Chapter 13

ONE MODEL, MANY INTERFACES

Fabio Paternò, Carmen Santoro
ISTI, CNR Institute, Via G. Moruzzi 1
I-56010 Ghezzano, Pisa (Italy)
E-mail: {f.paterno, c.santoro}@cnuce.cnr.it – URL: http://giove.cnuce.cnr.it/~fabio/
Tel: +39-050-315 30 66 – Fax: +39-050-313 80 91

Abstract The wide variety of devices currently available, which is bound to increase in the coming years, poses a number of issues for the design cycle of interactive software applications. Model-based approaches can provide useful support in addressing this new challenge. In this paper we present and discuss a model-based method for the design of nomadic applications showing how the use of models can support their design. The aim is to enable each interaction device to support the appropriate tasks users expect to perform and designers to develop the various device-specific application modules in a consistent manner.

Keywords: Context, Model-based design, Multi-platform user interfaces.

1. INTRODUCTION

Recent years have seen the introduction of many types of computers and devices (e.g. cellular phones, PDA's, WebTV, etc.) and the availability of such a wide range of devices has become a fundamental challenge for designers of interactive software systems [8]. Users wish to be able to seamlessly access information and services regardless of the device they are using, even when the system or the environment changes dynamically. To this end, computer-based applications need to run on a wide spectrum of devices.

Designing applications that exploit new technology is often a difficult problem. For software developers this introduces the problem of constructing multiple versions of single applications and endowing these versions with the ability to dynamically respond to changes in context. Creating different versions of applications for different devices engenders extra devel

opment and expensive maintenance cost of cross-platform consistency, complicates the problems of configuration management and dilutes the resources available for usability engineering. Additionally, current development tools provide little support for creating applications that change dynamically in response to changes in their environment, or that have to share data amongst heterogeneous device types. Although many dimensions must be considered when designing context-dependent applications (actors, platforms, environments, system resources, etc.), in this paper we focus on how to support a design addressing change of platforms.

Nowadays companies need to better consider the thorny issues involved in designing applications supporting multiple platforms. Currently, they often address different contexts of use by building different applications, even if supporting similar tasks accessing similar data. Different versions of applications should differ in their presentation, rather than trying to address different input/output platforms by just resizing the elements of the user interface. Versions may also differ in the tasks they can realistically support – for example, the PDA permits reading reviews, the phone permits ordering books, and the desktop PC supports both. In addition the set of functionality supported can change dynamically.

Technological advances solve some of the problems of engineering applications for multiple devices: XML (eXtensible Markup Language) documents supported by XSL (eXtensible Stylesheet Language) stylesheets allow creating customised presentations for different devices or users. Wireless Markup Language (WML) permits to produce device-independent presentations for a range of small display devices. Wireless Internet gateways automatically translate HTML documents into WML documents (although they may produce unusable results if they rely on large displays). XSL (and related technologies) help with user interface presentation, but are limited by the fact that a different interaction design may be necessary when moving between radically different device types. While these solutions help with parts of the problem, they do not provide high-level guidance for guaranteeing quality across multiple versions of applications.

The goal of this paper is to present and discuss a method to support design and development of highly usable context-sensitive interactive software systems. To this end we discuss the state of art in addressing issues related to multi-platform applications. Next, we identify the main phases of the proposed method and also illustrate how the information provided by task models can be useful for the design of user interfaces. Then, we analyse more in depth the method and show its application to an example taken from the design of a museum application. Lastly, some concluding remarks are provided.

2. RELATED WORK

In a recent paper, discussing the future of user interface tools, Myers, Hudson, and Pausch [7] indicate that the wide platform variability *encourages a return to the study of some techniques for device-independent user interface specification, so that developers can describe the input and output needs of their applications, so that vendors can describe the input and output capabilities of their devices, and so that users can specify their preferences. Then, the system might choose appropriate interaction techniques taking all of these into account*. The basic idea is that instead of having separate applications for each device that exchange only basic data, there is some abstract description and then an environment that is able to suggest a design for a specific device that adapts to its features and possible contexts of use. This is also called user interface plasticity [14]. Methods for modelling work context [2] can provide useful information for this type of approach.

