Chapter 1

XXL: A VISUAL+TEXTUAL ENVIRONMENT FOR BUILDING GRAPHICAL USER INTERFACES

Eric Lecolinet
Ecole Nationale Supérieure des Télécommunications & CNRS URA 820
Dept. INFRES, 46 rue Barrault, 75013 Paris, France
elc@enst.fr - http://www.enst.fr/-elc

Abstract This paper presents XXL, a visual+textual environment for the automated building of graphical user interfaces. This system uses a declarative language which is a subset of the C language and can either be interpreted or compiled. It includes an interactive builder that can both handle graphical and non-graphical objects. This tool makes it possible to create highly customized interfaces by visual programming or by "sketching" early interface ideas that are automatically interpreted by the system to produce executable GUI objects. This builder is based on the concept of textual+visual equivalence and is able to re-edit and modify any legible source code, not only the code it itself produced. This environment is thus a truly open system that can cooperate with higher-level tools.

Keywords: User interface design, interface builders, visual / textual equivalence, sketching, model-based interface development, specification languages.

1. INTRODUCTION

As noted in [9], the paradigm of model-based interface development has attracted a high degree of interest. However, despite its potential, this technology has not yet reached the marketplace and still remains limited to laboratory tools. On the contrary, "classical" interactive interface builders (GUIBs) are now quite widespread tools that are currently used by programmers and interface designers in spite of their well-known limitations. A possible reason for explaining this situation is that:

- Interactive builders make it possible to create highly customized GUIs that make use of quite a large set of GUI primitive elements (i.e. the "widgets" or "controls" that are provided by the underlying GUI toolkit.)

- Interactive builders are rather easy and intuitive to use because they are based on a direct manipulation paradigm (generally referred to as "Visual Programming".)

This paper does not present a high-level model-based approach but a textual+visual development tool, called **XXL**, that could be seen as the "missing link" between interactive GUI builders and high-level model-based environments. The core idea of this system is to unify textual programming (by using a specification language) and visual programming (by using an interactive GUI builder.) User interfaces can thus either be made through textual or visual programming or a free combination of both modes. Thus, as other interactive GUIBs, the XXL builder makes it possible to create highly customized interfaces by visual programming. The resulting source code is either interpretable or compilable by a standard C or C++ compiler. Most of this code follows a declarative style.

But unlike classical tools, the XXL system is also able to deal with preexisting source code. This code must just follow the (C language compatible) XXL syntax and can be produced by any means. It can for instance be directly written by a programmer or it can result from automatic code generation from another tool. By opposition with other systems, the XXL builder is thus able to re-edit and modify any legible source code, not only the code it itself produced. This property has important consequences:

- It avoids the usual strong separation between the encoding of the presentation and the interactive part of the GUI. Most interactive builders deal with both aspects in a completely different way: they make it easy to set presentation through visual programming, while they generally provide little help for specifying the GUI interaction (which usually require writing C or C++ source code). The XXL system provides several ways to integrate both aspects in a unified framework, as will be explained in the next sections.

- It allows for a truly iterative development scheme. It is for instance possible to create a GUI prototype by using the interactive builder, then to encode very specific behaviors textually in C/C++ language (for instance for dealing with variable amounts of data or data that changes dynamically at run time), then to re-edit interactively the graphical part of these textual specifications with the XXL builder in order to refine the GUI presentation, and so on. So, the interactive builder can be used at any stage of the development process because the generated source code can

be freely modified without preventing further interactive modifications by using this visual programming tool.

- Because it is based on a generic and declarative textual specification language, the XXL environment can be seen as a truly open system that can cooperate with higher-level tools. This point is especially interesting and effective because the system is based on the concept of visual and textual equivalence. One could for instance consider the following scheme:
  - standard XXL specifications could be produced by a high-level model-based system,
  - these specifications could be interactively modified by using the XXL builder in order to refine the GUI presentation and make them perfectly fit the user needs,
  - the resulting code could then either be feed back to the model-based system, or be enriched textually by adding specific dynamical behaviors, etc.

