A Design Environment for Graphical User Interfaces

Jürgen Herczeg

Research Group DRUID
Department of Computer Science, University of Stuttgart
Breitwiesenstr. 20-22, D-7000 Stuttgart 80, Federal Republic of Germany
Tel: +49 711 7816 364
Fax: +49 711 780 10 45
E-Mail: herczeg@informatik.uni-stuttgart.de

Today, building user interfaces for sophisticated applications may be characterized as a design activity that needs powerful programming tools. Many of the existing tools are either too low-level or too restricted. We will describe a layered architecture of object-oriented user interface design tools, comprising user interface toolkits as well as visual programming tools for building, inspecting, and modifying graphical user interfaces interactively. They correspond to different abstraction levels on which user interfaces may be constructed and modified by user interface designers or even end users. We will further argue that these tools are even more effective when they are integrated within a visual programming environment together with general browsing and tracing components.

Keywords: User Interface Design, Graphical User Interfaces, User Interface Toolkits, User Interface Builders, Construction Kits, Design Environments, Object-Oriented Programming, Visual Programming

1 Introduction

With the availability of window systems like the X Window System [ScheiflerGettys 86] on different computer platforms, the use of graphical user interfaces in different application domains is permanently increasing. These systems basically provide a communication protocol for window server and client programs, as for instance the X protocol [Scheifler 89], and low-level programming interfaces, such as XLIB [Nye 88], which are open to very different interface styles. Because of their generality these programming interfaces are not appropriate for direct application; they rather form the basis for higher-level tools. Especially for building graphical user interfaces, object-oriented programming techniques proved to be very useful [Barth 86]. *User interface toolkits* are object-oriented programming interfaces that provide a higher abstraction level. They may be basically characterized as a library of common user interface elements like buttons, menus, or scrollbars, also called *widgets*, which are used as building blocks for graphical user interfaces. Over the last years, numerous toolkits emerged, as for instance the X Toolkit and the various widget sets built on top of it [NyeO'Reilly 90]. Mostly, these toolkits are difficult to use and do not provide appropriate means for defining new building blocks, let alone modifying predefined building blocks interactively. On the contrary, they often give up generality in favor of restricting appearance and functionality to a particular interface style, a so-called "look and feel", e.g., Open Look [SUN 90] or OSF/Motif [OSF 90].

Although object-oriented approaches employed by many high-level toolkits have achieved substantial improvement, constructing user interfaces with toolkits still needs high programming skills. Therefore, building graphical user interfaces for sophisticated applications definitely needs more powerful programming tools. *User Interface Management Systems* (UIMS) denote tools that support the development of user interfaces separately from the underlying applications [Pfaff 85]. Many examples of UIMS are based on the formal description of user interfaces. Building user interfaces employing graphical visualization and direct manipulation techniques, however, may be rather characterized as a design activity than giving formal descriptions or just program coding. For these kinds of tasks, the idea of *construction kits* and *design environments* raised special interest over the last years [FischerLemke 88]. Especially in the domain of user interface construction and design, interactive programming tools using visual programming techniques have been successfully employed [Myers 90, Myers et al. 90]. So-called *user interface builders* can be very helpful for building interfaces by means of direct manipulation, especially as far as visual properties are concerned. However, they are typically restricted to a fixed set of predefined interface elements and mostly are very poor when a slightly modified appearance or functionality is required.

We have implemented a layered architecture of interactive tools for building, inspecting, and modifying graphical user interfaces for the X Window System, comprising user interface toolkits as well as visual programming tools integrated in an interactive programming environment. Their design and implementation follow the concepts of the Usit system [Herczeg 89], developed by the research group Inform at the University of Stuttgart, and make them available on a more standardized software platform. These concepts have been further developed towards a design environment for user interfaces.

2 A Layered Architecture for User Interface Design

Figure 1 shows the layered architecture of a user interface design environment for the X Window System. The layers correspond to different abstraction levels on which user interfaces may be constructed and modified [Herczeg et al. 91]. The components have been implemented in Common Lisp and Clos, the Common Lisp Object System.

- Xit is a user interface toolkit for the X Window System based on the lower-level programming interfaces Clx [ScheiflerOren 88] and Clue [KimbroughOren 90]. It is composed of two major components:
  - Xit$_{low}$ is a low-level toolkit providing a general framework for building user interface elements.
  - Xit$_{high}$ is a high-level toolkit providing a library of common user interface elements.
- Xact is a user interface builder for constructing user interfaces out of building blocks of Xit.

