Title:GUI builders: face lift for application development. (graphical user interface) (Open Computing)
Pub:***DEC Professional***
Detail:Philip L. Marquess and Philip E. Bourne. 12.n1 (Jan 1993): pp32(5). (2518 words)

Abstract:

The usage of graphical user interfaces (GUI) have revolutionized software packages. End-users can be up to 80 percent more productive using GUI-based applications rather than text-based applications. While GUIs have been a boon to end-users, software developers have had to spend up to 80 percent of their coding time on developing interfaces. GUI builders are interfaces that allow developers to create other GUIs. GUI builders make program development easier and this segment of the program development software market is growing dramatically. Instead of having to code, compiling and testing long listings of code, developers can lay out an interface and test it without having to write any code. Many GUI builders allow developers to drag and drop elements, as well as set attributes for each desired elements. When selecting a GUI builder, users should look for a programming language that best fits the layout and hierarchy of the application under development. Users should be able to try GUI builders before buying; there is a shareware GUI builder available via the Internet.

LISTEN CAREFULLY TO THE background noise in your workplace. The clack clack clack ding has given way to click click click, and this in turn is giving way to drag-click drag-click. So rather than continuously clicking or clacking, we are doing more productive things, such as thinking.

Users can be up to 80 percent more productive when using a graphical user interface (GUI) rather than a dumb terminal. Further, they are more accurate and can better visualize what they need to accomplish. It is not surprising that drag-and-click application processing is fast becoming the norm as aging character-based input devices are replaced by bitmapped displays.

This is good news for application users but not-so-good news for application developers, who spend up to 80 percent of their coding effort on interface development. The continuous clicking and clacking that was the responsibility of the user must now be anticipated by the code developer by way of a complex set of menus, buttons, scrolling lists and other widgets.

Coding the interface is not trivial, whether you are using DECwindows, Motif, Open Look, Microsoft Windows, NeXT's NextStep or some less popular toolkit. Take X/Motif, for example. The programmer must first appreciate what widgets are available and their numerous resources and then write a great deal of code, compile it and in it. Is a push button the wrong size? Edit the code. Should it be 50 pixels farther down? Edit the code. Should this group of buttons be replaced by a pop-up menu? Write a whole lot of new code. Recompile, and on and on.

When you are using X/Motif and DECwindows/Motif, user interface language (UIL) is of some help, but not much. UIL is a declarative, not procedural, language that specifies the hierarchy of widgets in an application, as well as their initial states. UIL is compiled into a binary form that is read at run time. The application makes one procedure call to suck in the entire hierarchy. UIL compiles very fast, so the programmer can tweak attributes of the interface without having to relink a huge executable.

But UIL is not significantly less verbose than C, nor is it less arcane. The price of a widget set's flexibility is the tremendous number of options that must be specified, which means that you learn, learn, learn and then type, type, type.

Enter the GUI builder, sometimes called a user interface management system, or UIMS. A GUI builder is

essentially an interface and associated application code for building interfaces. The importance of GUI builders is reflected by the number of software vendors trying for a segment of this expanding market.

GUI Topology

At the lowest level of a GUI is a protocol, such as the X protocol, that supports a client/server mode of operation. The protocol interfaces with the operating system, which in turn supports a library of calls, such as Xlib, from which the interface is built (see Figure 1). Building the interface directly from the library would be tedious and time-consuming, so a set of widgets that use the library are available as a toolkit, such as Xt. This may not be sufficient, so an extended toolkit such as Motif might be used. Using these particular toolkits gives an application a so-called look and feel. A UIL may be available to complement these toolkits.

Ideally, your own application code would be written so as to partition it into three layers: presentation, dialog -- which is also called the application program interface (API) -- and application. This is rarely the case, and suddenly having to modify the application to move from X/Motif to Windows, for example, involves a complete rewrite of the interface and a significant change to the application.

If the code had been modularized, only the presentation layer would need attention. GUI builders can help, since some of them support a separate API that hooks into various toolkits. The importance of an API becomes apparent when your application needs to support a variety of underlying windowing systems (see Figure 2).

Layout Editors

Building interfaces in C and UIL is like using hand tools to build power tools. An interface builder is a power tool for power-tool makers. Rather than coding, compiling and testing pages of dense code, the programmer lays out an interface and previews it before writing a line of code. This is a top-down approach (see Figure 3).

The sophistication of the layout editor varies from product to product. Ideally, it will have drag-and-drop capability. That is, you have a complete palette of widgets offered by the toolkit of the windowing system with which you are working. You grab a widget and drop it into the work area. Pop-up menus and the like allow you to select the attributes to associate with this widget.

A good test of the interface builder is to see how cumbersome it is to create a composite widget of, say, a text field and a push button. The market is so intense that any vendor will allow you to try its product before purchase, so you can try different layout tools. If you can't wait for a vendor to respond and you have access to the Internet, you can try the public domain GUI builder, Interviews, available via anonymous ftp from interviews.stanford.edu (36.22.0.175).

