

Model-Driven Engineering of Multi-Target Plastic User Interfaces

Benoît Collignon¹, Jean Vanderdonckt¹, Gaëlle Calvary²

¹*Université catholique de Louvain, Louvain School of Management*

Place des Doyens, 1 – B-1348 Louvain-la-Neuve (Belgium)

²*Université Joseph Fourier – Grenoble I, Laboratoire LIG*

385, rue de la Bibliothèque BP 53 - F-38041 Grenoble Cedex 9 (France)

jean.vanderdonckt@uclouvain.be – gaëlle.calvary@imag.fr

Abstract

A Multi-target user interface is composed of a series of interconnected variations of the same user interfaces, but tailored for different targets or different contexts of use. When access to software applications must be guaranteed in more than one context of use, it is necessary to adapt these user interfaces in order to preserve their usability when the switch between contexts occur. For this purpose, this paper proposes a model and a presentation technique to express and manipulate the plasticity domain of a user interface. The plasticity domain denotes the set of contexts of use it is able to cover while preserving its usability. In this paper, we focus on one aspect of the context of use: the platform screen size. A window requires a graphical area for its rendering and manipulation by the end-user. The model supports the definition of this graphical area in terms of window size and window place. The visualization technique helps in both making observable the set of presentations that fit the available space, and perceiving which operations could help in switching from one presentation to another one. The first benefit is powerful for eliciting the candidate presentations when the context of use changes. The model has been integrated in UsiXML, a XML-compliant user interface description language.

1. Introduction

One major issue with intelligent user interfaces is the diversity of contexts of use in terms of user, platform, and environment [9,16]. Traditional case per case approaches are outdated because of their development and maintenance costs. Several solutions have been proposed to this problem, each has advantages, as limitations. Our alternative is based on plasticity of User Interfaces (UI) [4,17], i.e. the capacity of a UI to withstand variations of contexts of use while preserving usability. We focus on the plasticity of Graphical User Interfaces (GUIs) made of a unique window but with multiple presentations. We hereby refer to *multi-presentation UI* as a UI exhibiting the capability of conveying the same information, but with different presentations depending on the constraints imposed by the platform. Plasticity relies on a set of pre-designed presentations. Each individual presentation is meta-

described with its plasticity domain, i.e. the set of contexts of use it is able to cover while preserving its usability [4]. The context of use is limited to the window size. The first part is devoted to related work. Section 2 presents a running example for illustration of our alternative. Then, we motivate a visualization technique (Section 3) and a model (Section 4) for dealing with multiple presentations. The Section 5 is devoted to tool support. It gives rise to a set of perspectives (Section 6) that are summarized in the conclusion (Section 7).

2. Related Work

In general, four kinds of tools for developing context-aware user interfaces can be distinguished: language based tools, application framework, automatic generators, and interactive tools. In the first case but the oldest technique too, the designer specifies the user interface in a special-purpose language. This language can take many forms, including context-free grammars, state-transition diagrams, declarative languages, event languages. The language is used to specify a UI [1,2].

Application frameworks offers important parts of an application, such as the main windows, the commands, etc., and the programmer so specializes these classes to provide the application-specific details, such as what is actually drawn in the windows and which commands are provided, like in CodeWarrior PowerPlant [11]. GADGET [5] consists of a toolkit specially de-signed to support the exploration of optimization as an approach to interface generation. A problem with all of the language-based tools is that the designer must specify a great deal about the placement, format, and design of the UIs. Automatic Generation Tools help solve this problem. Most are model-based. For instance, TERE-SA [12] uses a task model. Others like SUPPLE [6] use also a device and a user model. None of them provides any support for multiple presentations in a way that the UI dynamically change when the context changes. Those tools do however produce UIs for multiple targets, but they are not combined together.

Interface tools allow the designer to select from a pre-defined library (toolkit) of widgets, and place them on the screen to create dialog boxes, menus and windows. Some generate a description of the interface in a language that can be read at run-time. For example,

GrafiXML generates a UsiXML description [10]. These tools provide little guidance on creating usable UIs, but they cannot handle widgets that change dynamically. E.g., if the contents of a menu or the layout of a dialog box changes based on program state, this must be programmed by writing code.

Our solution can be situated in the Interactive Tools category [14,15]. It helps to define plasticity domains that are linked to specified presentations, each of them created thanks to a predefined library. However, some properties allow it to do more than a simple interactive tool. First of all, the language used to generate the multi-presentation UI, UsiXML, is model-oriented and multi-platform. Every interpreter could render the UIs generated using our solution. TERESA, for instance, can generate the code of the UI in some language but it is not universal. Finally, a lot of interesting features that come from automatic generation tools could be easily integrated in our tool because it is model-driven. Another manifestation of multi-presentation UIs also exist through adaptive layout [10], Art Resizing [5], and multi-device presentations [13].