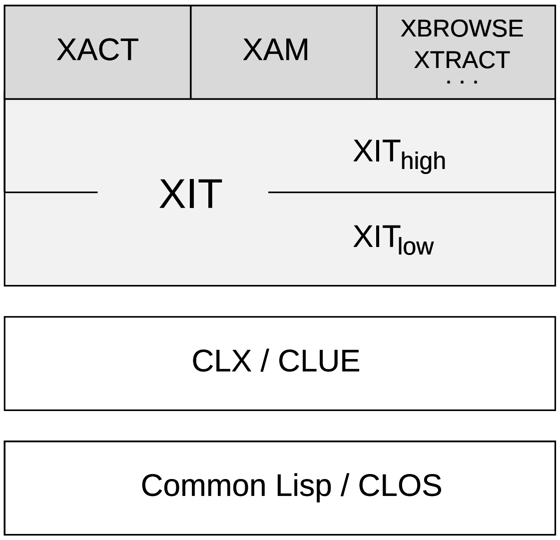


Figure 1: Layered Architecture of a User Interface Design Environment

- XAM is an interactive graphical tool for inspecting and modifying user interfaces built with XIT or XACT.
- General programming tools for CLOS, such as XBROWSE and XTRACT, are used to inspect and manipulate object-oriented application programs and their user interfaces at run time.

In the following, we will describe the different system components and how they may be used in an integrated visual programming environment by user interface designers and to some extent even by end users.

3 The User Interface Toolkit

XIT (X User Interface Toolkit) provides a general object-oriented framework for building graphical user interfaces. Its main design goals are generality, extensibility, and usability. These partly contrary goals have been achieved by separating XIT into a low-level and a high-level toolkit.

3.1 The Low-Level User Interface Toolkit

The low-level toolkit XIT$_{low}$ comprises basic building blocks and general mechanisms for uniformly building and composing arbitrary user interface elements, in the following called *interaction objects*. Figure 2, for example, shows how an interaction object representing a menu for window operations can be defined with XIT$_{low}$. Interaction objects are characterized by the following properties:

Structure: Each interaction object may be composed of other interaction objects (its *parts*) and, conversely, may be part of a compound interaction object (its *parent*). Thus, interaction objects are organized into *part-whole hierarchies*. The parts of a menu, for example, are the menu entries.

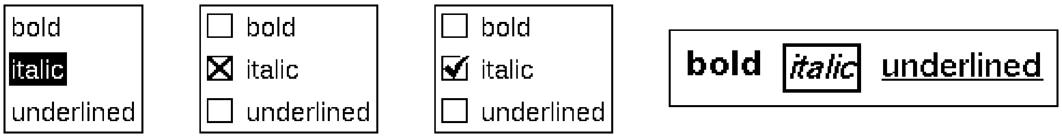
```
(make-window 'intel
             :view-of a_window_object
             :border-width 1
             :background "white"
             :layouter 'distance-layouter
             :parts '((:class text-dispel
                        :text "refresh"
                        :mouse-feedback :border
                        :reactivity ((:single-left-button (call :view-of refresh-window))))
                      (:class text-dispel
                        :text "totop"
                        :mouse-feedback :border
                        :reactivity ((:single-left-button (call :view-of totop-window))))
                      (:class text-dispel
                        :text "move"
                        :mouse-feedback :border
                        :reactivity ((:single-left-button (call :view-of move-window))))
                      (:class text-dispel
                        :text "close"
                        :mouse-feedback :border
                        :reactivity ((:single-left-button (call :view-of destroy))))))
```

Figure 2: Creation of a Menu with XIT$_{low}$

Geometry: Each interaction object contains attributes for its *position* (relative to its parent), its *size*, and *layout*. The layout of an object is based on relations between its position and size and the position and size of its parts. It is controlled by objects, called *layouters*, which define one-directional constraints between attributes of the interaction object and attributes of its parts (cf. [Barth 86, Myers 90]). Various layouters producing specific kinds of layout are predefined. In the menu example, a *distance-layouter* is used to position the menu entries one below the other; the size of the menu is determined by the position and size of the entries.

Presentation: Each visual interaction object defines how it is displayed on the screen by general attributes, like *background* and *foreground* color or *border width*. A compound interaction object is presented by displaying its parts. Basic interaction objects are *text* and *bitmap* elements or graphical elements, like *lines*, *arrows*, and *rectangles*. They form the leaf nodes of an interaction object hierarchy and are described by specific attributes, like *font*, *line width*, etc. The entries of the example menu are textual elements, depicting the possible selections. In addition to these customary interface elements, interaction objects presenting *sounds* and *videos* have been included for multimedia applications.