This problem is a novel challenge for model-based design and development of interactive applications. The potentialities of these approaches have been addressed in a limited manner. In the GUITARE Esprit project a user interface generator has been developed: it takes ConcurTaskTrees (CTT) task models [9] and produces user interfaces for ERP applications according to company guidelines. However, automatic generation is not a general solution because of many, varying factors that have to be taken into account within the design process. Semi-automatic support is more general and flexible: Mobi-D [11] is an example of a semi-automatic approach but it only supports design of traditional graphical desktop applications.

UIML [1] is an appliance-independent XML user interface language. While this language is ostensibly independent of the specific device and medium used for the presentation, it does not take into account the research work carried out in the last decade on model-based approaches for user interfaces: for example, the language provides no notion of task, it mainly aims to define an abstract structure. The W3C consortium has recently delivered the first version of a new standard (XForms) that presents a description of the architecture, concepts, processing model, and terminology underlying the next generation of Web forms, based on the separation between the purpose and the presentation of a form. If it shows the importance of separating conceptual design from concrete presentation, it also highlights the need for meaningful models to support such approaches.

More generally, the issue of applying model-based techniques to the development of UIs for mobile computers has been addressed at a conceptual and research level [3, 4], but there are still many issues that should be solved to identify systematic, general solutions that can be supported by automatic tools. Our approach aims to support design and development of nomadic ap

plications providing general solutions that can be tailored to specific cases, whereas current practise is still to develop *ad hoc* solutions with few concepts that can be reused in different contexts.

3. THE PROPOSED METHOD

The design of multi-platform applications can follow different approaches. It is possible to support the same type of tasks with different devices by changing the set of interaction and presentation techniques taking into account the resources available in the device considered. Another, more promising option is to consider different devices also with regard to the choice of the tasks to support (e.g. phones more suitable for quick access to limited information, desktop systems more suitable for browsing through large amounts of information). To complicate matters, even within the same class of devices there are different presentation models that need to be handled: for example, in WAP-enabled phones a number of micro-browsers tend to accept slightly different versions of WML, must interact with slightly different phones (for examples, phones with a different number of softkeys) and interpret the softkey interactions differently.

Our method tries to address such problems, and is composed of a number of steps that allows designers to start with an overall envisioned task model of a nomadic application and then derive concrete and effective user interfaces for multiple devices:

- *High-level task modelling of a multi-context application*. In this phase designers need to think about the logical activities that have to be supported and relationships among them. They develop a single model that addresses the various possible contexts of use and the various roles involved and also a domain model aiming to identify all the objects that have to be manipulated to perform tasks and the relationships among such objects.
- *Developing the system task model for the different platforms considered*. Here designers have to filter the task model according to the target platform. This involves creating task models in which the tasks that cannot be supported in a given platform are removed and the navigational tasks deemed necessary to interact with the considered platform are added. Thus, we obtain the system task model for the platform considered. Such models are specified using the ConcurTaskTrees notation. The CTTE (CTT Environment) tool (publicly available at http://giove.cnuce.cnr.it/ctte.html) supports editing and analysis of task models specified using this notation.

- *From system task model to abstract user interface*. Here the goal is to obtain an abstract description of the user interface composed of a set of abstract presentations that are identified with the support of the enabled task sets and structured by means of various operators. Then, still with the help of the task model, we identify the possible transitions among the user interface presentations considering the temporal relationships that the task model indicates.
- *User interface generation*. In this phase we have the generation of the user interface. This phase is completely platform-dependent and has to consider the specific properties of the target device. For example, if the considered device is a cellular phone, such information is not sufficient as we also need to know the type of micro-browser supported and the number and the types of soft-keys available.

In the following sections we better explain such steps while showing their application to a specific example.

We have defined XML versions of the language for task modelling (ConcurTaskTrees), enabled task sets and the language for modelling abstract interfaces and started development of the transformations among these representations.

4. MODEL-BASED DESIGN

Various models have been proposed to highlight important aspects in the design of user interfaces [5, 9, 12, 13, 15]. In our method we focus on models that can support development of user interfaces while preserving usability, in particular task models specifying the different activities that are supposed to be performed in an interactive system. Task models should be developed involving users so as to represent how they prefer to perform activities. By analysing the temporal relationships of a task model, it is possible to identify the sets of tasks that are enabled over the same period of time according to the constraints indicated in the model (*enabled task sets*). Thus, the interaction techniques supporting the tasks belonging to the same enable task set are logically candidate to be part of the same presentation though this criteria should not be interpreted too rigidly in order to avoid too modal user interfaces.

