

An Intelligent Editor for Multi-Presentation User Interfaces

Benoît Collignon, Jean Vanderdonckt

Louvain School of Management (LSM)
Université catholique de Louvain
Place des Doyens, 1
B-1348 Louvain-La-Neuve (Belgium)
Jean.vanderdonckt@uclouvain.be

Gaëlle Calvary

Laboratoire LIG
Université Joseph Fourier – Grenoble I
385, rue de la Bibliothèque BP 53
F-38041 Grenoble Cedex 9 (France)
Gaelle.Calvary@imag.fr

ABSTRACT

In ubiquitous computing, interactive applications are shipped with different variations of its user interface depending on the constraints imposed by the context in which they are running, such as the user, the computing platform and environment. A multi-presentation user interface is composed of a series of interconnected user interfaces for the same task to be carried out in different contexts of use. When access to software applications must be guaranteed in more than one context of use, it is necessary to automatically adapt the interface in order to preserve their usability when context switching occurs, for instance, a switch from a desktop to a pocket computer. To achieve this goal, this paper proposes a model and a visualization technique to express and manipulate the plasticity domains of a multi-presentation user interface. The plasticity domain denotes the set of contexts of use it is able to cover while preserving its usability. This paper focuses primarily on one aspect of the context of use: the computing platform and its screen size: when the dimensions of a graphical user interface change, the multi-presentation interface automatically switches to the presentation which is the most adapted to this screen. The model supports the definition of this plasticity domain in terms of window size and location. 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 model has been integrated into a user interface description language and is supported by an intelligent editor, because it infers from plasticity domains all the constraints and conditions required for context switching.

Categories and Subject Descriptors

D.2.2 [Software Engineering]: Design Tools and Techniques – *User interfaces*. H.1.2 [Information Systems]: Models And Principles – *User/Machine Systems*. H.5.2 [Information Interfaces and Presentation]: User Interfaces – *Graphical user interfaces, Prototyping, User interface management systems (UIMS)*.

Keywords

Adaptation, context of use, plasticity domain, plastic user interface, user interface extensible markup language.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SAC'08, March 16-20, 2008, Fortaleza, Ceará, Brazil.

Copyright © 2008 ACM 978-1-59593-753-7/08/0003...\$5.00.

1. INTRODUCTION

One major issue in ubiquitous computing is the diversity of contexts of use in which a ubiquitous system may be executed [9], in terms of user, computing platform, and environment [2]. Traditional case per case approaches are outdated because building a User Interface (UI) for each possible context of use would lead to prohibitive development and maintenance costs. Several solutions have been proposed to this problem [1,12,15], each having advantages and limitations. Our alternative is based on the concept of plasticity of UIs [3,17], i.e. the capacity of a UI to withstand variations of contexts of use while predefined properties of preserving usability [3]. This paper focuses on the plasticity of Graphical User Interfaces (GUIs) that are made of a unique window but with multiple presentations, perhaps on different computing platforms. The two other dimensions, i.e., the user and the physical environment, could lead to a generalization of this approach. A *multi-presentation UI* is defined as GUI composed of a series of interconnected user interfaces for a same task to be carried out in different contexts of use, here in different computing platforms, each platform being primarily characterized by its screen resolution. Each individual presentation of the multi-presentation is described with respect to its supported plasticity domain, i.e. the set of contexts of use it is able to cover while preserving its usability [17]. The next section reports on work relate to multi-presentation GUIs. Section 3 presents a running example in order to exemplify our alternative. Then, we motivate a visualization technique (Section 4) and a model (Section 5) for dealing with multiple presentations. The Section 6 is devoted to tool support, including some perspectives for generalization that are summarized in the conclusion (Section 7).

2. RELATED WORK

Our work falls in the category of interface builders. Four tool categories exist: language based tools, application frameworks, automatic generators based on models, and interactive tools.

In *language based tools*, the designer specifies the UI in a special-purpose language, which can take many forms, including context-free grammars, state-transition diagrams, declarative languages, finite state machines [8], and event languages. The language is typically used to specify the UI syntax of the UI.

Application frameworks support important parts of an application, such as the main windows and the commands. The developer 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 [4] consists of a toolkit specially designed to support the exploration of optimization as an approach to interface generation. A potential problem with the aforementioned

kinds is that the developer must specify a great deal about the placement, format, and design of the UIs.

Automatic Generation Tools help solve this problem by adopting a model-based approach: one or many models of the UI are exploited to automatically generate its corresponding code. TERESA [12] basically uses a task model, while SUPPLE [5] use a device and a user model. Early work on model-based approaches for automated UI generation includes ITS [21], Humanoïd and UIDE which merged into MASTERMIND [16], TRIDENT [18], and MOBI-D [14]. They all generate a single UI based on an initial set of models. Later on, Teallach [6] allowed the developer to initiate the development process from the task model, the data model or the presentation model and to complete the other aspects in any order. TERESA [12] generates one or multiple UIs for different platforms based on a task model that is subsequently refined into abstract presentations and UIs [2].