Reactivity: The *reactivity* of an interaction object describes the behavior of the object by mapping user interface *events*, e.g., keyboard input or mouse button clicks, onto *actions* of the underlying application. A menu entry, for example, may trigger an application-specific operation when it is clicked on with the left mouse button. Furthermore, attributes describing feedback behavior are provided.




```
(make-window 'multiple-choice-text-menu
             :view-of a_text_field
             :reactivity '((:part-event (call :view-of change-text-style *part-value*)))
             :selection '(:italic)
             :parts '((:text "bold"
                       :view-of :bold)
                      (:text "italic"
                       :view-of :italic)
                      (:text "underlined"
                       :view-of :underlined)))
```

Figure 3: Creation of a Multiple-Choice Menu with XIT$_{high}$

Application Link-Up: Interaction objects may be associated with objects of the application. Thus, user interface and application are represented by strictly separated objects communicating with each other by message passing or function calls, comparable to the *MVC paradigm* [KrasnerPope 88]. In the menu example, a selected menu entry sends a message to a window object associated with the menu by the *view-of* attribute.

XIT$_{low}$ provides mechanisms for constructing arbitrary components of a user interface. For common interaction objects like menus, however, this toolkit need not be used directly; they are available in higher-level tools built on top of XIT$_{low}$.

3.2 The High-Level User Interface Toolkit

XIT$_{high}$ is a library of predefined, commonly used interaction objects, e.g., *icons*, *buttons*, *menus* (including *single-choice* and *multiple-choice menus*), *switches*, *sliders*, and *property sheets*, as well as building blocks, like window *titles*, *scrollbars*, and *decorations*. They have been implemented out of building blocks of XIT$_{low}$. Each interaction object provides attributes to be adjusted for a particular application. Defaults are provided to show up a special "look and feel". Figure 3, for example, shows how a multiple-choice menu containing three options to change a text style may be created with XIT$_{high}$. Four different examples are presented how the menu may look, depending on the setting (or defaults) of attributes describing font, layout, or highlighting of selection.

By use of the mechanisms provided by XIT$_{low}$, the predefined interaction objects of XIT$_{high}$ may be modified or combined to more complex objects of a graphical user interface, e.g., dialog windows composed of (editable) text fields, buttons, menus, sliders, property sheets, etc (cf. figure 4).

XIT serves as a construction kit for arbitrary user interface elements. XIT$_{high}$ is just one example of a high-level toolkit. Other toolkits with different "look and feel" or even application-specific tools may be implemented with XIT quite easily.

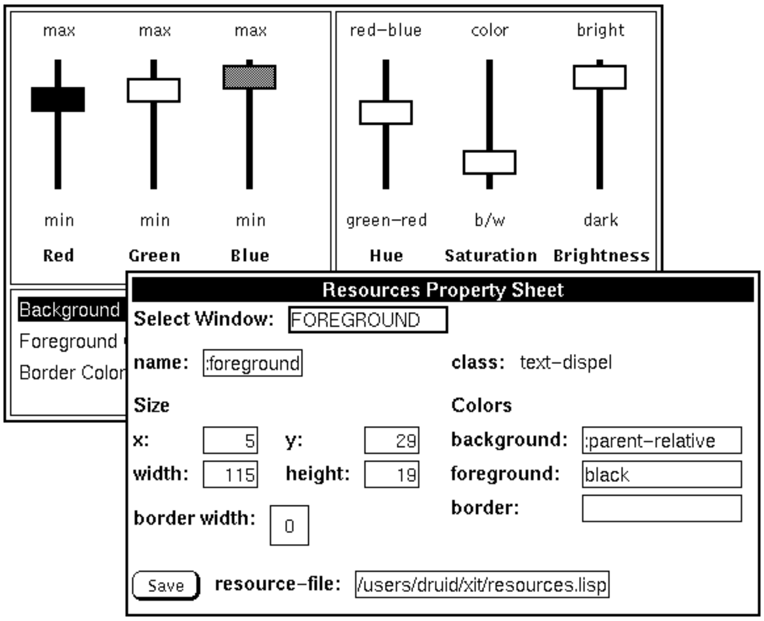


Figure 4: Resource Property Editor

By strictly separating application objects and interface objects, user interfaces may be developed and tested independently of an application program. Interaction objects of XIT may be created without links to application objects. These links may be established and changed at run time. Thus a user interface may be "plugged" into a running application. This is one of the important features required for the development of higher-level interactive programming tools for XIT. With XIT, user interfaces in various application domains have been implemented. Some of the largest applications of XIT are the tools described in the following sections.

