit card data and send its information.

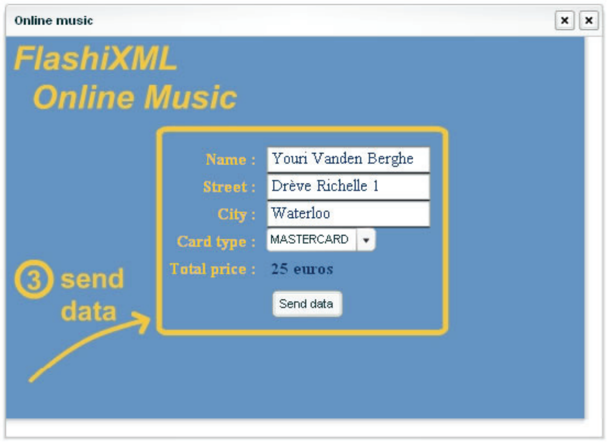


Figure 14. Virtual Store checkout window.

An excerpt of the code that corresponds to the checkout screen is shown below.

```
<box type="horizontal">
 <textComponent defaultContent="Name :"/>
 <textComponent textSize="16"/>
</box>
<box type="horizontal" id="box_text2">
 <textComponent defaultContent="Street :" />
 <textComponent textSize="16" />
</box>
<box type="horizontal">
 <textComponent defaultContent="City :" />
 <textComponent textSize="16" />
</box>
<box type="horizontal">
 <textComponent defaultContent="Card type :" />
  <ComboBox>
       <item defaultContent="VISA"/>
       <item defaultContent="MASTERCARD"/>
  </comboBox>
</box>
<box type="horizontal">
 <textComponent defaultContent="Total price :" />
```

```
  <textComponent defaultContent="25 euros" />
</box>
<box type="stack">
  <button defaultContent="Send data" />
</box>
```

VI. CONCLUSION

The objective of the present work was to contribute on the life cycle UI development for vectorial UIs. Particularly by reducing the time needed to maintain and update a UI. Two aspects of the proposed method aimed at contributing to this objective. The first refers to the selected target language, Macromedia Flash. That assures portability of the solutions generated in FlashiXML. The second aspect corresponds to the need of an UIDL independent of the context of the execution.

The future directions may include: improve the layer that interconnects the UI with the scripts, support dynamic changes on the user interface and a graceful degradation [9] when needed, investigate how to integrate new releases of Flash for Rich Internet Applications.

ACKNOWLEDGMENT

We gratefully acknowledge the support of the ITEA2 Call 3 UsiXML Project under Contract #2008026, the Alban program supported by European Commission, and the CONACYT program supported by the Mexican government. We also thank Youri Vanden Berghe for implementing FlashiXML, as reported in [27].

REFERENCES

[1] Allaire, J., Macromedia Flash MX- A next-generation rich client, White paper, Macromedia, March 2002, available on-line at: http://www.adobe.com/devnet/flash/whitepapers/richclient.pdf

[2] Barry, C. and Lang, M., A Survey of Multimedia and Web Development Techniques and Methodology Usage, *IEEE Multimedia*, Vol. 8, No. 2, 2001, pp. 52–60.

[3] Bodart, F. and Vanderdonckt, J., On the Problem of Selecting Interaction Objects, Proc. of BCS Conf. HCI'94 "People and Computers IX" (Glasgow, 23-26 August 1994), Cambridge University Press, Cambridge, 1994, pp. 163–178.

[4] 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*, Vol. 15, No. 3, June 2003, pp. 289–308.

[5] Collignon, B., Vanderdonckt, J., and Calvary, G., An Intelligent Editor for Multi-Presentation User Interfaces, Proc. of 23rd ACM Symposium on Applied Computing SAC'2008 (Fortaleza, 16-20 March 2008), ACM Press, New York, 2008, pp. 1634–1641.

[6] Contantine, L. and Lockwood, L., Software for Use: A Practical Guide to the Models and Methods of Usage-Centred Design, Addison-Wesley, Reading, 1999.

[7] Dragicevic, P., Chatty, S., Thevenin, D., Vinot, J-L., Artistic Resizing: A Technique for Rich Scale-Sensitive Vector Graphics, Proc. of ACM Symposium on User Interface Software Technology UIST'2005, ACM Press, New York, 2005, pp. 201–210.

[8] FlashiXML web site, available on-line at: http://www.usixml.org/index.php?mod=pages&id=24

[9] Florins, M., Montero, F., Vanderdonckt, J., and Michotte, B., Splitting Rules for Graceful Degradation of User Interfaces, Proc. of 8th Int. Working Conf. on Advanced Visual Interfaces AVI'2006 (Venezia, 23-26 May 2006), ACM Press, New York, 2006, pp. 59–66.