Interface tools allow the developer to select from a predefined library (toolkit) of widgets and place them on the screen to create dialog boxes, menus and windows. Some generate a UI description in a language that can be read at run-time. For example, GrafiXML [20] generates a UsiXML description [10] that can be later interpreted by a rendering engine. These tools provide little guidance on creating usable UIs and they cannot handle widgets that change dynamically. For instance, 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 located primarily in the Interactive Tools category since it helps the developer to define plasticity domains that are linked to specified presentations. However, some properties allow it to do more than a simple interactive tool. The main difference between our alternative and preliminary model-based tools is that all UIs corresponding to different platforms (here, different screen resolutions) are built and interconnected so that they are embedded in one running application instead on multiple ones. For instance, one can specify a UI model in UIML [1] and generate a UI for Java, HTML, or WML, but these UIs are separated. In our alternative, the context is sensed to determine which UI is the most appropriate for and then this UI is selected to be used. If the context of uses changes, that is if the screen resolution of the platform changes, another UI will be selected instead that is adequate to this resolution.

3. RUNNING EXAMPLE

In order to exemplify the development life cycle of a multi-presentation GUI, let us consider FlexClock [7], a multi-presentation GUI displaying the current time and date with various levels of details according to the screen size of the window. Sixteen presentations have been designed, some being reproduced in Figure 1: one presentation is displayed at a time depending on the screen resolution of the platform. For instance, W2 will be displayed on a watch UI, W12 on a mobile phone. If the screen size is expanded or reduced, another presentation is selected and displayed so as to best fit the actual screen. This application is implemented on top of the Mozart environment [13], itself based on Tcl/Tk which is available for Windows, Linux, and Mac platforms. When the user resizes the window and the window size oversteps the plasticity domain of the current presentation, then another presentation is 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.



Figure 1. FlexClock - Some possible presentations.

4. DESIGN PROCESS

By nature, 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)* [8]. When applied to plasticity, the corresponding alphabets for FSM are, on one hand, the events triggering the changes of the context of use, i.e. the six window resizing operations in FlexClock (Figure 2) and on the other hand, the available presentations. In the two following subsections, we investigate Moore Machines and Mealy Machines as visualization techniques for specifying plasticity domains. Their application to FlexClock motivates a proposition that is further addressed.

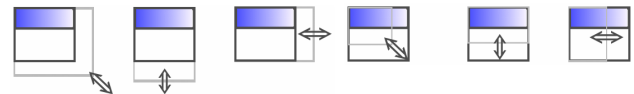


Figure 2. Windows resizing operations.

4.1 Moore Machines

Finite State Machines (FSMs) have been used for specifying the dialog, 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. For instance, Figure 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: {⏏}. Figure 3 shows that W12 can be shrunk into W8; W8 can be shrunk into W4 that can give rise to W2. Figure 4 represent all the transitions possible between all presentations: it is unreadable when the number of

states (only sixteen windows here) or transitions (only six operations here) increases. As a result, another visualization technique has to be investigated. Next section deals with Mealy Machines, which have never been used for UI design before.

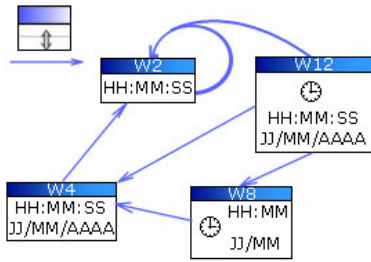


Figure 3. A Moore Machine-based representation illustrated on the vertical shrinking.

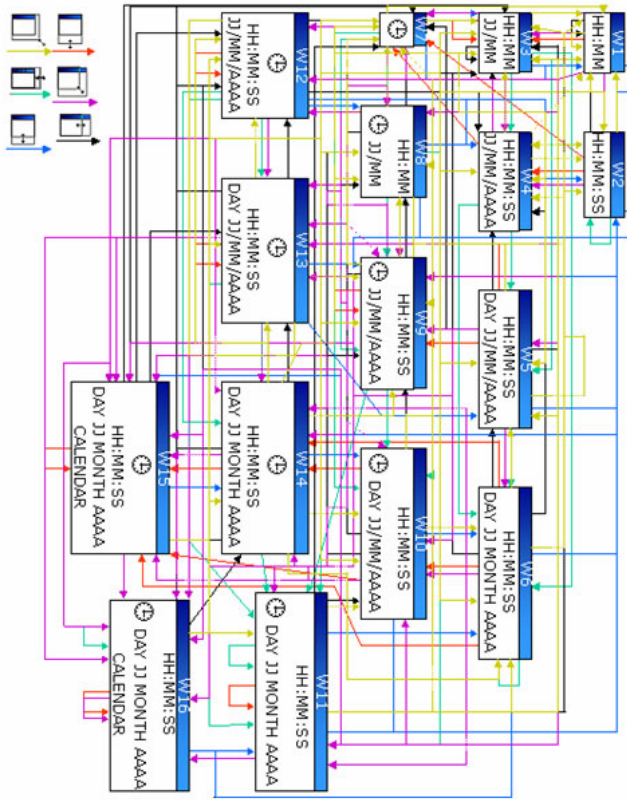




Figure 4. The Moore Machine representation for the full running example FlexClock (16 presentations, 6 operations).