4 The User Interface Construction Kit

The object-oriented design of XIT (and many related tools) by modularity and reusability substantially reduces the complexity of developing sophisticated graphical user interfaces. However, building user interfaces, even with the help of high-level toolkits, still requires high programming skills and deep knowledge of their functionality.

The design of XIT as an *internal* construction kit providing building blocks and mechanisms for composition suggests itself to be enhanced by an *external* visual construction kit. Especially in the domain of user interface construction, employing visual programming techniques is most natural, since the objects manipulated graphically are identical with the visual objects to be created.

XACT (XIT Application Construction Kit) is a user interface builder for XIT, which lets the user select graphically presented interaction objects and compose them to more complex user interface elements. By means of the metasystem, described below, attributes of the interaction objects involved may be modified. With XACT, a user interface may be constructed separately from the underlying application. At any time, it may be connected to an application to explore its functionality. When the constructed user interface shows the desired behavior and appearance, XIT program code may be generated automatically.

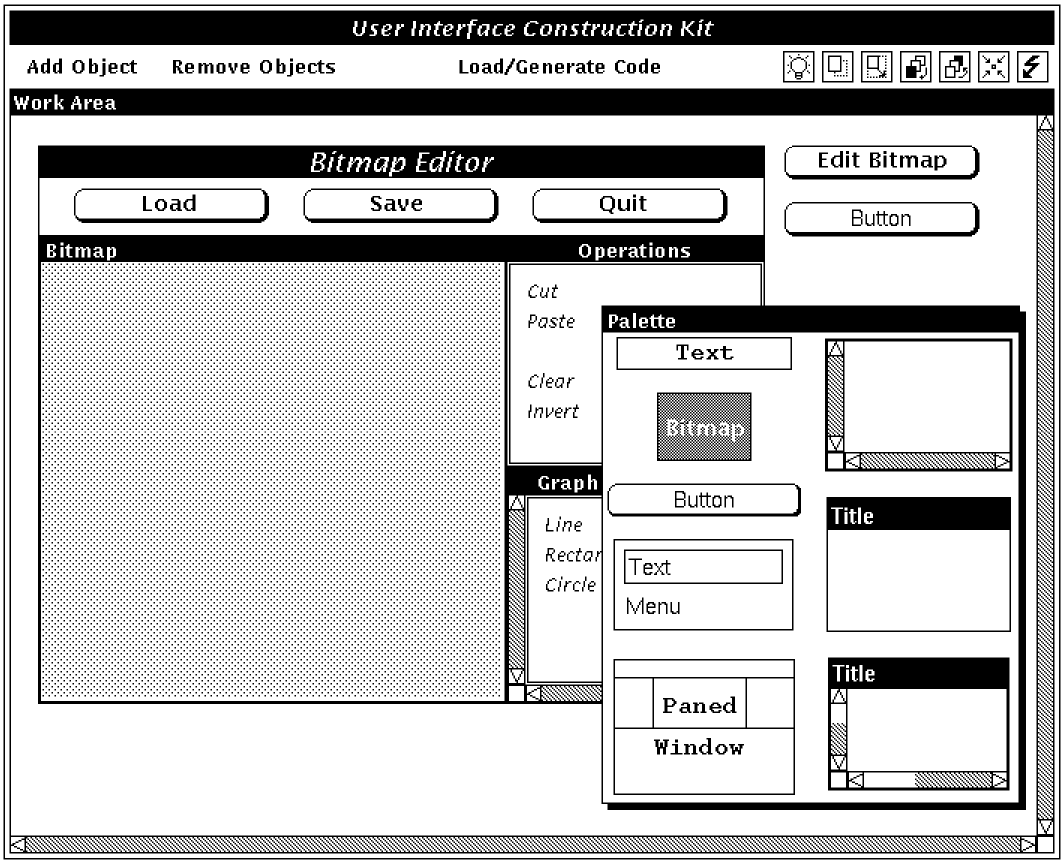


Figure 5: The User Interface Construction Kit XACT

Figure 5, for example, shows the construction kit building the user interface of a bitmap editor.

Currently, XACT does not cover all of the functionality of XIT. It only contains the most commonly used building blocks shown in the *Palette* window in figure 5. For geometrical properties, like position, size, and layout of windows, which can be best adjusted by directly manipulating graphical objects on the screen, this construction kit proved to be very helpful, especially for users with poor knowledge of XIT.

