Minsk, Belarus, October 2-4, 2001

User Interface Builders: Are they a Good Idea?

Begumhan Turgut, Nevin Aydin and Damla Turgut
Department of Computer Science and Engineering
The University of Texas at Arlington
P.O. Box 19015
Arlington, TX 76019-0015
Email: {bturgut, aydin, turgut}@cse.uta.edu

Abstract

This paper addresses the features of user interfaces. The importance of a good user interface is highlighted; the advantages and the disadvantages of using an interface along with some representational schemes are discussed. Then we suggest the reasons for the usage of a software tool to build a user interface. We then compare four tools based on some of the requirements elucidated. This paper is based on the importance of a Human Computer Interaction.

Introduction

A computer program normally consists of two major parts: a computational part that addresses the issues of application development and a user interface (UI) part [JWCK91]. The separation of the two parts into independent entities makes application development easier. The application developer can deal with only the issues related to syntactic elements rather than dealing with the semantic effects of the system.

The User Interface has been described by Myers, as "User Interface is that part of the computer program that handles the output to the display and the input from the user to the program" [BAM96]. The user interface has been described as a list of logical components. Each component description is a list of condition-action pairs, where a condition is a state inspection function and an action is a list of state manipulation functions [JWCK91]. The studies conducted show that 48% of the code developed is for the purpose of producing a good user interface and writing this part of the system constitutes 50% of the development time. [BAM92]. A study commissioned by NeXTStep, an organization that develops tools to help in the development of user interface tools indicate that using their tool reduced the code written by 83 % [NeXTStep]. As these studies have shown the necessity of a software tool that helps to develop a user interface has grown tremendously over the years.

A user interface development tool simplifies the coding of complex applications by providing the developer with building blocks (or widgets) of interface components. A user interface builder enhances usability by providing a development team with a prototyping capability such that proposed changes can be rapidly demonstrated to the user to secure requirements validation and acceptance. This aspect can decrease the turnaround time for making changes in the Operations and Maintenance (O&M) phase, in turn enhances maintainability.

There are four classes of people who are involved in the development and the use of the tool to develop applications. *End users* are anticipated to use the tool to develop applications. The person who designs the user interface is called the *Designer* or the *User Interface Designer*. The *application programmer* develops the application and finally the *tool creators* are the creators of the tool that the designer uses to develop the user interface [BAM95].

The user interface development tools can be broadly categorized into two types: Interface Development Tools (IDTs) and User Interface Management Systems (UIMSs). IDTs are used for building the interface. UIMSs extend the functionality of IDTs to include application development (code generation tools) or scripting tools and allow the developer to specify the behavior of an application with respect to the interface. These two types of graphical user interface (GUI) builders permit the interactive creation of the front-end GUI using a palette of widgets, a widget attribute specification form, a menu hierarchy (menu tree structure), a tool bar, and a view of the form. The UIMS adds the benefits of code generation tools, which can greatly increase the productivity of the GUI development staff. After the front-end is created by a UIMS, a code generator is used to produce C/C++ code, Motif User Interface Language (UIL) code, Java Code, Ada code or some combination of C, Ada, and UIL [GUIB]. GUIs were first thought of in the 1970's at Xerox Palo Alto Research Center; the main idea behind their reasoning was that in future computing power would be abundant and economic. The main inspiration behind the development of GUI was Alan Kay's vision of "Dynabook" [BAM95]. The growth in the use of UI builders has led to an increase in the number of tools that have been developed for this purpose. This in turn has lead to a fall in the cost of development of these tools.

2. User Interface

Some of the concerns when a UI is implemented are:

- State changes in the user interface (i.e., graphical, textual, etc.) during the user interaction.
- Functions that can be invoked by the user from the user interface.
- The necessary conditions for a particular function to be invoked.
- The inputs/outputs to/from a function and the produced results.

The state changes depend on the availability of the components, and the look and feel of the user interface. Since the state of the user interface depends on the calling function, the state change also considered being the responsibility of the designer. Hence the decision of calling a function based on the state of the machine depends on the function called. The two types of function calls are the state manipulation function calls, which may change the state of the user interface, and the state inspection function calls, which inspect the state of the machine [JWCK91]. The various function calls and the inputs to these functions are the responsibility of the User Interface Management System, which inspects the application's state, and changes to the state of the application are notified to the system.

Some of the motivations behind using a user interface are based on typical issues that relate to the usage of interfaces as described below:

- Enabling functions depend on the state of the application. Some of the functions are based on the state of the application, (i.e., the paste function cannot be called before the copy function has been called and the buffer is filled up).
- Some of the functions depend on the user setting some parameter before they can be invoked, (i.e., the user may have to enter the social security number of the user before he can hit the enter button to check the account status).
- Combining the result from multiple sources. The result to user input could lead to access of multiple resources. The results returned by the different components have to be combined and presented to the user.
- Implicit invocations are active calls being invoked when a particular state change occurs, (i.e., the document could be set to automatically save the file as soon as the user changes the contents).
- Repeated invocations are also active calls. They continue in endless loops until the user decides to break out of the loop.