In addition, several criteria can be considered to derive concrete user interfaces from task models. Examples of such criteria are:

- The logical decomposition of tasks can be reflected in the presentation by explicitly grouping interaction techniques associated with tasks that share the same parent task.

- Sequential tasks can be supported in various modalities: for example with separate presentations for each task rendered at different time, or with all the interaction techniques corresponding to the sequential tasks rendered in the same presentation.
- The type of task, the type of objects manipulated and their cardinality are other useful elements. For example, for single choice among low cardinality values we can select a radio-button.
- Tasks that interrupt (or activate) other activities should be represented in the same manner (e.g.: buttons with specific graphical attributes) and located in the same part of the layout (e.g.: the right bottom part of the screen) for consistency.

5. THE EXAMPLE AND ITS TASK MODEL

The example considered is about the design of a nomadic application for accessing museum information. A possible scenario is a user who comes across the museum while surfing the web at home. S/he accesses the web site and starts to look at the information contained and afterwards s/he proposes to some friends a visit to the Marble Museum. As they seem to be a bit reluctant, s/he accesses the WAP server trying to convince them by showing some more concrete element of what the museum offers. Finally, they decide to have a visit, so they access again the WAP server to check its location and schedule. When they arrive at the museum, they receive a guide implemented in PDA that provides audio-visual support for their visit.

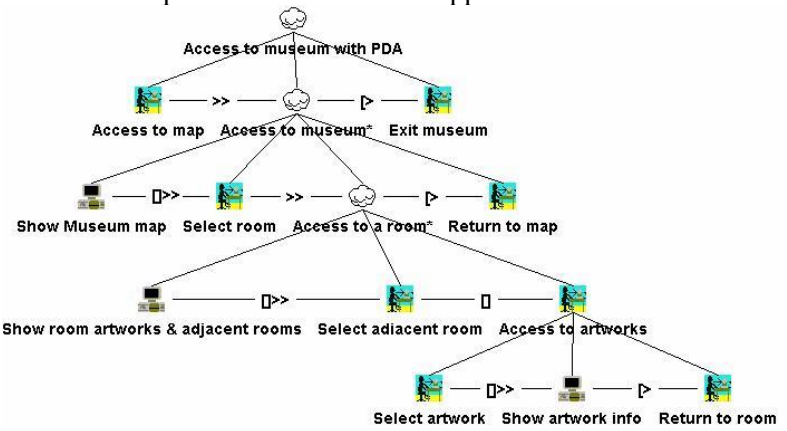

Figure 1. Part of the system task model for the PDA access.

For space reasons we provide only part of the system task model for the PDA (Fig. 1). Users can access the museum map (*Access to map*), and then (see the enabling CTT operator ">>"), after such map is displayed (*Show*

museum map), they can select a room (*Select room*). When they select a room, the presentation shows the different types of artwork in the room, so as to allow visitors to locate them. After the visitor selects a specific artwork (*Select artwork*), depending on the selection (see the enabling with information passing operator "[]>>") the PDA delivers information about the artwork (*Show artwork info*), with the possibility of returning to the previous room. Further actions will be made available to the user: either return to the global map (*Return to map*) or exit the museum (*Exit museum*), both appearing at the right of a disabling operator ("[>").

6. FROM THE TASK MODEL TO THE ABSTRACT USER INTERFACE

Starting with the task model of the system, we aim to identify the specification of the abstract user interface in terms of its static structure (the "*presentation*" part) and dynamic behaviour (the "*dialogue*" part): such abstract specification will be used to drive the implementation.

6.1 The Presentation Part

The first step is to calculate the Enabled Task Sets (ETSs) according to the system task model. The CTTE tool automatically performs the identification of these sets. For the considered example, the ETSs are:
ETS1: {Access to map}
ETS2: {Show Museum map, Exit museum}
ETS3: {Select room, Exit museum}
ETS4: {Show room artworks&adjacent rooms,Return to map, Exit museum}
ETS5: {Select adjacent room, Select artwork, Return to map, Exit museum}
ETS6: {Show artwork info, Return to room, Return to map, Exit museum}

6.1.1 Heuristics for Obtaining a Lower Number of Task Sets

Once ETSs have been defined, we need to specify some rules to reduce their number by merging two or more ETSs into new sets, called *Task Set* or TS (as the union of several ETSs is no longer an ETS). The reasons (and the aims) to do such step are various: first of all, reducing the initial number of ETSs which —as we previously noted— in some cases can be very high; secondly, keeping and highlighting in the same presentation significant information (as a data exchange is) even when the involved tasks belong to different ETSs; lastly, avoiding repeatedly considering groups of tasks which

all share a similar structure. Up to now, the heuristics that have been identified are the following:

- *H1*: If two (or more) ETSs differ for only one element, and those elements are at the same level connected with an enabling operator, they could be joined together with the Ordering operator.
- *H2*: If an ETS is composed of just one element, it should be joined with another ETS that shares some semantic feature.
- *H3*: If some ETSs share most elements, they could be unified. For example if the common elements all appear at the right of the disabling operator, they could be joined in only one TS.
- *H4*: If there is an exchange of information between two tasks, they can be put in the same TS in order to highlight such data transfer.

For example, ETS2 and ETS3 differ by only one element: applying H1 they could be unified into *TS1*={*Show museum map*, *Select room*, *Exit Museum*}. Also, ETS4 and ETS5 share most elements: H3 can be applied to obtain *TS2*={*Select adjacent room*, *Select artwork*, *Show Room Artworks & Adjacent Rooms*, *Return to map*, *Exit museum*}.

6.1.2 From TSs to Abstract Presentations

The set of TSs obtained is the initial input for building the abstract user interface specification which will be composed of interactors [10] (abstract interaction objects) associated with the basic tasks. Such interactors are high-level interaction objects that strongly depend on the particular corresponding task type. For simplicity now we focus only on a subset of the operators that have been identified taking into account criteria typically considered by user interface designers [6], namely:

- ***Grouping (G)****:* the idea is to group together two or more elements, so this operator should be applied when the involved tasks share some characteristics. A typical situation is when the tasks have the same parent (they are activities needed to perform a high level task).
- ***Ordering (O)*** operator: it is applied when some kind of order exists amongst elements. The more intuitive one is the temporal order.
- The ***Relation (R)*** operator should be applied when a relation exists between n elements y_i, i=1,…, n and one element x. Referring to the task model, a typical situation is when we have a leaf task t at the right side of a disabling operator: all the tasks that could be disabled by t (at whatever task tree level) are in relation with t.

Now we have to map each task of the task set into a suitable interactor and build a presentation structure that reflects the different relationships between such interactors by using the operators, whose applicability depends on the different temporal relationships specified in the task model. In order

to derive the presentation structure associated to the specific task set and deduce the operators that should be applied to them, we have to consider the part of the task model regarding the tasks belonging to a specific task set. In this process we have to consider that temporal relationships existing between tasks are inherited also by their subtasks. For example, the presentation structure obtained for TS2 is:

O (*Show Room Artworks & Adiacent Rooms*, **G** (*Select adjacent room*, *Select artwork*)) **R** *Return to map* **R** *Exit museum*

This is because the Ordering operator highlights the information transfer (specified by the []>> operator) between the *Show Room Artworks & Adiacent Rooms* and both *Select adjacent room* and *Select artwork* tasks (which are grouped together because of the choice "[]" operator). Each task is in turn in relation with *Return to map* and *Exit museum* because they both appear at the right of a disabling operator ([>). In the same way we can identify the abstract presentation associated to TS1 which is: **O**(*Show museum map*, *Select room*) **R** *Exit Museum*

In addition, we have to map each task into a suitable interactor, by considering relevant dimensions specified in the task model. For example with regard to a *selection* task (as *Select adjacent room* task) we identify the *type of selection* (single/multiple), the *type* of manipulated objects (boolean/quantitative/text, etc) and the *cardinality* (low/medium/high) of the dataset from which the selection should be performed, as relevant dimensions. Once such attributes have been identified, we define some rules to indicate in which case a widget is more appropriate than another one depending on the different values that each attribute assumes and on the specific platform and/or device considered. For example, as the *Select adjacent room* task is single selection task managing spatial information and the set of manipulated objects has a low cardinality, the interaction technique to be provided must allow a choice of elements by providing an interactive map.