3. Running Example

Let us consider FlexClock [7], a multi-presentation UI displaying the current time and date with various levels of usability according to the screen size of the window. Sixteen presentations have been designed (Fig. 1): one is displayed at a time. This application is multi-platform as it runs on top of Tcl/Tk which is available for Windows, Linux, and Mac platforms [13]. It is adaptive in the sense that the best presentation is selected at run-time depending on the screen size available on the platform. Of course, if the screen size is expanded or reduced, another presentation is selected and displayed so as to best fit the screen real estate.



Figure 1. FlexClock - Some possible presentations.

When the user resizes the window and the window size over-steps the plasticity domain of the current presentation, then another presentation (the most appropriate one) has to be selected and displayed. The model and the visualization technique presented in this paper aim

at supporting this choice of reaction when the context of use changes.

4. Design process

Plasticity can be modeled as a Finite State Machine (FSM). A FSM is defined by a *model of computation* consisting of a set of *states*, a *start state*, an input and output *alphabet*, and a *transition function* that maps input symbols and current states to a *next state*. Computation begins in the start state with an input string. It changes to new states depending on the transition function. There are many variants, for instance, machines having actions (outputs) associated with transitions (*Mealy machine*) or states (*Moore machine*), multiple start states, more than one transition for a given symbol and state (*nondeterministic finite state machine*), one or more states designated as *accepting states* (*recognizer*), etc [8]. When applied to plasticity, the corresponding alphabets for FSM are, on one hand, the events triggering the changes of the context of use (the six window resizing operations in FlexClock - Fig. 2) and on the other hand, the available presentations (the sixteen windows of FlexClock (some of them being in Fig. 1). In the two following subsections, we investigate Moore Machines and Mealy Machines as visualization techniques for specifying plasticity domains and their application to the running example.

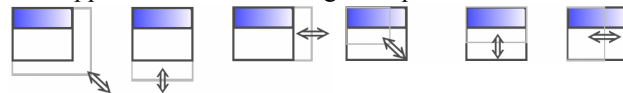



Figure 2. Window resizing operations.

4.1 Moore Machines

Finite State Machines (FSMs) have been used for specifying the dialog [18], the navigation of a UI. A FSM is defined as a model of computation consisting of a set of states, a start state, an input alphabet, and a transition function that maps input symbols and current states to a next state. In UI design, the Moore machine has been used almost everywhere with its associated shortcomings. In Moore machines, states typically represent UI parts and transitions denote navigation between these parts. In our usage, states represent one presentation at a time and transitions depict the presentation resizing operations that may trigger a change of context of use. Fig. 3 shows how FlexClock changes when the window is vertically shrunk. Only four states are considered: {W2; W4; W8; W12}. The input alphabet is limited to the vertical shrinkage: {}. Fig. 3 shows that W12 can be shrunk into W8; W8 can be shrunk into W4 that can give rise to W2. Fig. 4 represent all the transitions possible between all presentations: it becomes unreadable when the number of states (only sixteen windows here) or transitions (only six operations here) increases. As a result, another visuali-

zation technique has to be investigated. Next section deals with Mealy Machines, which have never been used for UI design before.

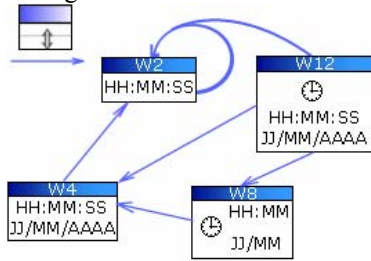




Figure 3. A Moore Machine-based representation illustrated on the vertical shrinking.

4.2 Mealy Machines

In Mealy Machines, states are resizing operations and transitions are composed of a GUI source and destination (denoted source/destination in Fig. 5). In Fig. 5, the states are the vertical shrinking , plus the start state . The input and output alphabets are made of the four considered presentations: {W2; W4; W8; W12}. At launch, W2 is the current presentation. It can be shrink into itself (W2 / W2). W8 can be shrink into W2 (W2; W8 / W2) or W4 (W8 / W2; W4). W12 can be shrink into W8, W4 and W2 (W12 / W8; W4; W2). Fig. 6 shows the complete Mealy machine of FlexClock. The transition conditions are mentioned on the arrow in case of simple expressions (Ws/Wd). Otherwise (complex or multiple expressions), the arrows are decorated by two numbers: the number of transition conditions and a reference to Fig. 7. For instance, there are eleven possible transitions between the vertical and horizontal shrinkages. They are elicited in the sixth area of Fig. 7. Compared to the Moore machine representation, one major advantage of Mealy machines is the factorization that may decrease the number of transitions between states. But there are two drawbacks: first, the representation isn't natural for a human being; secondly, it is not self-contained (Fig. 7 is necessary for describing the transition). We can see here that the Mealy machine corresponding to the same dialog is more compact than the Moore machine, but requires some ability to switch from one usual representation to another one that is less usual. We address this problem by introducing a new visualization technique.