2.1 Representational Schemes

As described above, the necessity for a good user interface is a representational scheme that completely describes the interface. The scheme developed needs to be analyzed for any uncertainties, violations of rules or principles and also checked for validity of the system. A user interface must not allow any loopholes that can be exploited by the user and cause harm to the system. In this section, we briefly discuss some of the more common representational schemes that have been suggested. A detailed approach to representational schemes of UI's can be found at [HH89].

The state transition diagram is one of the earliest schemes used to represent sequential transitions. Since most of the user interfaces are considered to be sequential, a state transition diagram can adequately represent the workings of a user interface. A user interface is initially in the start state or the initial state. According to a user input, it transits to a new state.

Some user interfaces are developed using the event approach (also known as sequential approach), where every stage is indicated as a window and the user decides on the next step. In this method, the user decides the next step in the sequence. In a non-sequential approach the user input may not necessarily decide the next step.

In the direct manipulation interface approach, the developer uses a package to develop the user interface. The developer can connect action listeners and triggers.

2.2 Requirements

In this section, we discuss the requirements that we proposed for a good user interface based our research. The importance of these requirements may change depending on the area the application is being developed. The requirements considered are as follows:

- Easy navigation through the system
- Provision of display management techniques
- Graphical representation for the information
- Provision of adequate user interaction
- Icons for main features
- Consistency
- Keep track of the current status
- Context Sensitive Help
- Reduced response time
- Minimize information on the screen

2.3 Advantages and Disadvantages

In this section, we discuss the advantages and disadvantages of using a user interface. The advantages are as follows:

- Separation of syntactic and semantic elements of the system. This allows the developer to concentrate on the syntactic elements of the program and develop the semantic elements separately and combine them.
- A Uniform "Look and feel" user interface would lead to ease the learning curve of the tool. When a uniform look and feel is maintained to all the interfaces in the system, the user familiarize with the system faster.

Some of the disadvantages are given below:

- It is very difficult to manipulate and generate a user interface based on direct manipulation.
- The user interface may be restricted by the existing components. This is normally a problem introduced by a tool, which does not support some of the common objects used by the user interface.
- A user interface is generally assumed to be sequential and if there is a need for any part of the system to execute parallel, a problem may arise.

3. User Interface Builders

The difficulty in developing useful user interfaces has led to the innovations in developing the tools to build them. The different components of a user interface builder can be seen in Figure 1. The main components are the toolkit and the windowing system. We discuss the windowing system, the toolkit and the higher-level tools in this paper. Interested reader(s) can refer to [BAM96] for more detailed description of these tools.

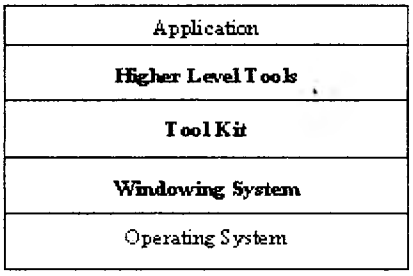

Figure 1. Components of the User Interface Builder (adapted from [BAM96])

The *windowing system* breaks up the window into different regions. It provides the procedures that allow the program to paint the user interface on the screen and also accepts input from the user and displays the result as an output to the corresponding region on the screen. Another part of the system called *window manager* allows the user to move the regions. The manager is also responsible for making sure that lines, text, and the icons are displayed properly in the user interface [BAM96].

The layer on top of the windowing system is the *toolkit*. The toolkit provides different libraries for application development. A widget is one of the main libraries that were used to include the libraries. The widget libraries include menus, scrollbars, text fields and so on. The advantage of toolkit is that most of the systems whose applications are developed using this tool would have the uniform look and feel; however, the toolkits are expensive to create. Another problem is that there are practically hundreds of procedures that have to be used to create a procedure and it is not often very clear about the way to use them [CAR85].

The higher level tools are placed on top of the toolkit layer. These tools provide phases that assist in the development of the user interface with three components: design time component, run time component, and validation component. The high level tools are separated based on the specification of the designer. There are tools based on specific languages, application frameworks, or the design model (i.e., OO based or structured design). Some tools build prototypes and provide graphical specification. A more detailed description is provided in [BAM96].

Some application domains specify tools that are customized for building applications that are specific to their domain of applications. Some of the tools that are specific to a particular domain are AutoCode and InterMAPhics.