[10] Flex overview, Adobe Systems Incorporated, 2009, available online at http://www.adobe.com/products/flex/overview/

[11] Forbrig, P., Müller, A. and Cap, C.H., Model-based user interface design using markup concepts., Proc. of 8th Int. Workshop on Design, Specification, Verification of Interactive Systems DSV-IS'2001 (Glasgow, 13-15 June 2001), LNCS, Vol. 2220, Springer-Verlag, Berlin, 2001, pp. 16–27.

[12] Haxe Development Document, HaXe, 2009, available on-line at: http://haxe.org/doc

[13] Lang, M. and Fitzgerald, B., New Branches, Old Roots: A Study of Methods and Techniques in Web/Hypermedia Systems Design, *Information Systems Management*, Vol. 23, no. 3, 2006, pp. 62–74.

[14] Limbourg, Q., Vanderdonckt, B., Michotte, B., Bouillon, L., and Lopez, V., UsiXML a language supporting multi-path development of user interfaces, Proc. of 9th IFIP Working Conf. on Engineering for Human-Computer Interaction jointly with 11th Int. Workshop on Design, Specification, and Verification of Interactive Systems EHCI-DSVIS'2004 (Hamburg, 11-13 July 2004), LNCS, Vol. 3425, Springer-Verlag, Berlin, 2004, pp. 200–220.

[15] Luyten, K., Vandervelpen, C., and Coninx, K., Adaptable user interfaces in component based development for embedded systems., Proc. of the 9th Int. Workshop on Design, Specification,and Verification of Interactive Systems DSV-IS'2002 (Rostock, 12-14 June 2002), LNCS, Vol. 2545, Springer, Berlin, 2002, pp. 44–58.

[16] Moshchuk, A., Gribble, S.D., and Levy, H.M., Flashproxy: transparently enabling rich web content via remote execution, Proc. of 6th Int. Conf. on Mobile systems, applications, and services MobiSys'2008 (Breckenridge, 17-20 June 2008), IEEE, pp. 81–93.

[17] Nielsen, J., Flash: 99% bad, AlertBox, 29 October 200, available on-line at: http://www.useit.com/alertbox/20001029.html

[18] OpenLaszlo Application Developer's Guide, Laszlo Systems Inc., 2009, available online at: http://www.openlaszlo.org/lps4.2/docs/developers/architecture.html

[19] Parameswaran, V., Multilingual Flash applications, Proc. of ACM Conf. on Computer Graphics and Interaction Techniques SIGGRAPH'2002 (San Antonio, 21-26 July 2002), ACM Press, New York, p. 316.

[20] Paternò, F. and Giammarino, F., Authoring interfaces with combined use of graphics and voice for both stationary and mobile devices, Proc. of ACM Conf. on Advanced Visual Interfaces AVI'2006, ACM Press, New York, 2006, pp. 329–333.

[21] Paulson, L.D., Building Rich Applications with Ajax, *IEEE Computer*, Vol. 38, No. 10, Oct. 2005, pp. 14–17.

[22] Pleuß, A., MML: A Language for Modeling Interactive Multimedia Applications, Proc. of 7th IEEE Int. Symposium on Multimedia ISM'2005 (Irvine, 12-14 December 2005), IEEE Computer Society, Los Alamitos, 2005, pp. 465–473.

[23] Pleuß, A., Vitzthum, A., and Hussmann, H., Integrating Heterogeneous Tools into Model-Centric Development of Interactive Applications, Proc. of 10th Int. Conf. on Model Driven Engineering Languages and Systems MoDELS'2007 (Nashville, 30 September – 5 October 2007), Lecture Notes in Computer Science, Vol. 4735, Springer-Verlag, Berlin, 2007, pp. 241–255.

[24] Puerta, A.R. and Hu, M., UI Fin: a process-oriented interface design tool, Proc. of ACM Conf. on Intelligent User Interfaces IUI'2009, ACM Press, new York, 2009, pp. 345–354.

[25] Rossi, G., Pastor, O., Schwabe, D., Olsina, L., Web Engineering: Modelling and Implementing Web Applications, Human-Computer Interaction Series, Springer, Berlin, 2008.

[26] Souchon, N. and 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 (Madeira, 4-6 June 2003), Lecture Notes in Comp. Science, Vol. 2844, Springer-Verlag, Berlin, 2003, pp. 377–391.

[27] Vanden 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.

[28] Vanderdonckt, J., A MDA-Compliant Environment for Developing User Interfaces of Information Systems, Proc. of 17th Conf. on Advanced Information Systems Engineering CAiSE'05 (Porto, 13-17 June 2005), Lecture Notes in Computer Science, Vol. 3520, Springer-Verlag, Berlin, 2005, pp. 16–31.