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 figure 5). In Figure 5, the states are the vertical shrinkage , 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). Figure 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 ar-

rows are decorated by two numbers: the number of transition conditions and a reference to Figure 7. For instance, there are eleven possible transitions between the vertical and horizontal shrinkages. They are elicited in the sixth area of Figure 7.

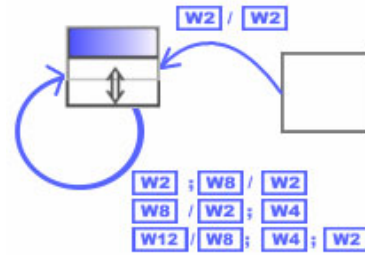


Figure 5. A Mealy Machine-based representation illustrated on the vertical shrinking.

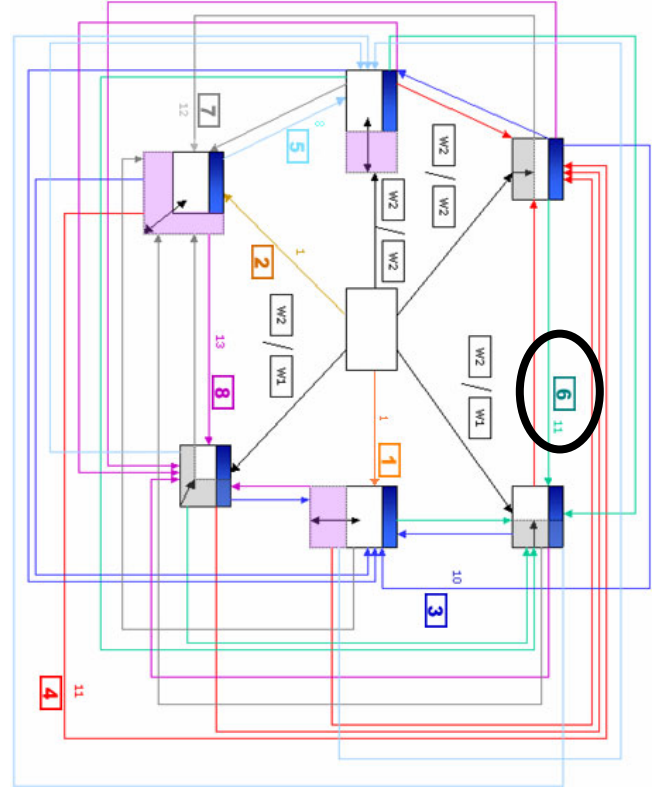


Figure 6. The Mealy Machine for FlexClock.

Compared to the Moore machine representation, one major advantage of Mealy machines is the factoring out decreases the number of transitions between states. But there are two drawbacks: the representation is less natural for a human legibility than in Moore machines and it is not self-contained (Figure 7 is necessary for describing the transition). We can see here that the Mealy machine corresponding to the same dialog is definitely more compact than the Moore machine, but requires some ability to switch from one representation. Therefore, we address this problem by introducing a new visualization technique.

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 Figure 8, FlexClock is 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 Figure 8, it is a rectangle: the window size can not exceed (250,150). Figure 9 represents the plasticity domains of {W2; W4; W8; W12}. Thanks to 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.

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 7. Mealy machine represented in a textual format.

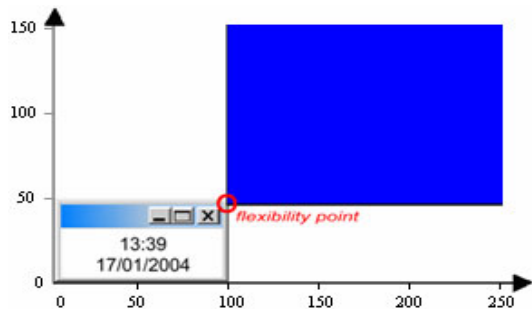


Figure 8. A 2D representation illustrated on one window.