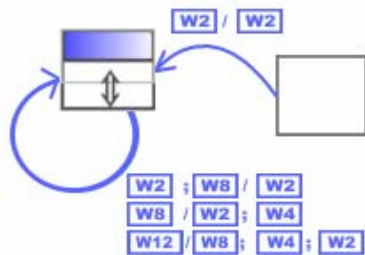


Figure 4. A Mealy Machine-based representation illus-

trated on the vertical shrinking.

4.3 Towards a new visualization technique

We propose a 2D representation for the plasticity domain of a GUI. Each window is positioned at the origin and its plasticity domain is represented as a colored area. In Fig. 8, FlexClock can be resized from (100, 50) to (250,150). In general, the plasticity domain of a window is a quarter of plan but it could be defined as any shape. In Fig. 8, it is a rectangle: the window size can not exceed (250,150). Fig. 9 represents the plasticity domains of {W2; W4; W8; W12}.

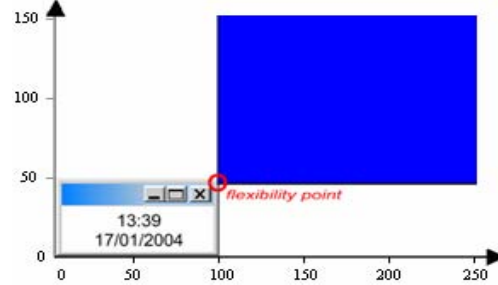


Figure 5. A 2D representation illustrated on one window.

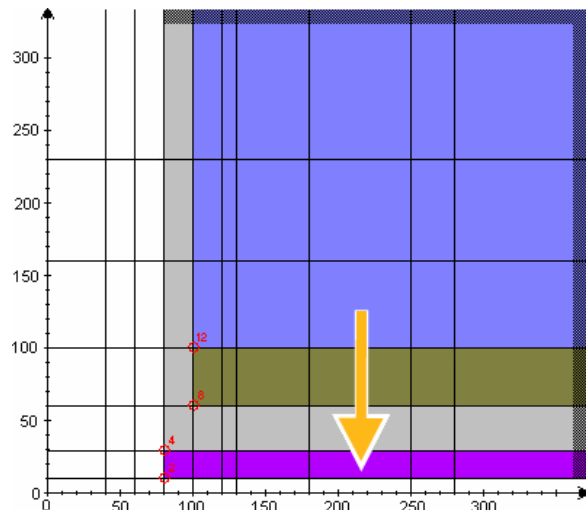


Figure 6. The visualization technique on FlexClock.

1. Vertical Enlargement after opening. W2 / W4; W7	
2. 2-dimensional Enlargement after opening. W2 / W5; W6; W8; W9; W10; W11; W12; W13; W14; W15; W16	
3. Vertical Enlargement W1 / W3 / W3 W2 / W4; W7 W4 / W7 W5 / W9; W13 W6 / W10; W14; W15 W8; W12 / W12 W10 / W14; W15 W13 / W13 W14; W15 / W15 W16 / W16	4. Vertical Shrinkage W1 / W1 W2; W4; W5; W6 / W2 W3 / W3 W7 / W3; W1 W8 / W4; W2 W10; W11 / W6; W2 W12 / W8; W4; W2 W13 / W9; W5; W2 W14 / W10; W6; W2 W15 / W14; W10; W6; W2 W16 / W14; W13; W12; W7; W3
5. Horizontal Enlargement W1; W2 / W2 W3 / W4; W5; W6 W4 / W5; W6 W7 / W8; W9; W10; W11 W12 / W11; W13; W14 W13 / W11; W14 W14 / W11 W15; W16 / W16	6. Horizontal Shrinkage W1; W2 / W1 W3; W7 / W3 W5 / W4; W3 W6 / W5; W4; W3 W8; W12 / W7; W3 W10 / W9; W8; W7; W3 W11 / W10; W9; W8; W7; W3 W13 / W12; W7; W3 W14; W15 / W13; W12; W7; W3 W16 / W14; W13; W12; W7; W3
7. 2-dimensional Enlargement W1 / ALL W2 / W4; W5; W6; W8; W9; W10; W11; W12; W13; W14; W15; W16 W3 / W4; W5; W6; W7; W8; W9; W10; W11; W12; W13; W14; W15; W16 W4 / W5; W6; W8; W9; W10; W11; W12; W13; W14; W15; W16 W5 / W6; W10; W11; W14; W15; W16 W6 / W11; W16 W7 / W8; W9; W10; W11; W12; W13; W14; W15; W16 W8 / W9; W10; W11; W12; W13; W14; W15; W16 W9 / W10; W11; W13; W14; W15; W16 W10 / W11; W14; W15; W16 W11; W15; W16 / W16 W12 / W13; W14; W15; W16 W13; W14 / W15; W16	
8. 2-dimensional Shrinkage W1; W2; W3 / W1 W4 / W3; W2; W1 W5 / W4; W3; W2; W1 W7 / W3; W1 W8 / W7; W4; W3; W2; W1 W9; W12 / W8; W7; W4; W3; W2; W1 W10 / W9; W8; W7; W6; W5; W4; W3; W2; W1 W11 / W10; W9; W8; W7; W6; W5; W4; W3; W2; W1 W13 / W12; W9; W8; W7; W4; W3; W2; W1 W14 / W13; W12; W10; W9; W8; W7; W6; W5; W4; W3; W2; W1 W15 / W14; W13; W12; W10; W9; W8; W7; W6; W5; W4; W3; W2; W1 W16 / ALL except W15	

Figure 6. The Mealy machine representation in a text.