5 The User Interface Metasystem

Building a user interface is an iterative design process where a prototypical implementation is incrementally modified. This stems from the fact that very often design flaws are not detected until the user interface is viewed as a whole. Even when the design process is finished and the application is passed to the end user there are still a lot of properties which should be modifiable according to special preferences or needs of the user, e.g., changing the layout of the objects on the screen, or changing font, color, text, or bitmap attributes, or even adding or removing entries of a menu. Instead of anticipating all these needs in the user interface of an application, this functionality should be provided by a separate, application-independent tool.

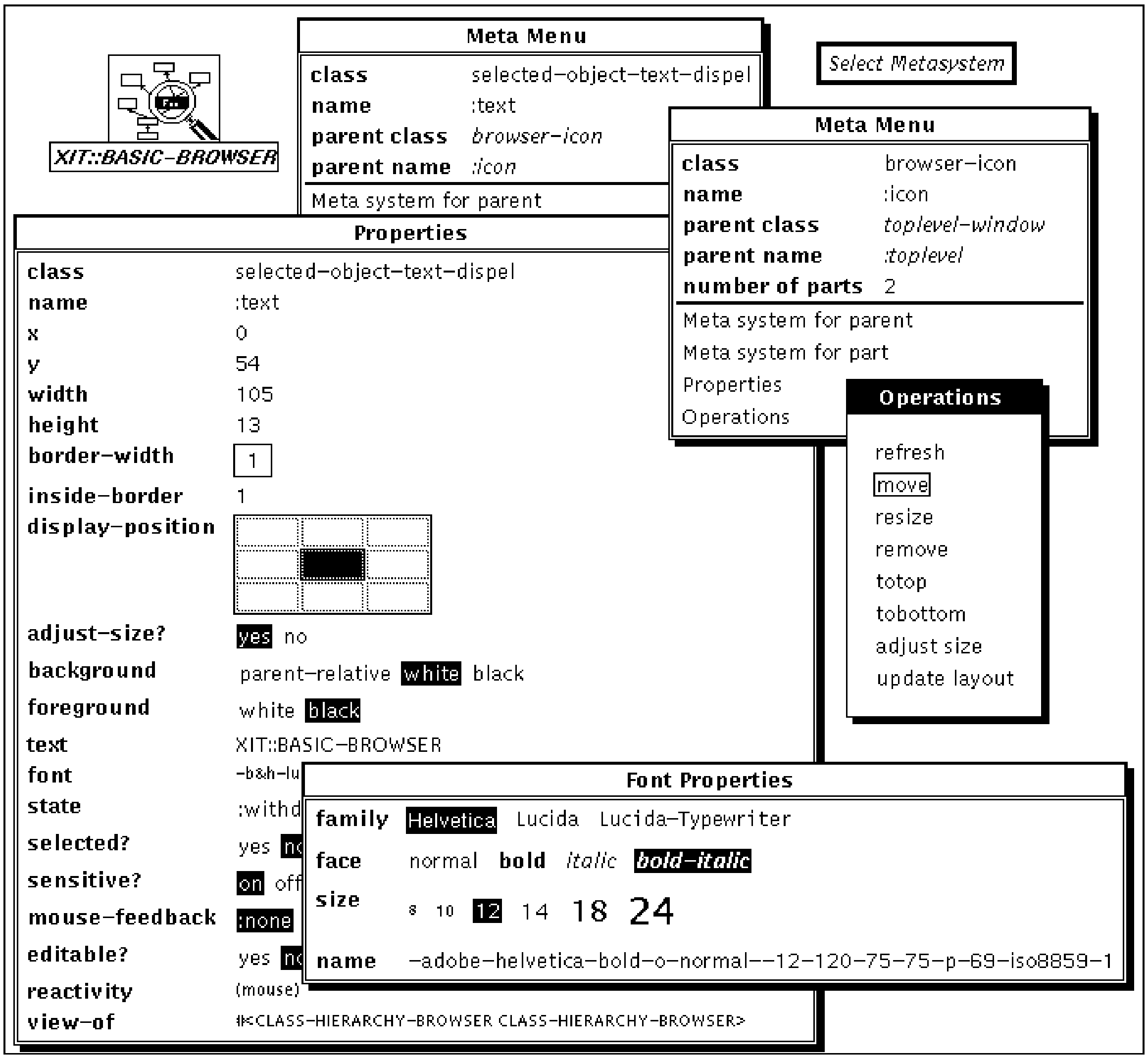


Figure 6: The User Interface Metasystem XAM

XAM (XIT Application Metasystem) is a tool for interactively inspecting and modifying user interfaces built by means of XACT or the toolkits of XIT. The metasystem may be invoked for any type of interaction object of XIT, either predefined or user-defined. It presents dialog windows containing menus and property sheets, which are dynamically created for the interaction object in question, i.e. the properties of an object determine the contents of the menus and property sheets. This process is controlled by inheritance of properties defined in the corresponding object classes. So usually there is no need to extend the metasystem for user-defined interaction objects. However, this can be easily done when an application-specific behavior is required.