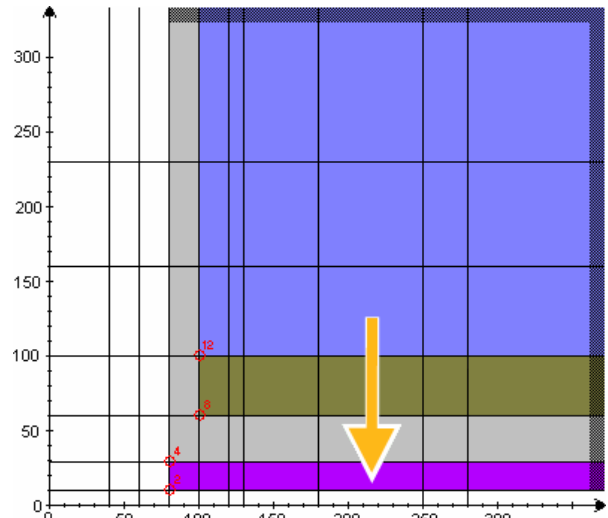


Figure 9. The visualization technique on FlexClock.

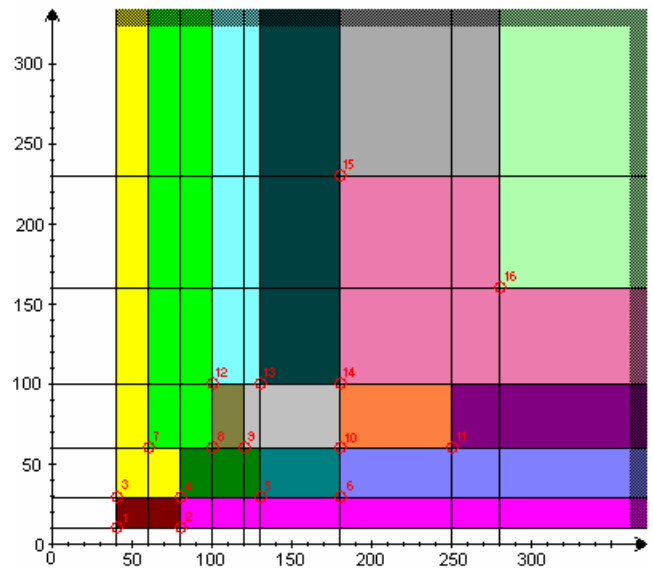


Figure 10. The proposed visualization technique illustrated on the full example of FlexClock.

Figure 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, Figure 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. Next section presents the underlying model.

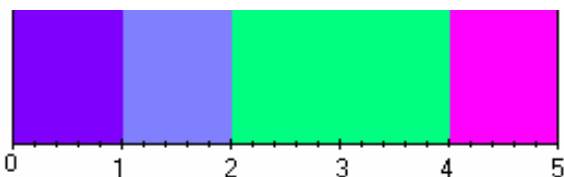


Figure 11. A 1D-representation for the task user experience.

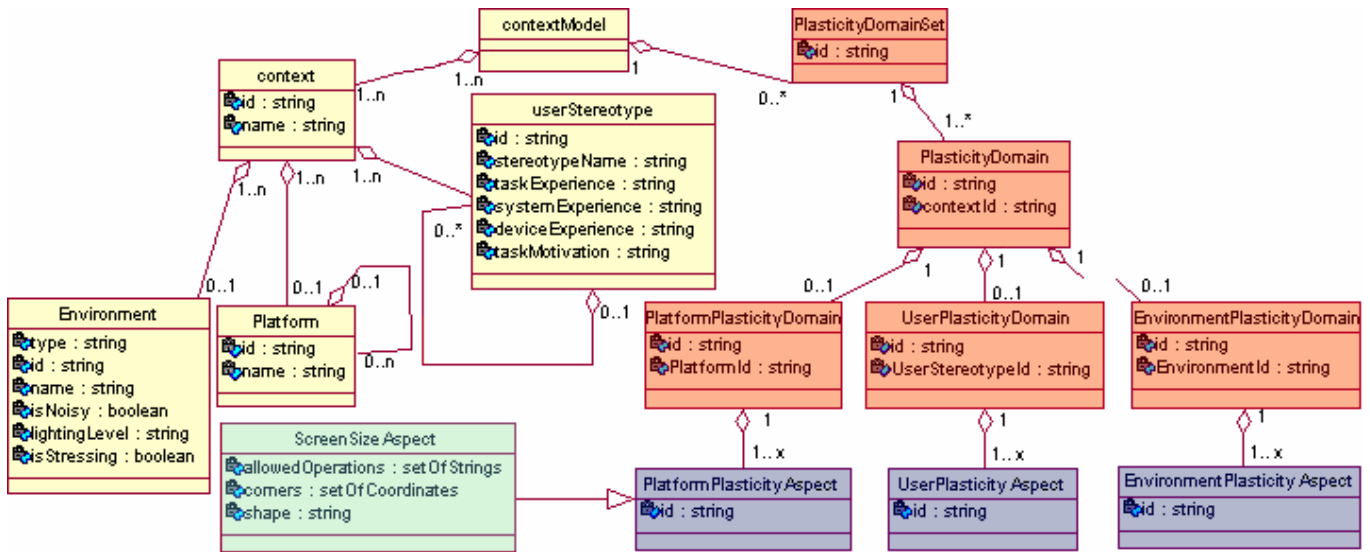


Figure 12. An extension of the UsiXML Context Model in order to incorporate the plasticity domain (in orange).