Thanks this presentation, starting with W12, a switch to W8, then W4, then W2 is required when vertically shrinking the window. It is also obvious that a 2D-enlarging is necessary to achieve W8 starting with W2. Fig. 10 illustrates the full running example: the sixteen plasticity domains are represented, each region being attached to a given UI. The figure remains more readable. The technique can be tuned to take into account other properties or attributes.

For instance, Fig. 11 focuses on the user task experience. It shows that the light grey window is appropriate for an experience ranging from two to four. In practice, the presentations can be combined to express the relevant dimensions of the context of use. An attention must be paid in order to preserve a 2D-representation. The next section presents the underlying model.

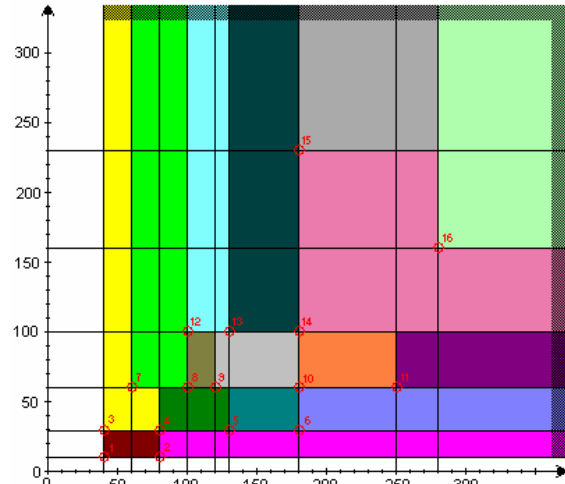


Figure 8. The proposed visualization technique illustrated on the full example of FlexClock.

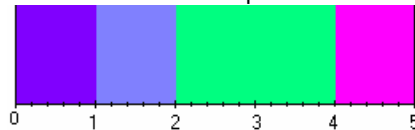


Figure 9. A 1D-representation for the task user experience.

5. The Model

This section is two-fold: in a first part, it proposes a model of the plasticity domain; in a second one, it deals with the switch between presentations. Both of them extend UsiXML [19], an XML language based on the Cameleon Reference Framework [3]. In UsiXML, the final user interface (FUI) refers to the actual UI, which is rendered on a given computing platform either by interpretation (e.g., HTML) or by code compilation (e.g., Java). The concrete user interface (CUI) abstracts the FUI into a definition that is independent of any programming or markup language of any computing platform: it contains a detailed UI description in terms of widgets (concrete interaction objects in UsiXML), layout, navigation and behavior. Concrete interaction objects (e.g., list box, check box, drawing canvas, radio button) are defined with abstract properties and could be arranged to produce a UI. The GrafXML editor has been developed for this purpose and could be freely downloaded from www.usixml.org, as well as the semantics and the syntax of this language.

5.1 Plasticity Domain

Since right now UsiXML does not support specifying multi-presentation UIs or UI with adaptivity, there is a need to expand this specification language with appropriate concepts. The plasticity domain of an interactive system (*PlasticityDomainSet*) is defined as the union of the plasticity domains (*PlasticityDomain*) of its presentations. A *PlasticityDomain* is related to a range of contexts of use, e.g. PDA as defined in

USiXML. The *PlasticityDomain* sets some attributes (aspects) of the platform, user and/or environment (e.g., the screen size). Fig. 12 depicts how USiXML (the bright classes on the left side) has been extended to take into account the plasticity domains. In this figure, the reference to the range of contexts of use is done through the *contextId* attribute in the *PlasticityDomain* class; the cardinality x makes reference to the number of aspects that can be set. Fig. 12 focus on the screen size aspects:

- The *allowedOperations* mention the resizing operations that are allowed on the plasticity domain. The available values are: vertical, horizontal, 2-D shrinkage and vertical, horizontal, 2-D enlargement. For instance, {vertical shrinkage; horizontal shrinkage; 2-D enlargement}.
- The *corners* are the set of points in pixels that define the boundary of the plasticity domain. For instance, {(100,200); (150,200); (150,550); (100,550)} for a rectangle. A *disc* is defined as the pair {(centerX,centerY); radius}. An *ellipse* is defined by a quadruplet {large axes (coord.); short axes (coord.)}. The keywords *ScreenSizeXLimit* and *ScreenSizeYLimit* can be used for non-limited shapes.
- The *shape* is the geometrical shape of the plasticity domain. In practice, the allowed values are: {(right-angled) triangle, (convex/concave) quadrilateral, rectangle, disc, ellipse, (convex/concave) polygon}.