Widgets exist in a hierarchy. A dialog box can be the parent of a scrolled window, which in turn might be the parent of two scroll bars and a text entry box. A helpful feature of some GUI editors is a "browser" or "overview" mode, which shows the widget family tree and lets you easily select a particular widget for modification.

Another feature to look for in the layout editor is the ability to test the functionality of the layout without actually compiling it. When you are satisfied with the layout, make the interface specification available to the functional part of the application.

Some interface builders generate an API. This implies an interface that abstracts the necessary graphics and event-handling features of particular windowing systems, resulting in a system-independent viewpoint. Obviously, you can only incorporate features that are available with all the windowing systems. The API can then be implemented in terms of any actual windowing system, and applications written to the API should be portable to any system.

A good example of this is Neuron Data's Open Interface, which runs not only on X-based systems but also on Windows and the Macintosh operating system. Other systems, such as Digital's VUIT, are unabashedly dedicated to a particular widget set, usually Motif. This dependence on Motif may be a shortcoming if you wish to have

your application run both with X/Motif and on a DOS platform running Windows.

Interface Integration

Interface builders have different ways of integrating the interface with the functional part of the code. The most popular approach is simply to generate C code and possibly UIL. The programmer edits the code to complete the interface logic and to graft on the application's functionality.

Depending on the interface builder, the programmer might include calls to the builder's run-time library to perform various widget-handling chores. There is a procedural logic to the interface itself, which the programmer must specify to some degree. Although some buttons will simply trigger a calculation, others will pop open a new window or otherwise change the interface.

Just as Motif uses UIL to separate form from function, some GUI builders generate two types of output. One generated file defines the procedural logic of the interface, while the other declares its visual attributes. This style gives the same advantages that UIL does: Compiling the visual declarations is very fast compared with rebuilding the executable, and the visual attributes can be modified separately. In fact, some products use UIL itself as their declarative language. VUIT and Hewlett-Packard's Interface Architect can read anybody's existing UIL files so that they can be displayed and edited graphically.

Another solution, used by Sunrise Software's ezX and TeleSoft's TeleUSE, is to write short text files, in the manner of HyperCard scripts, that state what each widget is supposed to do in response to various events. The ezX editor writes script templates that the programmer can customize. Some script commands control the interface, such as opening or closing windows. Other commands invoke the application's callback routines.

An approach along the same lines is widget control language (WCL), which places everything you are likely to change about a widget in a resource file external to the actual application code. An extension to this methodology can be used to maintain a widget hierarchy. The nicest feature of WCL is that it is free. It is available on the Internet from export.lcs.mit.edu (18.24.0.12). A disadvantage of products that separate procedural logic from visual attributes is that learning a scripting language or learning about the resources of a widget requires some training and has no applicability beyond the specific product.

As far as the code-generation step, the nature of the code generated and how you are licensed to use that code varies markedly from vendor to vendor. If the interface code calls the interface builder's proprietary library in addition to the standard windowing system libraries, you may have an unwanted dependency that will affect the development environment and may also affect the run-time environment.

Thus, compiling your application on a different hardware platform requires a copy of the proprietary library available on that platform, and a development license. This situation is no different from having to buy a FORTRAN compiler to compile a FORTRAN application on another platform, for example, but it still is an added cost. What is worse is that you may need to purchase a run-time license, which essentially says that you pay each time you run the interface. This seems like paying every time you run a FORTRAN program that you developed yourself.

The interface builder's layout editor and code generator do only part of the job. Although a layperson might be able to design a satisfactory interface, making it fully functional still requires a programmer. The difference is that the programmer will be relatively free to concentrate on the "payload" of the application. Much of the drudgery of repeated widget coding is now automated.

Some of the interface builders have attempted to take the integration of the interface with the application one step further through the use of a dialog editor (see Figure 3). The dialog editor allows you to anticipate the response to the standard callbacks -- for example, what happens when a particular button is pushed -- through the use of a 4GL. Although this simplifies development, it introduces a further interface-builder dependency that may not be desirable.

What Else To Look For

In choosing an interface builder, you need to look at the language that is generated to define the widget layout and hierarchy you have created. Most builders generate ANSI C code and possibly UIL as used by Motif and DECwindows. Support for C++ is gaining momentum as the object-oriented language itself gains momentum. This is a plus, since the modularity inherent in the object model further supports the notion of a separate API. Quest Windows' Object Views, CenterLine Software's CenterView, XVT Software's XVT Extensible Virtual Toolkit and the public domain Interviews generate C++ code.

Also look for support for extended graphics. Given the poor support for graphics in the major toolkits, support for extended graphics may take you into the world of proprietary widgets. A balance must then be struck between portability and getting the job done. Open Interface, czX, V.1.'s X-Designer and Integrated Computer Solutions' Builder Xcessory, with its pixmap editor, are worth noting for their graphics support. Another approach is offered by Ingres Windows/4GL, which provides extensive class libraries from which graphics applications can be developed for a variety of windowing systems.