5. THE MODEL

This section is twofold: 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 [7], an XML language based on the CAMELEON Reference Framework [2]. 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. For this purpose, the GrafiXML editor has been developed, it can be downloaded from www.usixml.org. The UsiXML semantics and the syntax can be found on the web site.

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). Figure 12 shows 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. Figure 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, horizon-

tal, 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}. A formalization could be done using for instance, the *Complex Theory* for concave shapes. But actually, we favor a pragmatic trade-off.

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* (Figure 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, called PlastiXML.

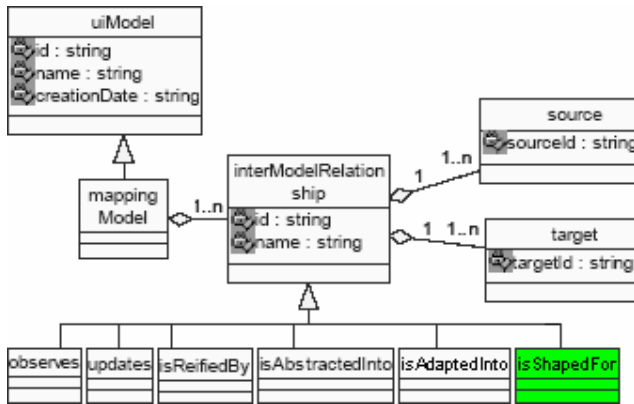


Figure 13. An extension of the USiXML Mapping Model to associate plasticity domains to presentations.

6. TOOL SUPPORT

6.1 PlastiXML

PlastiXML is a GrafiXML plug-in [20] for dealing with plasticity domains. Actually, it is limited to the platform screen size. Figure 14 presents the plug-in's UI. The designer defines the plasticity domains of the presentations he has prefabricated using GrafiXML. The definition is done by direct manipulation using the visualization technique described in the previous section. PlastiXML generates a UsiXML-compliant code defining the plasticity domain of each presentation: the blue region corresponds to one plasticity domain attached to the one presentation and the orange region corresponds to another presentation.

GrafiXML is developed in Java 1.5 and currently consists of about 110,000 lines of code, including the classes dealing with the UsiXML language that are used by PlastiXML. Since it is a plug-in, it is also developed in the same environment. The designer can download the plug-in on demand from the GrafiXML plug-in manager. Two examples are provided in the next section.

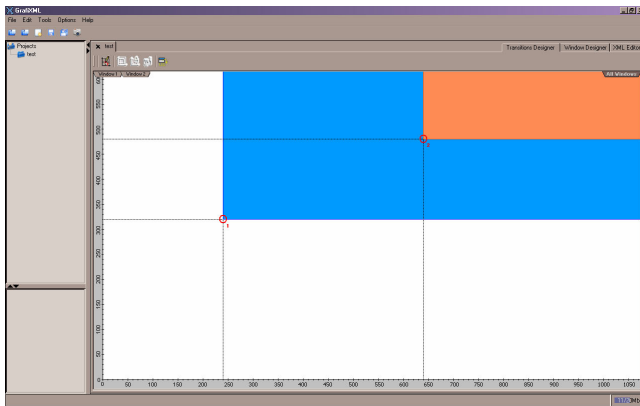


Figure 14. The PlastiXML editor.