  These three phases could take place at any time during the development cycle, and be mixed in any order until the final GUI is obtained.

The next section describes the underlying concepts of the XXL system and the properties of the interactive builder. Section 3 presents an extension of the system that makes it possible to create GUIs from "rough drafts". This module can be used at the first stages of the prototyping phase in order to produce an executable GUI from a "preliminary drawing" that helps designers conceive how the interface could look like. At last we will compare XXL with related work and we will conclude.

2. THE XXL MODEL

The XXL system is based on a generic Object Oriented Model that tends to integrate and make work together several programming modes that are usually dealt with separately. Besides its own "inner function", each XXL object must implement data and methods that makes it possible:

- To deal interactively with a graphical representation of this object in the XXL builder,
- To produce the corresponding textual representation of this object in XXL/C language source code,
- To retrieve and to decode its corresponding textual representation in a XXL/C source file.

This architecture implies several interesting characteristics that are detailed in the next subsections.

2.1 THREE VIEW EDITION

This model makes it possible to associate a visual representation with *any* object, including non-graphical objects. This point is especially important and is a major difference with classical point-and-click direct manipulation interface builders. These tools are generally based on a WYSISYG paradigm and can only display the "widgets" or "controls" that compose the GUI. But they are generally unable to represent (and to let the user interact with) the non-graphical objects that may be part of the UI.

This point can be seen as a consequence of the "concreteness" of WYSISYG representations. On the one hand, moving and manipulating widgets directly by using the mouse pointer is quite convenient and intuitive, especially for novice programmers. But this concreteness implies a corollary drawback: the inability to represent and modify "immaterial" behaviors, abstract specifications or GUI parameterizations in a simple and coherent way.

The XXL model provides an integrated way to represent and interact with any kind of object. The interactive builder provides three views of the interfaces that are being developed: the *graph view* (Fig. 1), the *text view* (Fig. 2) and the *widget view*. The text view shows the corresponding descriptions in the source code. The graph view is an iconic representation that is equivalent to the text view. These two views constitute the dual (textual and visual) "abstract" specification of the UI while the widget view can be seen as the "result" of this specification. These views are *linked together* and are *incrementally updated* whenever the UI is modified interactively by using the builder. There is a one to one correspondence between the various representations of a same object. For instance, control-clicking on an object representation in any view will highlight its other available representations in the other views.

This three-view model makes it possible not only to show and control the graphical aspects but can also represent the "hidden abstract parts" of the UI. This can be seen as an attempt to let the designer see "what is behind the curtain" and provide the kind of "indirect manipulation" defined in [7] as the ability to "directly manipulate an abstraction that controls the behavior or appearance of the actual objects".

2.2 TEXTUAL + VISUAL EQUIVALENCE

Textual + Visual Equivalence is another consequence of the system architecture. Textual specifications of XXL interfaces can either be interpreted or be compiled by using a standard C or C++ compiler. In both cases, the interactive builder will be able to establish the reverse correspondence between the dynamical objects that it manipulates and their textual descriptions in the original source files. The builder is thus able to deal with preexisting source code and let the user modify the corresponding objects dynamically (even if this source