Figure 6 shows different components of the metasystem invoked in a running application. The icon corresponding to an application window is inspected and modified. The property sheet for its text part has been invoked to change the font attribute, and the *move* operation is being selected to position the icon on the screen interactively. None of these modifications has been supplied by the application. The metasystem menu invoked for an interaction object provides the following choices:

- Invoke the metasystem for the parent or one of the parts selected from a menu. Thus, the user may navigate through the interaction object hierarchy.
- Change properties of the selected object. Attribute values may be either selected from menus, when a fixed set of possible values is available, e.g., font attributes, or they may be entered by keyboard, e.g., to specify an application function called as the reaction to an event. Alternatively, values may be copied from other interaction objects on the screen by simple point-and-click actions, e.g., to give a text element the font or color of an object already presented on the screen.

  Modification of particular properties, such as font, color, layout, or reactivity, trigger the invocation of more specific property sheets, e.g., for describing a font specification by *family*, *face*, and *size* (cf. figure 6), or for specifying the red/green/blue values of a color by moving sliders (cf. figure 4).
- Perform operations on the selected object; these may be either general operations, such as moving or resizing a window with the mouse, or object-specific operations, such as adding an entry to a menu.

Each modification performed with the metasystem is immediately applied to the corresponding object, which triggers an update of the presentation on the screen.

XAM is used in combination with XACT to change attributes of newly created interaction objects. When XACT produces program code the current values adjusted by the metasystem are inserted. The metasystem may also be used without the construction kit for inspecting or modifying arbitrary user interfaces built with XIT. This is not only useful for debugging but also for adapting the user interface of an application to specific needs at run time. Because the metasystem mainly works by menu selection and direct manipulation, this may be even performed by end users (cf. [FischerGirgensohn 90]).

For a set of so-called *resource attributes* of interaction objects, including position, size, border width, color, font, text, bitmaps, etc, a *resource property editor* has been added to the metasystem (figure 4). Modifications to a user interface performed with this editor can be saved on external files and reloaded, whenever the application in question is used again. Users may thus keep individual versions of the user interface for the same application.

6 An Integrated Visual Programming Environment

XACT and XAM are visual programming tools for designing user interfaces with XIT independently of an application program. Like most graphical user interface building tools, they mainly concentrate on the visual properties of an interface. For sophisticated applications, however, user interface design and the development of the underlying application are tightly interwoven. For establishing links between user interface and application objects, for specifying application functions invoked in response to user interface events, and for updating the user interface according to changes in the application, understanding and managing the application program also plays a crucial part in user interface construction. Object-oriented programming environments like SMALLTALK-80 [Goldberg 84] are based on general programming tools for inspecting and modifying user interface objects and application objects uniformly. *Browsers* provide insight into static properties of objects,