6.2 Examples

The first example we consider is the subset of FlexClock limited to {W2; W4}. Figure 15 presents the generated code limited to the Context Model and the Mapping Model of UsiXML. In this first example, we can notice that all the plasticity domains refer to the same context of use as defined in USiXML (a PC).

```

<contextModel id="test-contextModel_14"
  name="test-contextModel">
  <contextResource contextId="cont1">
    <Environment id="envir1" name="envir1" ... />
    <userStereotype id="ustr1" stereotypeName="ustr1" ... />
    <Platform id="plat1" name="plat1">
      <HardwarePlatform ... screenSize="640x480"/>
      <SoftwarePlatform ... />
      <NetworkCharacteristics ... />
      <WapCharacteristics ... />
      <BrowserUA ... />
    </Platform>
  </contextResource>
  <PlasticityDomainSet id="plasts1">
    <PlasticityDomain id="plast1" contextId="cont1">
      <PlatformPlasticityDomain id="plapt1" PlatformId="plat1">
        <ScreenSizeAspect id="scrd1" shape="rectangle"
          corners="{(80,10);(80,30);(ScreenSizeXLimit,10);
            (ScreenSizeXLimit,30)}"
          allowedOperations="{vertical shrinkage}"/>
      </PlatformPlasticityDomain>
    </PlasticityDomain>
    <PlasticityDomain id="plast2" contextId="cont1">
      <PlatformPlasticityDomain id="plapt2" PlatformId="plat1">
        <ScreenSizeAspect id="scrd2" shape="polygon"
          corners="{(80,30);(ScreenSizeXLimit,30);
            (ScreenSizeXLimit,60);(100,60);
            (100,ScreenSizeYLimit);
            (80,ScreenSizeYLimit)}"
          allowedOperations="{vertical shrinkage}"/>
      </PlatformPlasticityDomain>
    </PlasticityDomain>
  </PlasticityDomainSet>
</contextModel>
<mappingModel id="map1" name="map1" ...>
  <isShapedFor id="adp1" name="adp1">
    <source sourceId="W2"/>
    <target targetId="plast1"/>
  </isShapedFor>
  <isShapedFor id="adp2" name="adp2">
    <source sourceId="W4"/>
    <target targetId="plast2"/>
  </isShapedFor>
  <isShapedFor id="adp3" name="adp3">
    <source sourceId="W12"/>
    <target targetId="plast4"/>
  </isShapedFor>
</mappingModel>

```

Figure 15. Example 1 – The code generated by PlastiXML on the FlexClock example limited to two presentations.

The next example concerns didactic contents for students in medicine. The scenario to be supported is the following: students can attend courses in auditorium and continue reviewing the course contents afterwards or during the practical experiments related to a course. For this purpose, a multimedia application has been developed for a PC platform that provides students with explanations of how to conduct a diagnosis on different systems of the human body (e.g., cerebral system, digestive system, respiratory system). This desktop system is only available in computer pools, but not in experiment rooms. Therefore, a PDA version has been developed that offers multiple presentations of the same contents depending on the screen resolution. Our example therefore involves two contexts of use: a PC and a PDA. In the PC version (Context 1), the user has the ability to study her course and get some help by the use of two windows. In the PDA version (Context 2), three windows present the same information but distributed in another way (Figure 16).

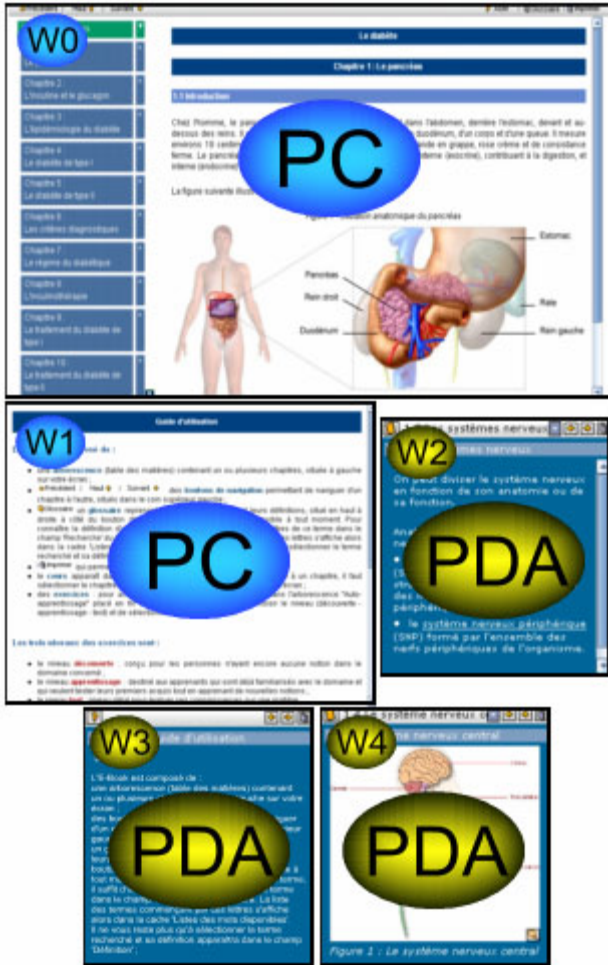


Figure 16. A GUI defined in two specific context of use.

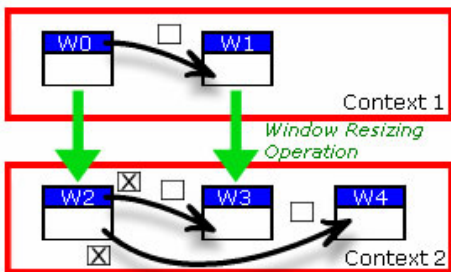


Figure 17. Schematic version of the GUI defined in two specific contexts of use.