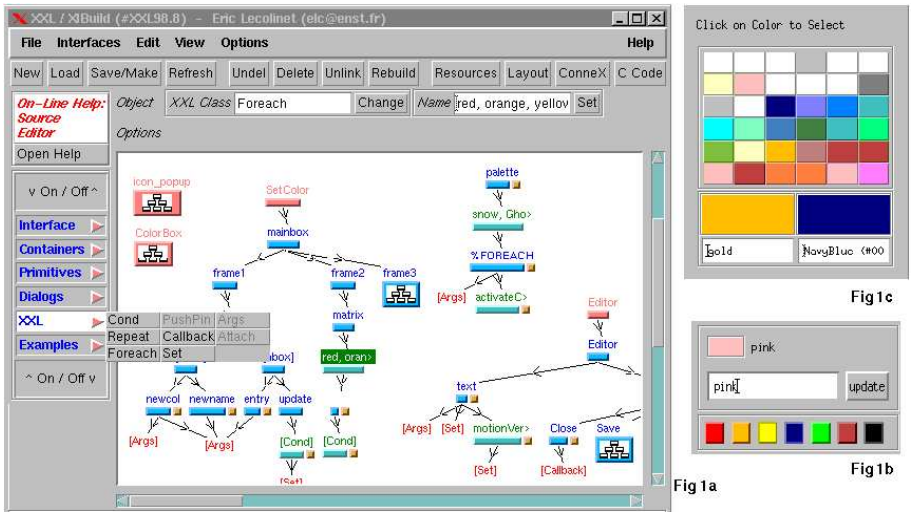

Figure 1.1 The graph view and two GUI examples.

code is C compiled). This source code must just follow the XXL syntax. It can be produced by any means and does not have to be necessarily generated by the XXL builder.

Moreover, XXL objects can dynamically modify their own external representation at run-time (i.e. their textual specification in the XXL/C source code). As a consequence, the original source code is modified in an incremental way by the builder and these modifications can even be made *while the application is running*. This property results from the XXL OO model as all objects are intrinsequely able to control their own representations when created, deleted or modified.

It is interesting to notice the XXL/C source code is post-interpreted from the internal representation of the objects that have been previously created. We call this "reverse-interpretation" as the parsing of the source code is guided by the run-time process (which actually results from the compilation of the same source code). In other words, XXL can be seen as a (C compatible) compilable language that is able to post-interpret its own source code dynamically at run-time in order to establish the reverse correspondence between the internal (i.e. binary) and external (i.e. textual+visual) representations of the produced objects.

2.3 GRAPHICAL, GENERIC AND IMMATERIAL OBJECTS

The current version of the system includes four main categories of objects:

- *Graphical objects* that have a physical representation in the resulting GUI (that is to say the “widget view’).

- *Control objects* that specify control statements such as repetitions, conditions or embedded pieces of code such as call-back functions.

- *Structuring objects* whose aim is to decompose the interface into homogeneous and reusable components.

- *Property objects* that parameterize the appearance and behavior of the actual GUI.

All objects are represented in the text and in the graph views. Programmers can not only interact with graphical components but also with the “immaterial” objects that enforce more abstract specifications. Objects have different representations in the graph view, depending on their actual type. The graph view is not a mere “widget tree” but a direct oriented graph (DAG) of various specification objects.

The *graphical objects* encapsulate the graphical widgets of an underlying toolkit (the current implementation of the XXL system being based on the X-Window / Motif 1.1 toolkit.) These objects handle three different representations (in the text, the graph and the widget views). The widget view is always active and can also be handled with in a direct manipulation style.

A graphical object does not necessarily correspond to a single specific widget of the underlying toolkit. The system also provides contextual “generic” objects that make it possible to handle actual graphical components with a higher level of abstraction. Such objects are dynamically instantiated into different actual widgets according to their structural and functional context. Moreover, object classes can be changed interactively when using the XXL builder. These changes are then recursively propagated to the children of the modified objects. This conception scheme makes it possible to hide many low-level details to the designer and remodel GUIs after their initial creation in quite an efficient way. For instance, a box containing a set of buttons can immediately be changed into a radio box, a dialog box, a menu bar or a menu by only changing the type of the container object. The actual corresponding widgets (and child widgets) are automatically changed in order to match the generic specification.