Finally, there is a genre of interface builder that is tied to a specific class of application. For example, SmartStar is designed to create a logical interface to a variety of databases.

TOOLS TO ASS[ST IN interface development are easy to come by. All support Motif, but there are those for Open Look, Windows, Presentation Manager and NextStep also. In principle, modifying an existing windowing application to use an interface generated with one of these tools should be fairly simple, because all the windowing systems use some sort of event-driven callback strategy.

The programming style imposed by an interface builder will be familiar to anyone who has written a windowing application from scratch. Retrofitting a GUI onto an old menu or command-line program will be more work, since the control flow is entirely different. But an interface builder will still do much of that work, and the necessary modifications will probably make the program easier to understand and modify. Programmers may get tired, but it will be a good kind of tired. --Philip L, Marquess is a Software Developer with the Genetics Computer Group, Madison, Wisconsin. Philip E, Bourne is UNIX Editor of DEC PROFESSIONAL.

Interface Builder Vendors

AT&T 550 Madison Ave. New York, NY 10022 (212) 605-5500

Bluestone Consulting Inc. 1200 Church St., Ste. 7 Mt. Laurel, NJ 08054 (609) 727-4600

CenterLine Software Inc. 116 John St., 20th Fl. 5959 Cornerstone Ct. W. 10 Fawcett St Cambridge, MA 02138 (617) 498-3000

Digital Equipment Corp. 146 Main St. Maynard, MA 01754 (508) 493-5111

Expert Object Corp. 500 Hyacinth R. Highland Park, IL 60035 (708) 926-8500

Four Seasons Software 2025 Lincoln Hwy. Edison, NJ 08817 (908) 248-6667

(413) 586-4144

Hewlett-Packard Co. 3000 Hanover St. Palo Alto, CA 94304 (415) 857-1501

(514) 332-6430

Ingres An ASK Group Co. 1080 Marina Village Pkwy. Alameda, CA 94501 (510) 769-1400 (303) 443-4223

Integrated Computer Solutions Inc. 201 Broadway Cambridge, MA 02139 (617) 621-0060

Island Graphics Corp. 4000 Civic Center Dr. San Rafael, CA 94903 (415) 491-1000

The Jonathan Corp. 150 Boush St. Norfolk, VA 23510 (804) 640-7200

JYACC 116 John St., 20th Fl. New York, NY 10038 (212) 267-7722

Kinesix 9800 Richmond Ave., Ste. 750 Houston, TX 77042 (713) 953-8300

(51 O) 748-6145

MetaCard Corp. 4710 Shoup PI. Boulder, CO 80303 (303) 447-3936

(908) 287-2100

Neuron Data Inc. 156 University Ave. Palo Alto, CA 94301 (415)321-4488

(413) 586-4144

Non Standard Logics Inc. 99 Bedford St. Boston, MA 02111 (617) 482-6393

(514) 332-6430

Quest Windows Corp. 4677 Old Ironsides Dr. Santa Clara, CA 95054 (408) 496-1900

Siemens Nixdorf Information Systems Inc. 200 Wheeler Rd. Burlington, MA 01803 (617) 273-0480

SmartStar Corp. 120 Cremona Dr. Goleta, CA 93116 (805) 685-8000

Sunrise Software Int'l 170 Enterprise Ctr. Middletown, RI 02840 (401)847-7868

TeleSoft 5959 Cornerstone Ct. W. San Diego, CA 92121 (619) 457-2700

Uniface Corp. 1420 Harbor Bay Pkwy., Ste. 140 Alameda, CA 94501 (510) 748-6145 (51 O) 748-6145

UniPress Software Inc. 2025 Lincoln Hwy. Edison, NJ 08817 (908) 287-2100 (908) 287-2100

VA. Corp. 47 Pleasant St. Northampton, MA 01060 (413) 586-4144 (413) 586-4144

Visual Edge Software 3870 Cote Vertu St. Laurent, PQ H4R 1V4 (514) 332-6430 (514) 332-6430 XVT Software Inc, 4900 Pearl E. Cir. Boulder, CO 80301 (303) 443-4223

This list is representative of interface builder vendors in the Digital market and is not meant to be comprehensive.

Source Citation
Marquess, Philip L., and Philip E. Bourne. "GUI builders: face lift for application development." *DEC Professional* 12.1 (1993): 32+. *General OneFile*. Web. 10 Nov. 2011.

Document URL
http://find.galegroup.com.ezproxy.lib.swin.edu.au/gtx/infomark.do?&contentSet=IAC-Documents&type=retrieve&tabID=T003&prodId=ITOF&docId=A13337744&source=gale&srcprod=ITOF&user

Gale Document Number:A13337744