Figure 17 depicts two gateways between the two contexts of use: on one hand, the transition between W0 and W2, and on the other hand, the transition between W1 and W3. These gateways must not be confused with dialog transitions [19]. For instance, the accomplishment of the user task presented in W0 triggers the presentation W1. As a result, there are two clusters of presentation: on one hand, W0 and W2; on the other hand, W1 and W3. Figure 18 presents the two corresponding graphics making observable the possible transitions inside a cluster. Arbitrary height and width have been attributed to windows (W0:[800x600]; W1:[480x320]; W2:[240x320]; W3:[240x320]), as well as the allowed resizing operations.

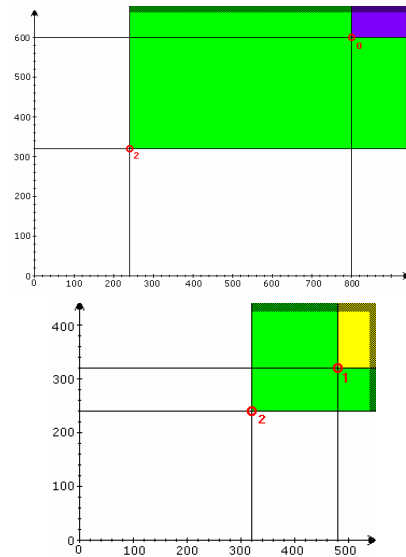


Figure 18. Example 2 – An example involving two contexts of use as defined in USIXML.

```

<cuiModel id="test-cui_14" name="test-cui">
  <window id="window_component_0" name="window_component_0" width="800" height="600"> ... </window>
  <window id="window_component_1" name="window_component_1" width="480" height="320"> ... </window>
  <window id="window_component_2" name="window_component_2" width="240" height="320"> ... </window>
  <window id="window_component_3" name="window_component_3" width="240" height="320"> ... </window>
  <window id="window_component_4" name="window_component_4" width="240" height="320"> ... </window>
  ...
</cuiModel>
<cuiContextModel id="test-contextModel_14" name="test-contextModel">
  <contextRessource contextId="cont1">
    <Environment id="envir1" name="envir1" .../>
    <userStereotype id="uster1" stereotypeName="uster1" .../>
    <Platform id="plat1" name="plat1">
      <HardwarePlatform ... screenSize="1024x768"/>
      <SoftwarePlatform ... />
      <NetworkCharacteristics ... />
      <WapCharacteristics ... />
      <BrowserUA ... />
    </Platform>
  </contextRessource>
  <contextRessource contextId="cont2">
    <Environment id="envir2" name="envir2" .../>
    <userStereotype id="uster2" stereotypeName="uster2" .../>
    <Platform id="plat2" name="plat2">
      <HardwarePlatform ... screenSize="240x320"/>
      ...
    </Platform>
  </contextRessource>
  <PlasticityDomainSet id="plasts1">
    <PlasticityDomain id="plast1" contextId="cont1">
      <PlatformPlasticityDomain id="platp1" PlatformId="plat1">
        <ScreenSizeAspect id="socrd1" shape="rectangle"
          corners=
            "{(800,600);(800,ScreenSizeYLimit);(ScreenSizeXLimit,600);
              (ScreenSizeXLimit,ScreenSizeYLimit)}"
          allowedOperations="{vertical shrinkage ; horizontal
            shrinkage ; 2-D enlargement}" />
      </PlatformPlasticityDomain>
    </PlasticityDomain>
  </PlasticityDomainSet>

```

```

</PlasticityDomain>
<PlasticityDomain id="plast2" contextId="cont2">
  <PlatformPlasticityDomain id="platp2" PlatformId="plat2">
    <ScreenSizeAspect id="schr2" shape="rectangle"
      corners="{(240,320);(400,350);(240,ScreenSizeYLimit);(320,ScreenSizeYLimit);(800,600);(800,ScreenSizeYLimit);(ScreenSizeXLimit,600)}"
      allowedOperations="{vertical shrinkage ; horizontal shrinkage ; 2-D enlargement}"/>
    </PlatformPlasticityDomain>
  </PlasticityDomain>
</PlasticityDomainSet>
</contextModel>

```

Figure 19. Example 2 - The code generated by PlastiXML.

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 GrafXML 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: by applying the model at various granularities of widgets, by considering other media types, by considering other aspects of the context of use.

8. ACKNOWLEDGMENTS

We gratefully acknowledge the support of this research by the MulPlex project funded by First Europe Objectif 1 Initiative (DG TRE, Walloon Region, Belgium). Principal Partners are: MOPSY Company (Soignies, Belgium, www.mopsys.com) and the IHM Team of Laboratoire LIG (Grenoble, France). We also thank the SIMILAR network of excellence, supported by the 6th Framework Program of the European Commission, under contract FP6-IST1-2003-507609 (www.similar.cc). We also thank the anonymous reviewers for their constructive comments.