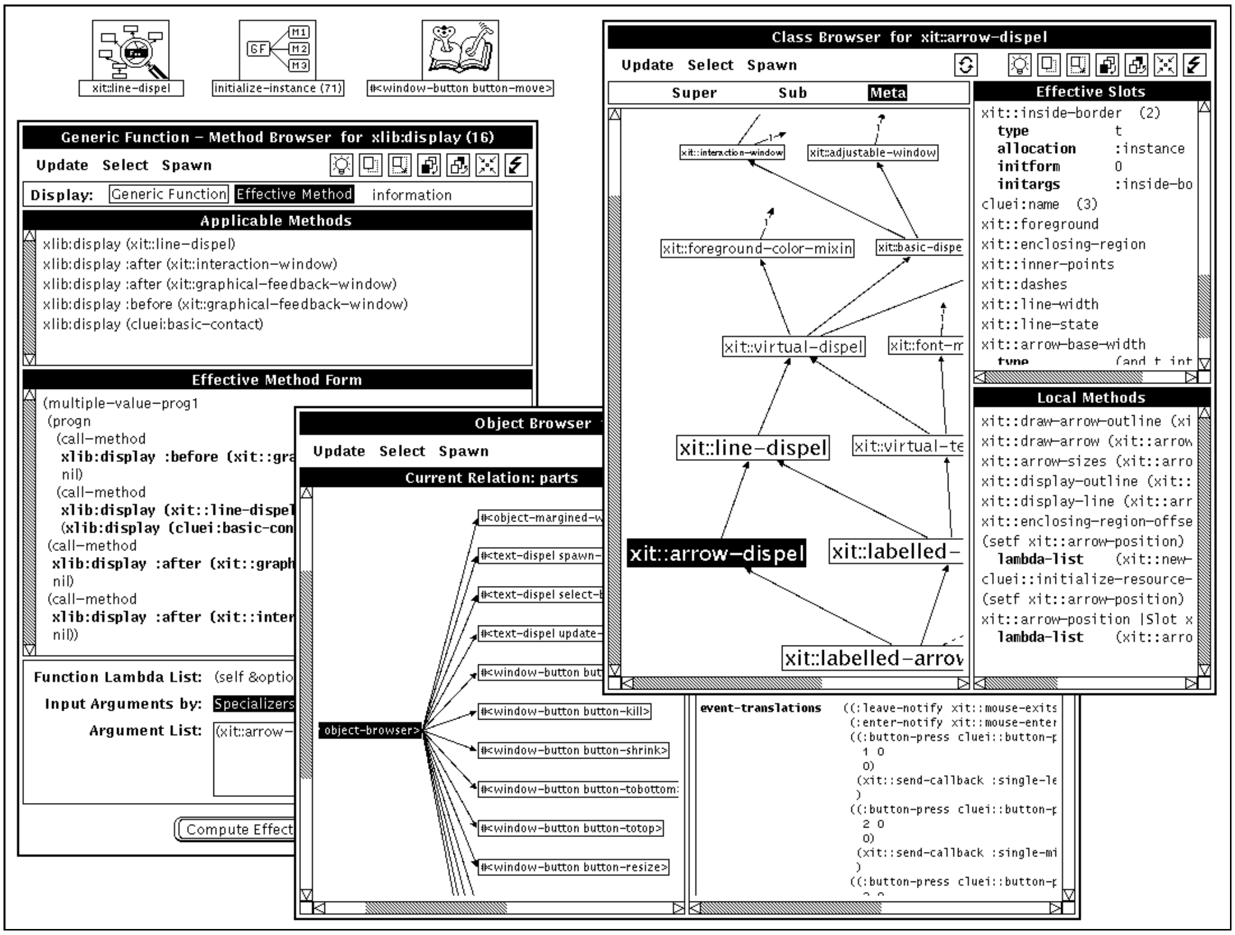


Figure 7: Browsing Tools of XBrowse

tracers visualize their dynamic behavior. By means of Xit we have implemented general visual programming tools for Clos:

- XBrowse is a browsing component including general object browsers, class browsers, and method browsers (figure 7). It combines text-oriented and graphics-oriented visualization and interaction techniques [HerczegHohl 91]. For the development of user interfaces, XBrowse may be used to visualize interaction object hierarchies and links between user interface objects and application objects and to inspect and modify the corresponding object definitions. The browsers in figure 7, for example, show a part of the class hierarchy of Xit (*Class Browser*), a window hierarchy for a selected window object (*Object Browser*), and dependencies between different implemented *display* operations (*Generic Function/Method Browser*).

- XTract is a general tracing component for generic functions and methods of Clos. In addition to textual and graphical trace information, it may provide arbitrary visual program animations. For the development of user interfaces, XTract may be used to visualize the communication between user interface objects and application objects at run time, e.g., reactivity and update operations.

As a supplement to the user interface programming tools Xact and Xam, general visual programming tools like XBrowse and XTract can play an essential part in the design of user interfaces with XIT. In an integrated visual programming environment, all of these tools could be used simultaneously, invoking each other mutually.

7 Conclusions and Future Work

We have implemented a layered architecture of programming tools for designing graphical user interfaces in an object-oriented programming environment. XIT provides a powerful user interface toolkit for the X Window System, XACT and XAM are visual user interface construction and design tools for XIT. We will further extend the functionality of these tools and integrate them with browsing and tracing components like XBROWSE and XTRACT in a visual programming environment, where we can investigate, how each of these tools may benefit from the others and conversely enhance their application. Another important aspect will be the integration of knowledge-based tools into this environment providing both tool-specific and tool-independent user interface design knowledge.