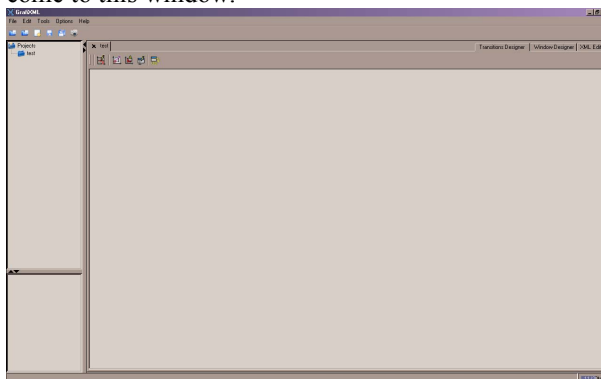
PlasticityDomainSets are not mandatory (it's still possible to build non plastic UIs). For instance, the resolution change has an effect on presentation but the user's task experience is not modified.


5.2 Mapping between plasticity domains and presentations

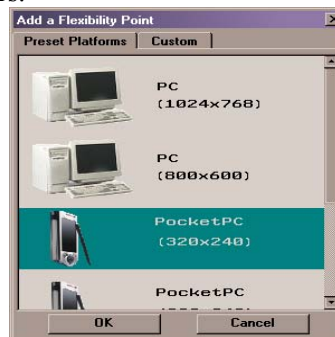
In order to associate a plasticity domain to a presentation, we define a new kind of USiXML inter-model relationship: *isShapedFor* (Fig. 13). *isShapedFor* is defined by a source (a GUI) and a destination (a plasticity domain). It is essential to clearly make the distinction between the relationships *isAdaptedInto* and *isShapedFor*. Hence, *isAdaptedInto* enables to provide a trace of the adaptation of one component in another. So, *isAdaptedInto* expresses the switch between presentations while *isShapedFor* only associates a plasticity domain to a presentation. Thanks to the transformation mechanism that is part of GrafiXML environment, it is possible to save the various adaptations applied to a starting UI and to specify all adaptations in a declarative way instead of developing them all by hand. In this way, the adaptivity mechanism is specified in the UI that could render an appropriate presentation depending on the constraints imposed by the screen of the computing platform. Next section presents our tool.

6. Scenario

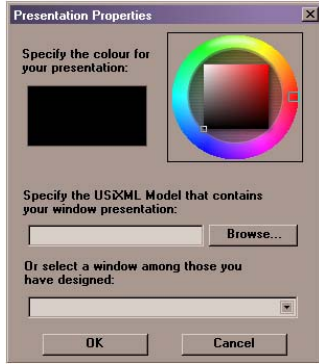
In this section, we detail how to use PlastiXML, a Java editor for multi-presentation user interfaces that generates USiXML specifications corresponding to multiple contexts of use, more particularly multiple domains of plasticity. The aim of the PlastiXML plug-in is to render in the USiXML formal language the different transitions issued by window resizing operations that can exist between the different possible window presentations of a graceful degradation application. First, we start by creating a new project thanks to the GrafiXML wizard. So, we select 'PlastiXML project' and we come to this window:



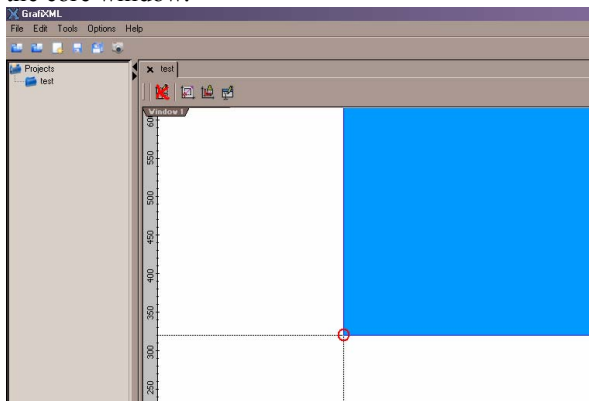
Now, we'll define a new presentation for our Plastic GUI. We can see the only icon we can interact on is . The function hidden behind it consists to help define a new presentation window for our plastic GUI. In fact, by defining a new presentation, we create a new flexibility point. We click on this icon and a new window appears:









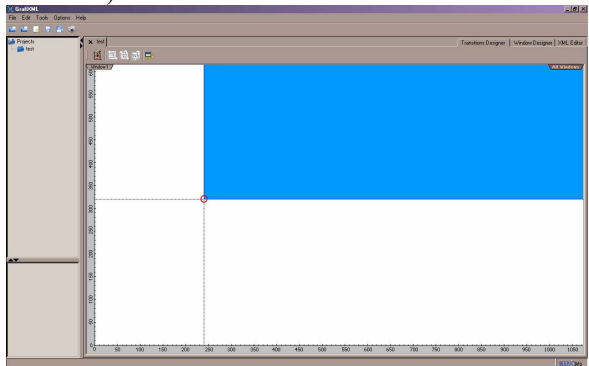
We select the PocketPC Platform with the 320x240 screen resolution and press the button OK. Then, we have to specify the presentation properties (the colour chosen for the surface representing the places where this presentation will be visible and the user interface definition (previously created in GrafiXML). These tasks can be performed inside this window:



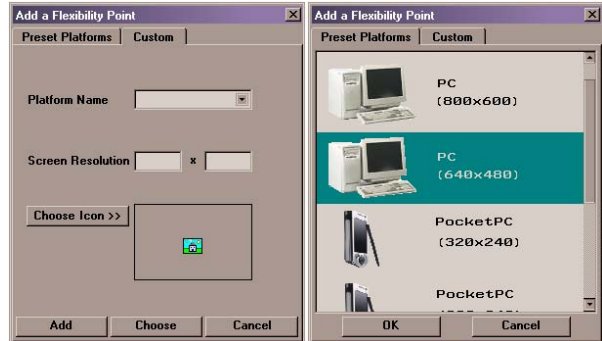
After we selected the colour and the USiXML file representing our window presentation (examples are shown at the end of this document), some results appear on the core window:



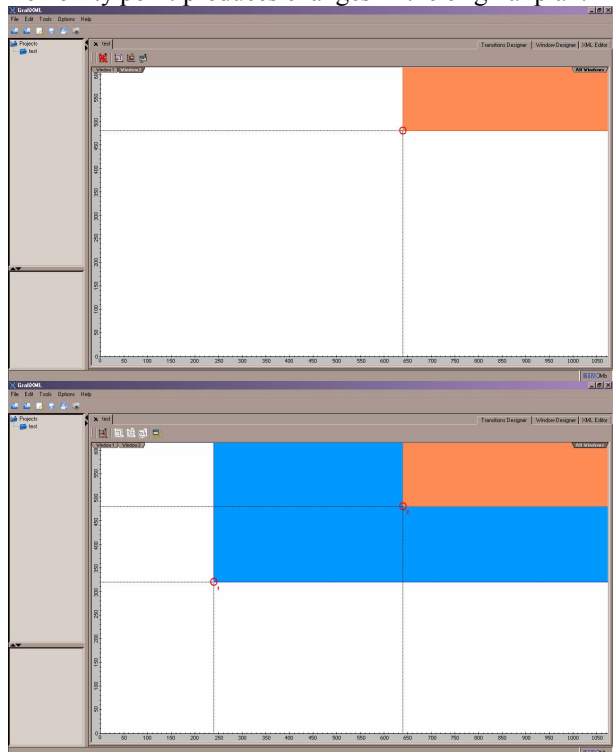
On the current selected window presentation (the selected tab is 'Window 2'), we can now remove this presentation , specify new coordinates for its flexibility point , change the colour of the area representing the part of the screen where this presentation will be visible , change its graphical presentation by specifying another USiXML Model . But if we select the tab 'All Windows', then we have the following options: add a flexibility point , generate the USiXML code (e.g. the transitions between presentation units) .



So, we decide to add a new flexibility point. But we'd like to choose a specific platform which isn't yet in the preset platform list. So, we select the *custom* tab and we fill the form. After that, we can add the specified platform to the preset list or choose it directly. As we know we'll often use this platform, we prefer to add it to the list.

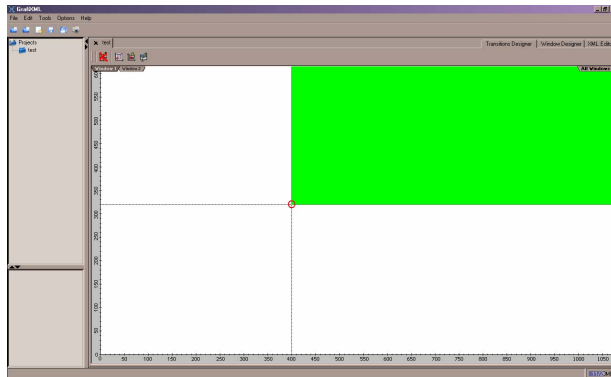
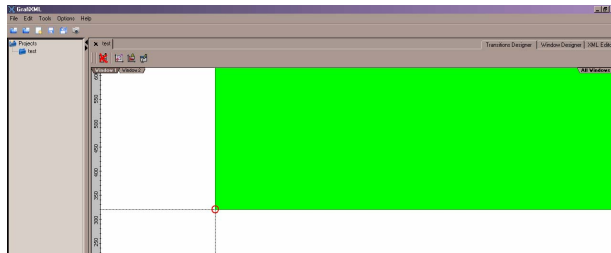
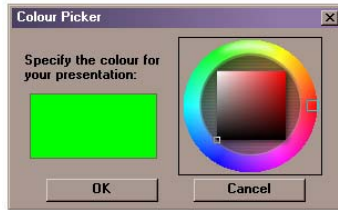