Implicit behaviors are automatically added when combining certain objects. A menu (or a dialog box) can for instance be specified as a button child. This menu (or this dialog) will then be automatically popped up when its button parent is pressed (or clicked in second case.) Moreover, the type of the actual menu widget will also depend on context: this menu will be a “pull-down” menu if its button parent is part of a menu bar and a contextual “pop-up”menu in other cases. Thus, presentation and interaction are often implicitly deduced

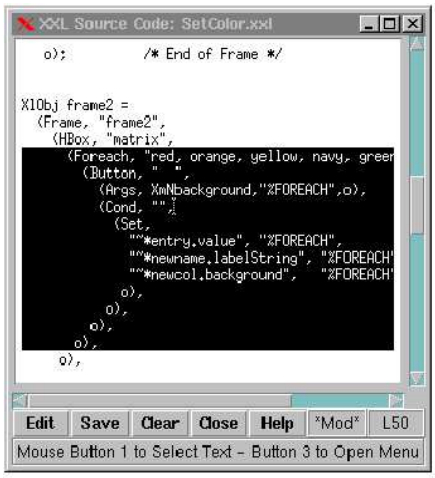

Fig 2a

```
X10bj open_dialog =
  (FileDialog, "open_dialog",
    (Args,"dialogTitle","Open Image",o),
    (Callback,"",OpenProc,NULL,o),
   o);

X10bj file_menu =     //each button opens a menu
  (Menu "file_menu",
    (Button, "Open", open_dialog,o),
    (Button, "Save", save_dialog,o),
   ....etc .....
   o);

X10bj menubar =
  (MenuBar, "menubar",
    (Button, "File", file_menu,o),
    (Button, "View", view_menu,o),
   ....etc .....
   o);

(HBox,"",
  (Label,"newcol",o), (Label,"newname",o),
o),
(HBox,"",
  (TextField,"entry",o),
  (Button,"update",
    (Cond,"",
      (Set,
        "~*newname.labelString","{~*entry.value}
        "~*newcol.background","{~*entry.value}",
       o,)o),
```

Fig 2b

Figure 1.2 Text view and textual specifications.

from context. Error checking is performed in real-time in order to detect possible incoherences and to warn the user in an appropriate way.

This features tries to solve a classical drawback that interface builders are usually blamed for: they require taking decisions that fix the presentation too early in the conception process [13]. The XXL builder allows for "second thoughts" by letting programmers modify their initial decisions and deeply change the structure, behavior or presentation of the UI at any stage of the iterative design process.

2.4 THE XXL SPECIFICATION LANGUAGE

The XXL system is based on an underlying specification language which is designed to be compact and reasonably easy to understand. The main interest of this language is that it is not "yet another programming language": in spite of its very specific form, it is actually a subset of the ANSI C language. This means that XXL descriptions can be freely included into C or C++ functions or other constructs and can be compiled as any other C statement. XXL interfaces can thus make use of the whole power of a standard programming language. This point is especially important when designing complex interfaces that deal with variable amounts of data or that evolve dynamically at run time. To paraphrase a quotation from [12], this means that we can both benefit from the intuitiveness - but (relatively) low expressivity - of interface builders and the low intuitiveness - but high level of expressivity - of standard programming languages.

Various behaviors can be specified in a declarative way thanks to a feature called Conditional Evaluation. This mechanism makes it possible to reevaluate a subpart of an XXL specification when a certain condition is satisfied (for instance when a certain event occurs on a graphical object on when an active value is changed). It is thus possible to specify object creation, modification or deletion in a declarative style. Fig. 1b illustrates a basic example of this mechanism (the corresponding code is at the bottom of Fig. 2b). This interface changes the color and the label of the two top widgets by reading a string that is entered by the user in a text field. When the user clicks on the update button, the sub-expression that is included in the *Cond* statement is reevaluated. This sub-expression specifies an assignment that gets the string value that was entered in the entry widget, converts it to the appropriate types and changes the string label and the background color of the appropriate objects. This mechanism can be seen as a way to specify call-back functions in a declarative way.