6.1.3 The Dialogue Part

Once the *static* arrangement of the abstract user interface is identified, we have to specify its *dynamic* behaviour. To this aim, an important role is played by the so-called *transition tasks*. For each task set T, we define *transition tasks(T)* the tasks whose execution makes the abstract user interface pass from the current task set T into another task set T'. For each task set T, a set of rules (*transition_task*, *next_TS*) should be provided whose meaning is: when *transition_task* is executed, the abstract user interface passes from T to *next_TS*. For example, TS1 has two transition tasks: *Select Room* and the task *Exit Museum*. To express that via the transition task *Select room* the abstract interface passes from TS1 to TS2 we can use the following rule:

<task_set **TS1** */task_set ><behaviour><rule>*
<transition_task **Select room** */transition_task> <next_TS* **TS2** */next_TS>*
</rule></behavior>

6.2 From the Abstract User Interface to its Implementation

Once the elements of the abstract user interface have been identified, every interactor has to be mapped into interaction techniques supported by the particular device configuration considered (operating system, toolkit, etc.), and also the abstract operators have to be appropriately implemented by highlighting their logical meaning: a typical example is the set of techniques for conveying grouping relationships in visual interfaces by using presentation patterns like proximity, similarity and continuity [6]. A possible implementation for the presentation corresponding to TS2 is shown in Fig. 2, supposed that the current room is about Roman Archaeology.

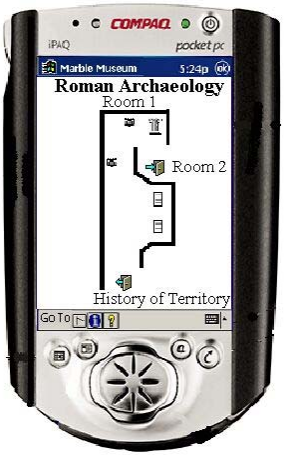

Figure 2. One presentation of the PDA user interface.

With the suggested scenario it is possible to show how the same task can be supported differently depending on the platform used. Consider accessing the artworks list grouped by the material used: in the desktop application this task could be supported by buttons whose colour is similar to the material used and a long textual introduction to the section is presented too (right part of Fig. 3). However, this solution cannot be implemented in the WAP interface (left part of Fig. 3) as it does not support colours and there is no room for buttons and long texts, so a list of links is used in this platform.

Fig. 4 shows another example of different presentation of some visual information between a desktop system and a WAP phone: as you can see from the bottom part of the picture, while in the desktop system it is possible to have details of the work (title, type, description, author, material, and date of

creation), in the WAP interface we can have only low resolution images allowing to have just a rough idea of what the work of art is, together with the indication of the title and the related museum section.

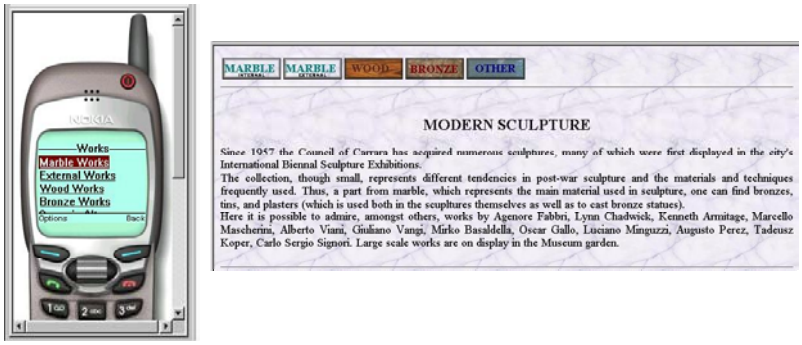

Figure 3. An example of different support for the selection task.

The top part of Fig. 4 shows how in the CTTE tool it is possible to specify for each task that each platform (PDA, desktop system, cellphone or other platforms) can support different sets of objects manipulated during the task performance. For example, for the "*Show artwork info*" task considered in the picture, the designer has specified that the title is available on all the platforms considered (all the checkboxes labelled "*PDA*", "*Desktop*", and "*Cellphone*" have been selected), whereas the description is available only for the desktop system.

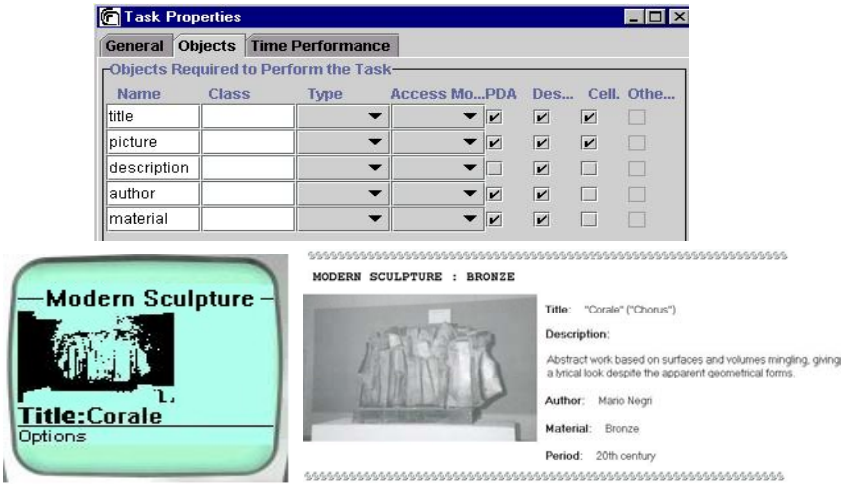

Figure 4. Example of different presentations of a work of art.