3.1 Requirements

The requirements that we identified in this section were based on the needs for a particular class of user interfaces. The requirements considered are as follows:

1. Provision of building blocks
2. Ability to drag and drop objects onto an interface
3. Ability to manipulate object properties
4. Configuration of action listeners and triggers
5. Provision of component view
6. Ability to move components and update relevant properties automatically
7. Generation of reusable and maintainable code
8. Generation of code in multiple languages
9. Provision to view/edit code and reflect changes
10. Provision for group development
11. Provision for portability of the user interface

One of the needs for a user interface builder would be the availability of building blocks. The various building blocks that are available are widgets, canvas, panels, etc. The ability to drag and drop objects and building blocks onto the interface is considered important because the user can then drag and drop the objects that need to be on the user interface directly. As mentioned later, the ability to move them around the interface is also another of the important properties. The user interface builder has to allow the user see the various components to build an interface. If the user choose to add a particular object, it must be allowed to change the various properties of that object such as width, height, x-position, y-position etc., either manually or by choosing the property window for that item and making the appropriate changes.

Another important feature that must be supported by the user interface builder is to have means to automatically add action listeners and triggers to the various items, (i.e., detecting single click or double click on a button). This property is considered as essential. The user interface providing a good navigation facility and the availability of

icons for application development is based on detecting some kind of action listener or trigger.

As mentioned earlier, one of the most important reasons for developing tools to build user interface is to reduce the amount of code to be written. It must be ensured that the code developed by a tool is easy to understand and must follow coding standards. The code generated is important such that any changes can be successfully incorporated in the system.

With the advances in technology, programming languages are being developed to support applications in different domains. Since the programming languages are becoming more application domain dependent, the user interface should be able to generate code in multiple languages.

The feature of providing different views of the user interface would allow the user to view the code. The user would be able to manually change the user interface and the changes would be reflected back on the user interface in the other view.

Another essential feature of a user interface is portability. The behavior of the UI must not undergo and detrimental changes once it has been ported to the other system.

3.2 Effects of the User Interface Builders

One of the requirements for a user interface indicates the need for a graphical representation, which is connected to the availability of the objects for building the interface. Other features included are a good display management and easy navigation through the system. The tool must therefore allow the user to use menus, icons and lists. These features are hence driven by the availability of the building blocks available in the tool.

The features of navigation of a system and maintaining consistency in the system (i.e., the cut works the same way in all the interfaces etc.) are dependent on recognizing the action listeners and the triggers correctly.

Providing information about the status of the system and good display management capabilities are again dependent on the items that are available for building components that can access the status information. The display management depends on the properties of an item that can be manipulated by the user dynamically. This is to ensure that there is a view of the item and a view to manipulate the properties of that particular item.

Minimizing the response time depends on many factors as the user interface may have to access many system resources that may take time. This issues can be addressed by the user interface builder by providing good interfaces for items such as buttons and making sure the action listeners attached to them are sensitive.

Minimizing the information on the screen and making sure the correct information gets written to the correct region on the screen are some of the factors that must be addressed in the design phase of the system.

4. Evaluation of User Interface Builders

In this section, we first present the features of the tools considered which are Semantic Café, GIPSY, Builder Xcessory and SpecTcl, and summarize the evaluation of these tools according to the requirements of a user interface builder that we have identified in Section 3.1.

4.1 Features of the Tools Evaluated

Some of the main features of these tools considered are presented below.

Semantic Café: Some of the features available in semantic café are: [SC]

- Provides components for database connectivity.
- Provides wizards for action listeners.
- Support for Java swing components.
- The help features were very good.
- The code generated was well documented and easy to maintain.
- Changes made to the code were reflected back in the UI.
- Windows to change the component properties.
- Window to view and edit the code generated.

	Semantic Café	GIPSY	Builder Xcessory	SpecTcl
1	Panels and Canvas	Widgets and layouts	Provides motifwidgets, panels and canvas	Provides widgets
2	Provides drag and drop facility	Provides drag and drop facility	Provides drag and drop facility	Provides drag and drop facility
3	Provides a separate view to manipulate the item properties in a separate window	Provides a separate view to manipulate the item properties in a separate window	Provides a separate view to manipulate the item properties in a separate window	Provides a restricted access to change the object properties.
4	Provides wizards to add action listeners	Not Known	Not Known	Not Provided
5	Provides component view in a separate window	Provides component view in a separate window	Provides component view in a separate window	Provides component view in a separate window
6	Easy to move components around	Not Known	Not Known	Restricted by the widget size.
7	Generates well documented code	Not Known	Not Known	Code not well documented
8	Generates code in Java	Generates Tcl Scripts	Generates Tcl Scripts	Generates code in Java, C++, HTML, PERL
9	Code can be viewed in a separate window and changes made are reflected in the UI.	Not Known	Provides a browser to view and edit the code	Facility to view the code
10	Not Known	Support for group development available	Not Known	Not Known
11	Java files are portable	Not Known	Not Known	Java files are portable

Table 1. Evaluation of Tools

The numbers 1-11 correspond to the requirements presented in section 3.1.