More sophisticated behaviors or constructs can also be specified. For instance, the *Foreach* abstract object makes it possible to iterate an XXL sub-expression. The simple color palette shown on Fig. 1c is created by using this feature: there is only one color button specification (Fig. 2a) that is iterated for several color names. Several actual widgets are thus generated but they all correspond to a single XXL graphical object. The behavior of these objects is also specified in a declarative way by means of a *Cond* statement which is itself included in the *Foreach* statement.

XXL specifications can also constitute interface models that can be instantiated several times. For instance, the five icons that are included in the interface shown in Fig. 3a result from five successive realizations of the same XXL sub-interface This example also shows the use of behavior objects that implement dynamic manipulations techniques. The icon interface specification (Fig. 3b) includes a *MoveHandle* object that automatically makes the icon instance movable by grabbing it interactively with the mouse. These icon instances are linked together with *DLinks* graphical objects. These links (that are materialized by arrows on the screen) will automatically follow the objects they are related to (that is to say the actual realizations of the icon specification) when these objects are moved.

As said before, XXL specifications can be included in C or C++ programs and be compiled "as it". They can also be interpreted by using a special-purpose Unix shell or by loading them dynamically from a C program. This feature can be used for testing new interfaces at the beginning of the prototyping phase. It also makes it possible to *exchange* XXL Interfaces dynamically between separate (and possibly remote) programs: XXL specifications are then sent through the network by using sockets and are interpreted at run-time by the receiving program. This mechanism can also be used for modifying objects or calling functions that reside in remote programs.

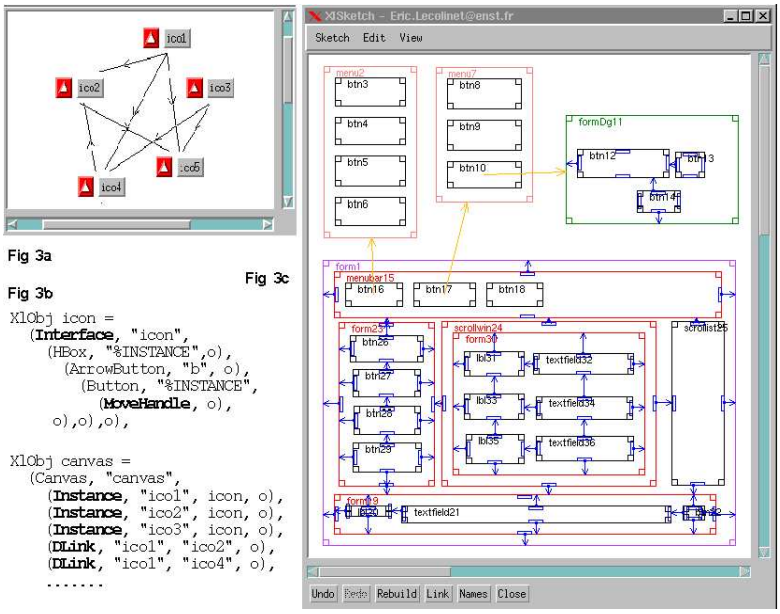

Figure 1.3 Interface and instances (a, b); The sketch view (c)

3. SKETCH DRAWING

This section presents an extension of the system that makes it possible to create GUIs from "rough sketches". This module provides a separate interactive way of designing interfaces at the early stages of their conception. It makes it possible to create an executable GUI from a "preliminary drawing" that helps designers conceive how the interface could look like. The idea is to let them draw early interface ideas in the same way as they would do on a piece of paper. But this electronic sketch will also produce a fully operational interface.

The Sketch View (Fig. 3c) lets designers conceive a first draft of the GUI by drawing a sketch on the screen. This drawing is dynamically interpreted by the system in order to produce an executable GUI. Actual XXL objects are implicitly "deduced" from the sketch and the resulting widget view is produced in real time so that designers can immediately see the result of their drawing and correct it iteratively (the system providing full undo capabilities). The graph view and the text view are made available to the designer once the sketching stage is over. Visual and/or textual edition can then be performed to refine the GUI, add call-back functions, etc.