This information has been used to generate different user interfaces for the cellular phone and the desktop system: Fig. 4 shows that the title is available on both the platforms whereas the description is available only for the desktop system.

7. CONCLUSIONS

In this paper we have described a method for designing multi-device, nomadic applications starting from their task model and using a number of abstractions in the design cycle to obtain final applications able to effectively support the users' activities through various devices accessed in different contexts. The application of the method helps maintain a high-level of consistency across the multiple user interfaces developed. As the tool support in the user interface generation phase is still limited, future work is planned to extend it and create a complete semi-automatic environment implementing the presented method.

REFERENCES

[1] Abrams, M., Phanouriou, C., Batongbacal, A., Williams, S., and Shuster, J., *UIML: An Appliance-Independent XML User Interface Language*, in Proc. of WWW'8 (Toronto, May 1999).

[2] Beyer, H. and Holtzblatt, K., *Contextual Design: Defining Customer Centred Systems*, Morgan Kaufman, San Francisco, 1998.

[3] Einsenstein, J., Vanderdonckt, J., and Puerta, A., *Applying Model-Based Techniques to the Development of UIs for Mobile Computers*, in Proceedings of ACM Conference on Intelligent User Interfaces IUI'2001, ACM Press, New York, 2001, pp. 69-76.

[4] Calvary, G., Coutaz, J., and Thevenin, D., *A Unifying Reference Framework for the Development of Plastic User Interfaces*, Proc. of EHCI'2001, Springer-Verlag, 2001.

[5] Johnson, P., Wilson, S., Markopoulos, P., and Pycok, J., *ADEPT - Advanced Design Environment for Prototyping with Task Models*, in Proceedings of InterCHI'93 (Amsterdam, 24-29 April 1993), ACM Press, New York, 1993, pp. 56-57.

[6] Mullet, K. and Sano, D., *Designing Visual Interfaces*, Prentice Hall, 1995.

[7] Myers, B., Hudson, S., and Pausch, R., *Past, Present, Future of User Interface Tools*, ACM Trans. on Computer-Human Interaction, Vol. 7, No. 1, March 2000, pp. 3-28.

[8] Olsen, D., *Interacting in Chaos*, Keynote address, IUI'98, San Francisco, 1998.

[9] Paternò, F., *Model-based Design and Evaluation of Interactive Applications*, Springer-Verlag, Berlin, November 1999.

[10] Paternò, F. and Leonardi, A., *A Semantics-based Approach to the Design and Implementation of Interaction Objects*, Computer Graphics Forum, Blackwell Publisher, Vol.13, No. 3, 1994, pp. 195-204.

[11] Puerta, A.R., *A Model-Based Interface Development Environment*, IEEE Software, July/August 1997, pp. 40-47.

[12] Sukaviriya, P.N., Foley, J.D., and Griffith, T., *A Second Generation User Interface Design Environment: The Model and the Runtime Architecture*, in Proc. of InterCHI'93 (Amsterdam, 24-29 April 1993), ACM Press, New York, pp. 375-382.

[13] Szekely, P., Luo, P., and Neches, R., *Beyond interface builders: Model-based interface tools*, in Proceedings of InterCHI'93, ACM Press, New York, 1993, pp. 383-390.

[14] Thevenin, D. and Coutaz, J., *Plasticity of User Interfaces: Framework and Research Agenda*, in A. Sasse and C. Johnson (eds.), Proc. of Interact'99 (Edinburgh, August 1999), IOS Press Publ., 1999, pp. 110-117.

[15] Vanderdonckt, J. and Bodart, F., *Encapsulating Knowledge For Intelligent Automatic Interaction Objects Selection*, in Proc. of InterCHI'93 (Amsterdam, 24-29 April 1993), ACM Press, New York, 1993, pp. 424-429.