Of course, after specifying the platform, we have to choose the colour and the USiXML definition that will be used for our new window presentation (it can be done by choosing a window among those we have designed in the Window Designer Tab or by specifying a USiXML Description filename that already exists (cf. Figure 5)). Here, we'll follow the first option by selecting windows we previously designed in the Window Designer Tab (cf. *infra*). The introduction of a new flexibility point produces changes in the original plan:

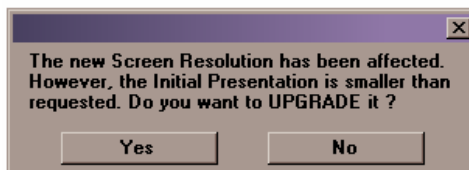


Then, we remember that the GUI defined for the Window/Platform numbered one is graphically smaller

than what the device will provide. So, we modify the properties of this presentation: we select the 'Window 1' tab, we also modify the colour which is not visually pleasant (we'd like to print and show this plan when finished), we change the position of the flexibility point (in fact, the screen space taken by the presentation).

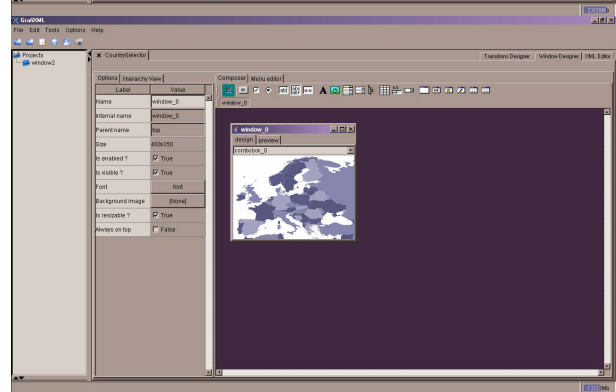
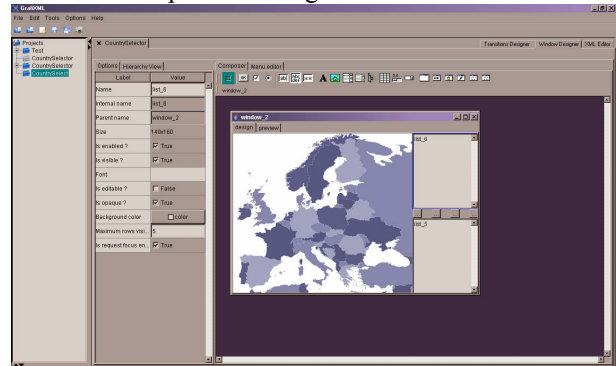


The Flexibility point can be moved from its original position but we mustn't forget the graphical presentation is still the same. So, a new dialog box is opened:



Upgrading it means we'll apply some rules that will modify the original presentation (widget substitution, image replacement/enlargement, etc...) in order to fit with the new screen resolution. But it's not what we

want to do in this specific case. The next figure represents a simple Geographical Information System (GIS) that has been specified using this method.



7. Conclusion

This paper deals with plasticity of User Interfaces. It proposes a model and a visualization technique for managing plasticity domains. Both of them have been implemented in the PlastiXML tool. It helps in defining the plasticity domains of presentation and appreciating the appropriateness of transitions when the context of use changes. Therefore, the following advantages of our approach are not provided by any other tool or method so far: (i) it supports designing multi-presentation UIs by specifying the different presentations and a mechanism for switching from one presentation to another depending on the screen size in a logical way instead of programming everything by hand; (ii) the tool automatically generate (X)HTML or Java (Swing) code corresponding to these UsiXML specifications, which could be reused in other tools of the UsiXML suite; (iii) the code generated intrinsically supports the adaptivity property; (iv) instead of designing all presentations in isolation, it is possible to "copy/paste" a presentation for one resolution to get a starting point for another resolution, thus encouraging reusability; (v) the tool provides the designer with a graphical mechanism to design what kind of presentation is adapted to what kind of resolution. PlastiXML is a

plug-in developed for this purpose in the GrafiXML environment and has been used to develop a multi-presentation on-line course for teaching medical representatives who are using very different platforms. Shortcomings identified so far are: one presentation is viewed at a time thus preventing the designer to easily compare two or more presentations; Mealy machines have been proved more compact to use for specifying all transitions between the presentations, but still remain abstract to be usable in a graphical editor; if a new presentation is defined, PlastiXML does not automatically produce a starting point from a previously existing presentation that could be adapted. Instead, it merely reuses what has been designed so far. The work can be extended in many ways: first of all by applying the model at various granularities from the window to the widget level (this could be powerful for reasoning about detachable user interfaces); secondly, by considering other aspects of the context of use. Today, it is limited only to the platform screen size.