These various views are mostly used at different steps of the development process. The sketch view let designers focus attention on the global layout of the GUI without having to take care of implementation details. The graph view

makes it possible to refine the presentation and to deal with immaterial objects that represent abstractions, while the text view is mostly used for dealing with implementation details or with the dynamical management of user interfaces (when GUIs must handle variable amounts of data or data that changes at run time). So, the XXL system tries to integrate high-level design as well as low-level implementation details into a unified framework.

3.1 SKETCH INTERPRETATION

Graphical objects are created by drawing rectangles at a certain location in the sketch. The first rectangle drawn in the sketching area is implicitly considered as a "main box" (that will not necessarily be the actual main window of the final application but can be included into another object at a later stage). Then, an included horizontal rectangle, located at the top of this main box will automatically be seen as a menu bar by the system (Fig. 3c). Drawing enclosed rectangles inside this menu bar will generate menu bar buttons. Menus (and dialog boxes) are created by drawing vertical (or horizontal) rectangles outside the main box. Menus and dialogs are attached to the button that will open them by drawing a link between these components.

Other rectangles (drawn in the main box, the menus or the dialog boxes) will be first interpreted as button objects. Buttons are then automatically transformed into intermediate container boxes if another rectangle (i.e. a button) is drawn inside them. Object type can also be explicitly set by the designer (for instance for transforming a button into a text area or whatever.)

The system proposes default rules for lay out management. Objects are automatically aligned but this default layout can be changed interactively. Graphical constraints are automatically computed by the system. These constraints are materialized by arrows on the drawing (fig 3c). Constraints can be set in a direct manipulation style by moving objects with the mouse or by attaching or detaching the corresponding arrows.

This way of designing GUIs favors the use of spatial topological constraints instead of fixing absolute x, y coordinates by moving and resizing widgets directly with the mouse. This leads to a more flexible representation that can evolve dynamically at run-time when the final user resizes the windows or customizes the application (for instance by specifying larger fonts). The use of such constraints is rather easy and natural here because they are either deduced from the drawing, or explicitly drawn in a simple way. Thus, it is interesting to notice that the drawing performed by the user is not a WYSISYG but a logical representation of the GUI. The actual GUI will not exactly look exactly the same as the drawing but will follow the logical constraints specified by the designer. It is up to the graphical system to adjust and lay out the corresponding widgets in an appropriate way.

The sketching module is based on a set of contextual rules that implicitely transform the user drawing into structural or topological constraints. These rules are encoded in an object oriented style. Drawing a new component produces a new sketching object that is managed by its own container. The combination of the rules defined in both objects will control the graphical aspect of these two sketching objects and will also implicitly produce XXL objects of an appropriate type (or modify existing objects in an appropriate way.)

4. RELATED WORK

The XXL system is related to the three following domains: interface builders, model-based interface development and sketch drawing. Similarly to standard interface builders (a comprehensive survey can be found in [8]), the XXL builder makes it possible to create highly customized interfaces by visual programming. However, the underlying model is quite different: the system can handle generic and immaterial objects, it uses a declarative language (which is a subset of the C language) and it is based on a textual+visual equivalence paradigm.

Promising approaches have been proposed in the field of model-based interface development [13], [12], [2], [14], [11], [9]. However XXL is not a high-level model-based approach but a visual+textual environment for the automated building of UIs. In that sense, it could be compared to certain aspects of other systems that can generate user interfaces from partial models such as Janus [1], or Mobi-D [10], that includes an interactive tool.

Interface sketching is a rather new approach and there are very few systems that implement this idea. For instance, the SILK system [3] provides an interactive tool that allows designers to quickly sketch an interface by using an electronic pad and stylus. By opposition, the XXL system uses a “faked metaphor” (designers draw sketches as they would do in “reality”, by not in the same “material” way) in order to prevent users from pattern recognition errors to ease interaction with ordinary pointing devices, and thus, to avoid the use of specific hardware.