GIPSY: GIPSY is a tool that was developed to help build UI's for large control systems. GIPSY provides some of the features like: [GIPSY]

- Provides tools for creation and manipulation of objects.
- Provides windows and views of the various items and their properties.
- Generates Tcl scripts.
- Saves the files in separate directories.
- Runs on Windows, UNIX, Solaris, etc.

- Does not provide help for action listeners and triggers

Builder Xcessory: This a tool provided to generate UI's, it generates Tcl scripts. Some of the supported features are: [BX]

- Provides support for Java swing components.
- Supports different views.
- Runs on UNIX, Irix, Solaris and so forth.
- Provides good help features.
- Does not provide help to add action listeners and triggers.

SpecTcl: Some of the features that are available in SpecTcl are: [SPT]

- Runs on Windows.
- Provides widgets that are easy to handle.
- Does not provide much help to add action listeners.
- Number of items that can be used is limited.
- Does not provide adequate help files.
- Code generated was not adequately commented.

4.2 Evaluation of selected tools.

The four tools that we selected to be evaluated in this paper were Semantic Café by Semantic Inc, GYPSY a UI Builder for control systems, Builder Xcessory a UI builder and SpecTcl a UI Builder that supports many languages.

We evaluated the tools based on the requirements identified in section 3.1. The evaluation a criterion of the tools was to try and build a UI and make modifications to the code generated. Two of the tools GIPSY and Builder Xcessory could not be evaluated in this manner as the system requirements could not be met.

Table 1 gives a brief description of the features of the four tools selected for evaluation. The property with the highest importance was the feature of the tool to provide means to drag and drop items onto some building block. It was seen that all the four tools supported this feature.

Another important feature that was required was the ability to provide wizards to add action listeners and triggers. It was seen that only Semantic Café provided that feature. All the tools provided features to edit code and move the items around the interface. All the tools also provided feature to edit the property of the items.

The code generated by Semantic Café was found to be well documented and easily maintainable. Gipsy and Builder Xcessory were not tested on the code because of system requirement conflicts.

5. Conclusions and Future Work

In this paper we presented some of the motivations behind using a user interface for application development. Studies have indicated that the Human Computer Interaction factor is very essential. The usage of a UI helps increase the Human Computer Interaction by providing the user with visual knowledge of the data interaction.

We provided the requirements of a good user interface and the features that a user interface builder tool must incorporate. We discussed the importance of each factor and identify the best tool for use. We concluded that for our application domain the tool that met most of the requirements for this class of user interfaces was Semantic Café.

The area of user interface builders is rapidly expanding with the development of classes of programming languages to aid different classes of problems. Hence there is a need to support multiple languages in a tool. In distributed computing, applications are development as separate modules and these modules are integrated later on. Future tools must therefore provide means to support distributed development and integration. Some of the new multimedia components also need to be incorporated in a user interface.

References

[BX] Builder Xcessory available at:
http://www.ics.com/

[GIPSY] GIPSY available at:
http://www.prs.de/int/products/gipsy/g4short.html

[GUIB] User Interface Builders Available at:
http://www.uky.edu/~gbenoit/637/GUIB.html

[SC] Semantic Café available at:
http://www.symantec.com/product/

[SPT] SpecTcl is available at:
http://www.interwoven.com/company/features/ajuba/

[BAM96] Brad A. Myers 1996. UIMs, Toolkits, Interface Builders

[BAM95] Brad A. Myers 1995, User Interface Software Tools. ACM Transactions on Computer Human Interaction. Vol. 2, no 1, March 1995.

[BAM92] Brad A Myers and Mary Beth Rosson. Survey on User Interface Programming. In Human Factors in Computing Systems. Pages 195-202. Proceedings SIGCHI'92, Monterey, CA, May 1992.

[CAR85] Luca Cardellli and Rob Pike. Squeak: A Language for Communicating with Mice. In Computer Graphics, pages 199-204. Proceedings SIGGRAPH'85, San Francisco, CA, July 1985.

[JWCK91] J.W.C. Koorn. Connecting Semantic Tools to the Syntax-Directed User-Interface.

[HH89] H.R. Hartson and D. Hix. Human-Computer Interface development: Concepts and systems for its management. ACM Computing surveys, 21(1), 1989.

[NeXTStep] Booz Allen and Hamilton Inc. NeXTStep vs. Other Development Environments: Comparative Study. 1992. Report available from NeXT Computer, Inc.

[TBNM] Tilmann Bruckhaus, Nazim H Mahavji, McGill University. Ingrid Janssen and John Henshaw, IBM Canada. The Impact of Tools on Software Productivity.