9. REFERENCES

- [1] Ali M.F., Pérez-Quiñones M.A., and Abrams M. Building Multi-Platform UIs with UIML. In [16], 95–118.
- [2] Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., and Vanderdonck, J. A Unifying Reference Framework for Multi-Target User Interfaces. *Interact. with Comp.* 15,3 (2003) 289–308.
- [3] Calvary, G., Coutaz, J., and Thevenin, D. Supporting Context Changes for Plastic User Interfaces: a Process and a Mechanism. In *Proc. of Joint Conf. IHM-HCI'01* (Lille, Sept. 2001). Springer, Berlin, 2001, 349–363.
- [4] Fogarty, J. and Hudson, S.E. GADGET: A toolkit for optimization-based approaches to interface and display generation. In *Proc. of UIST'03* (Vancouver, Nov. 2-5, 2003). ACM Press, New York, 2003, 125–134.
- [5] Gajos, K. and Weld, D.S. SUPPLE: Automatically Generating User Interfaces. In *Proc. of IUI'04* (Funchal, January 13-16, 2004). ACM Press, New York, 2004, 93–100.
- [6] Griffiths, T., Barclay, P.J., Paton, N.W., McKirdy, J., Kennedy, J.B., Gray, P.D., Cooper, R., Goble, C.A., and Pinheiro da Silva, P. Teallach: a model-based user interface development environment for object databases. *Interacting with Comp.* 14,1 (2001) 31–68.
- [7] Grolaux D., Van Roy P., and Vanderdonck J. FlexClock: A Plastic Clock Written in Oz with the QtK Toolkit. In *Proc. of TAMODIA'2002* (Bucharest, July 18-19, 2002). Academy of Economical Studies of Bucharest, Bucharest, 2002, 135–142.
- [8] Hopcroft, J.E. and Ullman, J.D. *Introduction to automata theory, languages, and computation*. Addison-Wesley, Reading, 1979.
- [9] Kray, C., Wasinger, R., and Kortuem, G. Concepts and issues in interfaces for multiple users and multiple devices. In *Proc. of M3UI'04*, 2004.
- [10] Limbourg, Q., Vanderdonck, J., Michotte, B., Bouillon, L., Lopez, V. UsiXML: a Language Supporting Multi-Path Development of User Interfaces. In *Proc. of EHCI-DSVIS'2004* (Hamburg, July 11-13, 2004). LNCS, Vol. 3425, Springer, Berlin, 2005, 200–220.
- [11] Metrowerks, Inc., PowerPlant for CodeWarrior. Austin, 1996. Accessible at <http://www.metrowerks.com/>
- [12] Mori, G., Paternò, F., and Santoro, C. Design and Development of Multidevice User Interfaces through Multiple Logical Descriptions. *IEEE Trans. on Soft. Eng.* 30,8 (August 2004), 507–520.
- [13] Mozart Consortium, The Mozart Programming System (Oz 3). Accessible at <http://www.mozart-oz.org/documentation>.
- [14] Puerta, A.R. A Model-Based Interface Development Environment. *IEEE Software* 14,4 (July/August 1997) 41–47.
- [15] Seffah, A. and Javahery, H. (eds.). *Multiple User Interfaces: Cross-Platform Applications and Context-Aware Interfaces*. John Wiley & Sons, Chichester, 2003.
- [16] Szekely, P.A., Sukaviriya, P.N., Castells, P., Muthukumarasamy, J., and Salcher, E. Declarative interface models for user interface construction tools: the MASTERMIND approach. In *Proc. of EHCI'95*. Chapman & Hall, London, 1996, 120–150.
- [17] Thevenin, D. and Coutaz, J. Plasticity of User Interfaces: A Framework and Research Agenda. In *Proc. of INTERACT'99* (Edinburgh, Aug. 30-Sept. 3, 1999). IOS Press, 1999, 110–117.
- [18] Vanderdonck, J. and Bodart, F. Encapsulating Knowledge for Intelligent Automatic Interaction Objects Selection. In *Proc. of INTERCHI'93*. ACM Press, NY, 1993, 424–429.
- [19] Vanderdonck J., Limbourg Q., Florins, M. Deriving the Navigational Structure of a User Interface. In *Proc. of INTERACT'2003* (Zurich, Sept. 1-5, 2003). IOS Press, 2003, 455–462.
- [20] Vanderdonck, J. A MDA-Compliant Environment for Developing User Interfaces of Information Systems. In *Proc. of CAISE'05* (Porto, June 13-17, 2005). Vol. 3520. Springer, 2005, 16–31.
- [21] Wiecha, Ch., Bennett, W., Boies, S.J., Gould, J.D., and Greene. ITS: A Tool for Rapidly Developing Interactive Applications. *ACM Trans. Inf. Syst.* 8,3 (1990) 204–236.