5. CONCLUSION AND FUTURE WORK

This paper presents an hybrid approach which tries to mix several aspects of interactive GUI builders, model-based systems and interactive sketching. This new system lets designers produce generic interface specifications in an interactive way, either by textual or visual programming or by constrained sketching. The conception process is fully iterative and is consistent from the very early stages of design to the realization of the final application. The full system provides four different views of the GUI that correspond to various

stages of the conception process (or to the various level of expertise of the designers involved in this task).

The system has been fully implemented and relies on the X-Window system and the Motif toolkit. The XXL builder has been used for realizing various tools and students' projects at our institute. The XXL environment has also been used for creating and refining the interactive builder itself. The system is freely available at URL: `http://www.enst.fr/~elc`.

We are now developing a new GUI toolkit called **Ubit** that is based on a "brick construction game" metaphor [6]. Such a model is especially well suited for visual programming tools and should offer new perspectives when combined with the XXL builder.

References

[1] Balzert H., Hofmann F., Kruschinski V., Niemann C., *The JANUS Application Development Environment-Generating More than the User Interface*. In Jean Vanderdonckt, Ed., CADUI'96, Presses Universitaires de Namur, pp. 183-206, 1996.

[2] Bodart F., Hennebert A-M., Leheureux J-M., Provot I., Sacre B., Vanderdonckt J., *Towards a Systematic Building of Software Architecture: The Trident Methodological Guide*. Workshop on Design, Specification, Verification of Interactive Systems, pp. 237-253, 1995.

[3] Landay J., Myers B., *Interactive Sketching for the Early Stages of User Interface Design*. Proc. of the CHI Conference, 1995.

[4] Lecolinet E., *XXL: A Dual Approach for Building User Interfaces*. Proc. ACM UIST, pp. 99-108, Seattle, USA, Nov. 1996.

[5] Lecolinet E., *Designing GUIs by Sketch Drawing and Visual Programming*. Proc. Int. Conf. on Advanced Visual Interfaces (AVI). ACM Press, pp. 274-276, 1998.

[6] Lecolinet E., *A Brick Construction Game Model for Creating Graphical User Interfaces: The Ubit Toolkit* Proc. INTERACT'99.

[7] Morse A., Reynolds G., *Overcoming Current Growth Limits in UI Development*. Communications of the ACM, Vol.3 6, No. 4, April 1993.

[8] Myers B.A., *User Interface Software Tools*. ACM Trans. on Computer-Human Interaction, Vol. 2, No. 1, pp. 64-103, 1995.

[9] Puerta A., *The MECANO Project: Comprehensive and Integrated Support for Model-Based Interface Developement*. In Jean Vanderdonckt, Ed., CADUI'96, Presses Universitaires de Namur, 1996.

[10] Puerta A., Cheng E., Ou T., Min J., *MOBILE: User-Centered Interface Building*. Proc. of the CHI Conference, 1999.

[11] Schlungbaum E., Elwert T., *Automatic User Interface Generation from Declarative Models*. In Jean Vanderdonckt, Ed., CADUI'96, Presses Univ. de Namur, pp. 3-17, 1996.

[12] Szekely P., Luo P., Neches R., *Beyond Interface Builders: Model-Based Interface Tools*. Proc. INTERCHI'93, pp. 383-390, 1993.

[13] Wiecha C., Bennett W., Boies S., Gould J., Greene S., *ITS: A Tool for Rapidly Developing Interactive Applications*. ACM Trans. on Information Systems, Vol 8, No. 3, July 1990.

[14] Wilson S., Johnson P., *Bridging the Generation Gap: from Work Tasks to User Interface Designs*. In Jean Vanderdonckt, Ed., CADUI'96, Presses Univ. de Namur, pp. 77-94, 1996.