# User Interface Software Technology

BRAD A. MYERS

*Human Computer Interaction Institute, Carnegie Mellon University ⟨bam@cs.cmu.edu⟩*

The user interface of an application is the part that the person using the software sees and interacts with. The part of the software that makes the user interface work is often large, complex, and difficult to implement, debug, and modify. Today direct manipulation interfaces (also called *g*raphical *u*ser *i*nterface (GUI's)) are almost universal, and the part of the software that handles the user interface generally takes a significant percentage of the total system design and implementation time [Myers and Rosson 1992]. Therefore, specialized software tools have been created to ease the programmer's burden, and today virtually all new user interface software uses tools that make the implementation easier. Many of these tools have demonstrated significant productivity gains for programmers and have become important commercial products. Although most commercial tools concentrate on building the surface *look* of the interface (the placement of the menus and buttons and the choice of colors), some systems have addressed how the interface *behaves* (what happens when the buttons are pressed and objects are moved around).

There are many different kinds of user interface software [Myers 1995]. The *windowing system* supports the separation of the screen into different (usually rectangular) regions, called "windows." The X/11 window system for Unix divides the window functionality into two layers: the window system, which is the functional or programming interface, and the window manager, which is the user interface. The window system provides procedures that allow the application to draw pictures on the screen and get input from the user, and the window manager allows the end user to move windows around, and is responsible for displaying the title lines, borders, and icons around the windows. However, many people and systems use the name "window manager" to refer to both layers, since systems such as the Macintosh and Microsoft Windows do not separate them.

On top of the windowing system is the *toolkit*, which contains many commonly used widgets such as menus, buttons, scroll bars, and text-input fields. Toolkits usually connect to application programs through *call-back procedures* defined by the application programmer that are used when a widget is operated by the end user. Examples of toolkits include Motif for X/11 and the widgets in the Windows and Macintosh toolboxes. There are also many research toolkits [Myers 1995], such as Garnet [Baecker et al. 1995], Amulet, InterViews, and tcl/tk. Virtual toolkits allow an application to be written once and then to execute on Windows, Macintosh, or Motif (and others) by providing a single programming interface and many different "looks and feels." Another name for these tools is "cross-platform development systems." Examples of commercial virtual toolkits include Galaxy, Open Interface, and XVT.

On top of the toolkit might be *higher-level tools* that help the designer use the toolkit widgets, which include the following:

*Prototyping tools,* which allow the designer to quickly mock up some exam-

ples of what the screens in the program will look like. Often these tools cannot be used to create the real user interface of the program because they only simulate the widgets. The first prototyping tool was probably Dan Bricklin's Demo. Apple's HyperCard and MacroMedia's Director are often used as prototypers. A research system is SILK, which allows free-hand gestures and storyboards for sketching the initial designs [Landay and Myers 1995].

*Card-based tools,* which support interfaces that can be presented as a sequence of mostly static pages, sometimes called "frames," "cards," or "forms." The most famous example is Apple's HyperCard.

*Interface builders,* also called *Interface Development Tools,* which allow the designer to create dialog boxes, menus, and windows by laying out widgets interactively with the mouse. Other properties of the widgets can be set using property sheets. The behavior (what happens when widgets are operated) must usually be programmed separately by writing conventional code. An early research example is Steamer, and there are literally hundreds of commercial tools in this category today, including UIM/X for Motif and Novell's AppWare for PCs. Microsoft's Visual Basic is essentially an Interface Builder coupled with an interpreted programming language.

*Application frameworks,* which generally provide an object-oriented architecture that guides programmers and helps define how the software should be designed. Examples include Apple's MacApp [Wilson 1990] and Microsoft's OLE [Brockschmidt 1995].

*Database interfaces,* which support form-based or GUI-based access to databases. Major database vendors such as Oracle provide tools that allow the user interface to be designed using interactive form editors (which are essentially interface builders). Fourth-generation languages (4GLs) that support defining the interactive forms for accessing and entering data also fall into this category.

*Screen scrapers* or *front-enders,* which provide a graphical user interface to old programs without changing the existing application code by providing an in-memory buffer that pretends to be the screen of an old character terminal. A leading program of this type is Easel.

*User interface management systems (UIMSs)* [Olsen 1992], which usually provide a new language in which the user interface is programmed. Examples include HP/Apollo's Open-Dialogue, VAPS from Virtual Prototypes, and many research systems. UIMSs have used state-transition networks, context-free grammars, event languages, declarative languages, and constraint languages to specify the behavior of the interface.

*Automatic generation tools,* which create parts of the user interface such as the dialog boxes from a list of the contents. A research system that provides this is ITS from IBM Research [Baecker et al. 1995].

*Demonstrational tools* [Cypher et al. 1993], which allow parts of the interface other than the widgets to be specified interactively without programming. For example, Lapidary and Marquise allow the dynamic creation and behavior of objects to be demonstrated.

A comprehensive list of commercial and research tools for general use is available at http://www.cs.cmu.edu/~bam/toolnames.html.

## REFERENCES

BAECKER, R. M., GRUDIN, J., BUXTON, W. A. S., AND GREENBERG, S. 1995. *Readings in Human-Computer Interaction: Toward the Year 2000.* Morgan Kaufmann, San Francisco, CA.

BROCKSCHMIDT, K. 1995. *Inside OLE*, 2nd ed. Microsoft Press, Redmond, WA.

CYPHER, A., HALBERT, D. C., KURLANDER, D., LIEBERMAN, H., MAULSBY, D., MYERS, B. A., AND TURRANSKY, A., EDS. 1993. *Watch What I Do: Programming by Demonstration.* MIT Press, Cambridge, MA.

LANDAY, J. AND MYERS, B. A. 1995. Interactive sketching for the early stages of user interface design. Human factors in computing systems. In: *Proceedings SIGCHI'95.* (Denver, CO, May), 43–50.

MYERS, B. A. 1995. User interface software tools. *ACM Trans. Comput. Human Interaction 2*, 1 (March), 64–103.

MYERS, B. A. AND ROSSON, M. B. 1992. Survey on user interface programming. Human factors in computing systems. In: *Proceedings SIGCHI'92,* (Monterey, CA, May), 195–202.

OLSEN, D. R., JR., 1992. *User Interface Management Systems: Models and Algorithms.* Morgan Kaufmann, San Mateo, CA.

WILSON, D. 1990. *Programming with MacApp.* Addison-Wesley, Reading, MA.