The construction and design of graphical user interfaces is still a difficult task, demanding major portions of the development time for an application. Releasing the user interface designer from low-level programming tasks and providing him with the tools appropriate for the problem at hand, will enable him to concentrate on more important and even more interesting design tasks, which eventually helps building better user interfaces. Therefore, especially for user interface design, the idea of visual programming environments will play an ever increasing part in the near future.

Acknowledgments

I want to thank Michael Herczeg, who by implementing the USIT system prepared the ground for the implementation of XIT and XAM. I would also like to thank Hubertus Hohl and Matthias Ressel, who substantially contributed to the development of the systems described in this paper, Michael Wichert, who implemented major parts of the construction kit and the metasystem, and Uta Wollensak, who implemented XTRACT.

References

[Barth 86] P. S. Barth. An Object-Oriented Approach to Graphical Interfaces. *ACM Transactions on Graphics*, 5(2):142–172, April 1986.

[FischerGirgensohn 90] G. Fischer and A. Girgensohn. End-User Modifiability in Design Environments. In *Human Factors in Computing Systems, CHI'90 Conference Proceedings*, pp. 183–191, Seattle, WA, April 1990.

[FischerLemke 88] G. Fischer and A. Lemke. Construction Kits and Design Environments: Steps Toward Human Problem-Domain Communication. *Human-Computer Interaction*, 3(3), 1988.

[Goldberg 84] A. Goldberg. *SMALLTALK-80, The Interactive Programming Environment.* Addison-Wesley Publishing Company, Reading, MA, 1984.

[Herczeg et al. 91] J. Herczeg, H. Hohl and T. Schwab. XIT — A Multi-Layered Tool for User Interface Design. In *Proceedings of the Fourth International Conference on Human-Computer Interaction*, pp. 678–683, Stuttgart, Germany, September 1991.

[Herczeg 89] M. Herczeg. USIT: A Toolkit For User Interface Toolkits. In *Proceedings of Third International Conference on Human-Computer Interaction*, Boston, MA, September 1989.

[HerczegHohl 91] J. Herczeg and H. Hohl. Building Browsers for the Common Lisp Object System. In *Proceedings of EastEurOOPe'91, Conference, Tutorials and Exhibition on Object-Oriented Programming*, pp. 27–36, Bratislava, CSFR, September 1991.

[KimbroughOren 90] K. Kimbrough and L. Oren. *Common Lisp User Interface Environment*. Texas Instruments Incorporated, Dallas, TX, July 1990.

[KrasnerPope 88] G. E. Krasner and S. T. Pope. A Cookbook for Using the Model-View-Controller User Interface Paradigm in SMALLTALK-80. *Journal of Object-Oriented Programming*, 1(3), August/September 1988.

[Myers et al. 90] B. A. Myers et al. Garnet — Comprehensive Support for Graphical, Highly Interactive User Interfaces. *IEEE Computer*, 23(11):71–85, November 1990.

[Myers 90] B. A. Myers. Creating User Interfaces Using Programming by Example, Visual Programming, and Constraints. *ACM Transactions on Programming Languages and Systems*, 12(2):143–177, April 1990.

[Nye 88] A. Nye (Ed.). *Xlib Reference Manual*, Volume 2 of *The Definitive Guides to the X Window System*. O'Reilly & Associatives, Inc., Sebastopol, CA, 1988.

[NyeO'Reilly 90] A. Nye and T. O'Reilly. *X Toolkit Intrinsics Programming Manual*, Volume 4 of *The Definitive Guides to the X Window System*. O'Reilly & Associatives, Inc., Sebastopol, CA, 1990.

[OSF 90] Open Software Foundation, Cambridge, MA. *OSF/Motif Style Guide*, 1990.

[Pfaff 85] G. Pfaff. *User Interface Management Systems*. Springer-Verlag, Berlin - Heidelberg - New York, 1985.

[Scheifler 89] R. W. Scheifler. *X Protocol Reference Manual*, Volume 0 of *The Definitive Guides to the X Window System*. O'Reilly & Associatives, Inc., Sebastopol, CA, 1989.

[ScheiflerGettys 86] R. Scheifler and J. Gettys. The X Window System. *ACM Transactions on Graphics*, 5(2):79–109, April 1986.

[ScheiflerOren 88] R. W. Scheifler and L. Oren. *CLX — Common LISP X Interface*. Texas Instruments Incorporated, Dallas, TX, 1988.

[SUN 90] SUN Microsystems. *OPEN LOOK Graphical User Interface Application Style Guidelines*, June 1990.