8. References

- [1] Abrams, M., Phanouriou, C., Alan, L., Batongbacal, C., Williams, S.M., and Shuster, J.E., "UIML: An Appliance-Independent XML User Interface Language", Proc. of WWW8.
- [2] Ali M.F., Pérez-Quiñones M.A., and Abrams M., "Building Multi-Platform User Interfaces With UIML", Seffah, A. and Javahery, H. (eds.), *Multiple User Interfaces: Cross-Platform Applications and Context-Aware Interfaces*, John Wiley & Sons, 2003, pp. 95–118.
- [3] 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.
- [4] Calvary, G., Coutaz, J., and Thevenin, D., "Supporting Context Changes for Plastic User Interfaces: a Process and a Mechanism", Proc. of IHM-HCI'01 (Lille, Sept. 2001), Springer, Berlin, 2001, pp. 349–363.
- [5] Dragicevic, P., Chatty, S., Thevenin, D., and Vinot, J.-L., "Artistic resizing: a technique for rich scale-sensitive vector graphics", Proc. of 18th ACM Symp. on User interface software and technology UIST'2006, ACM Press, New York, 2006, pp. 201–210.
- [6] Fogarty, J. and Hudson, S.E., "GADGET: A toolkit for optimization-based approaches to interface and display generation", Proc. of UIST'03, pp. 125–134.
- [7] Gajos, K. and Weld, D.S., "SUPPLE: Automatically Generating User Interfaces", Proc. of IUT'04 (Funchal, January 13-16, 2004), ACM Press, 2004, pp. 93–100.
- [8] Grolaux D., Van Roy P., and Vanderdonckt J., "Flex-Clock: A Plastic Clock Written in Oz with the QTK Toolkit", Proc. of TAMODIA'2002 (Bucharest, July 18-19, 2002), Bucharest, 2002, pp. 135–142.
- [9] Hopcroft, J.E. and Ullman, J.D., *Introduction to automata theory, languages, and computation*, Addison-Wesley, Reading, 1979.
- [10] Keränen, H., Plomp, J., "Adaptive runtime layout of hierarchical UI components", Proc. of the 2nd Nordic conference on Human-computer interaction NordiCHI'2002, ACM Press, New York, 2002, pp. 251-254.
- [11] Kray, C., Wasinger, R., and Kortuem, G., "Concepts and issues in interfaces for multiple users and multiple devices", Proc. of Workshop on Multi-User and Ubiquitous User Interfaces M3UI'04, 2004.
- [12] Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., and Lopez, V., "UsiXML: a Language Supporting Multi-Path Development of User Interfaces", Proc. of EHCI-DSV-IS'2004 (Hamburg, July 11-13, 2004), Springer, Berlin, 2005, pp. 200–220.
- [13] Luyten, K., Thys, K., Vermeulen, J., and Coninx, K., "A Generic Approach for Multi-device User Interface Rendering with UIML", Proceedings of the Sixth International Conference on Computer-Aided Design of User Interfaces CADUI'2006 (Bucharest, 6-8 June 2006), Springer, Berlin, 2007, pp. 175-182.
- [14] Metrowerks, Inc., PowerPlant for CodeWarrior, Austin, 1996, accessible at <http://www.metrowerks.com/>
- [15] Mori, G., Paternò, F., and Santoro, C., "Design and Development of Multidevice User Interfaces through Multiple Logical Descriptions", *IEEE Trans. on Software Engineering* 30,8 (August 2004), pp. 507–520.
- [16] Mozart Consortium, The Mozart Programming System (Oz 3), accessible at <http://www.mozart-oz.org/documentation>.
- [17] Puerta, A.R., "A Model-Based Interface Development Environment", *IEEE Software* 14,4 (July/August 1997), pp. 41–47.
- [18] Puerta, A.R. and Eisenstein, J., "XIML: a common representation for interaction data", Proc. of IUI'2002, ACM Press, New York, 2002, pp. 214–215.
- [19] Seffah, A. and Javahery, H. (eds.), *Multiple User Interfaces: Cross-Platform Applications and Context-Aware Interfaces*, John Wiley & Sons, Chichester, 2003.
- [20] Thevenin, D. and Coutaz, J., "Plasticity of User Interfaces: A Framework and Research Agenda", Proc. of INTERACT'99, IOS Press, Amsterdam, pp. 110–117.
- [21] Sottet, J.S., Model-Driven Engineering of Plastic User Interfaces, Proc. of Interact'2007, Springer, 2007.
- [22] Vanderdonckt J., Limbourg Q., and Florins, M., "Deriving the Navigational Structure of a User Interface", Proc. of INTERACT'2003, 2003, pp. 455–462.
- [23] Vanderdonckt, J., "A MDA-Compliant Environment for Developing User Interfaces of Information Systems", Proc. of CAiSE'05 (Porto, June 13-17, 2005), Springer, Berlin, 2005, pp. 16–31.