SANDIA REPORT

SAND97–0052 • UC–905
Unlimited Release
Printed January 1997

Programming Software for Usability Evaluation

Tony L. Edwards, Heather W. Allen

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550
for the United States Department of Energy
under Contract DE-AC04-94AL85000

Approved for public release; distribution is unlimited.

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Available to DOE and DOE contractors from
Office of Scientific and Technical Information
PO Box 62
Oak Ridge, TN 37831

Prices available from (615) 576-8401, FTS 626-8401

Available to the public from
National Technical Information Service
US Department of Commerce
5285 Port Royal Rd
Springfield, VA 22161

NTIS price codes
Printed copy: A03
Microfiche copy: A01

SAND97-0052
Unlimited Release
Printed January 1997

Distribution
Category UC-905

Programming Software for Usability Evaluation

Tony L. Edwards
Decision Support Systems Architectures Department

Heather W. Allen
Statistics and Human Factors Department

Sandia National Laboratories
Albuquerque, NM 87185

Abstract

This report provides an overview of the work completed for a portion of the User Interface Testbed for Technology Packaging (UseIT) project. We present software methods for programming systems to record and view interactions with a graphical user interface. A brief description of the human factors design process is presented. The software methods exploit features available in the X Window System and the operating system for Windows™ 95 and Windows™ NT®.

This page intentionally left blank.

Contents

This page intentionally left blank.

Programming Software for Usability Evaluation

Introduction

Graphical user interfaces are used by people throughout the world to interact with computers. It is very important for these interfaces to be designed correctly. For example, if a user is forced to use an interface incorrectly due to design problems, valuable time and resources are wasted.

Evaluating a user interface design is a time consuming, but necessary, part of the software engineering process. A development team should not put a design into the field until the design has been thoroughly tested by the actual users. In order to evaluate a design, we must find a way to record a user's interactions with the user interface and then analyze the interactions. With this information an interface designer will know how the user interface was used in a particular session. Typically this work is completed with a VCR and exhaustive analysis of the recorded or edited tapes. Software methods are needed to automate portions of the analysis process.

This document begins with a brief description of the user interface evaluation process, followed by an overview of the software libraries, protocols, and methods available to evaluate users' interactions with user interfaces. Problems associated with the use of these libraries, protocols, and methods will also be discussed.

This report assumes at least some basic programming knowledge of either the X Window System or Microsoft® Windows™ applications.

User Interface Design Evaluation Process

To properly design a user interface a designer should first understand the needs and limitations of the intended user community. These needs and limitations can be discovered through interviews, testing of current software, and questionnaires.

After the design needs are evaluated, an iterative design process is started. A user interface designer must be aware of color use, typography styles, system style guides, corporate style guides, system architectures, and many other guidelines and requirements. The designer must test the user interface during the design process. A completely working user interface is not necessary. A sketch on a napkin might be enough to test a design!

Once initial and final user interface designs are created a designer can then start the usability inspection process. This process is completed to assure that the user interface meets the requirements of the intended users. Heuristic Evaluation and The Cognitive Walkthrough Method

are two processes that are popular at the time of this writing. A very good reference of the different testing methods can be found in *Usability Inspection Methods* [1].

Although the software design process is never completed, the software should be released when it meets the intended users' requirements. This should be determined by the users and not by the user interface designer.

Overview of Software for Usability Evaluation

Motivation for this work comes from the need to automatically evaluate and record a user's interactions with a graphical user interface (GUI). Many publications and reports stress the need for human factors evaluation in the interface design process. The prototype application that was created for this project could be used by a human factor specialist to recognize actions that show confusion or inefficiency in a design. While this project did not seek to replace a human factors specialist, we wish to lower the cost, in terms of dollars and time, of the evaluation process. Some initial evaluation work can be recorded without the presence of a specialist. This work can be completed by the software or interface designer and then reviewed by the specialist.

The usability evaluation process often results in the collection of large quantities of data. This data can typically be recorded, statistically analyzed, and the results presented visually. Interface design recommendations can then be based on the qualitative results of the evaluation. If the focus is on quantitatively assessing an interface design, then it is logical to consider implementing a scheme by which the usability of an interface design could be assessed automatically with recommendations for candidate modifications presented to the designer. A GUI should be easy to use, meet international and/or corporate standards, meet windowing system style guides, and be visually appealing.

User Interface Testbed

A goal of the User Interface Testbed (UseIT) was to create an application that could help a user interface designer or human factors specialist handle the data collection for a usability evaluation. We wanted to use software that could access a GUI without modifying source code. There are many applications on the market today that can record and playback the interactions of a GUI. Many of these record and playback applications work by modifying source code. Most of these applications were implemented to do regression testing and do not help the interface designer. Screen size and location can cause problems with interaction playback. These problems are associated with the individual recording of a button press. Many applications record the x and y location of the button press. This approach is not appropriate for an interface that can change location on the screen. Many recording applications also require modification to the source code and relinking to a specific library in order to function.

The UseIT team implemented a recording and playback mechanism to visually evaluate basic task analysis techniques using existing standards based on the X Window System release

6.1 (X11R6.1) [2] and Microsoft Windows 95. Although we chose Windows 95 as our programming platform for personal computers, the methods discussed in this document are similar for Windows 3.1 and Windows NT™. With UseIT we were able to communicate with clients via an external application without modification to the clients' code. We use the term *client* to define a GUI application running on a particular operating system.

Link Analysis

Link analysis is an important part of interface design. With link analysis, an interface designer can learn what areas and features of a GUI are being accessed the most, usage of a particular feature, and what traversal paths are taken. Many human factors specialists videotape an end user using a GUI during testing. Extensive notes are taken to supplement the videotape. Reviewing this recorded data from the tested users can be difficult and time consuming. A way to visualize this data is needed.

The Layout Appropriateness [3] and AIDE (semi-Automated Interface Design and Evaluator) [4] research of Andrew Sears was the basis of our application. Although Sears incorporated many algorithms into his research, we focused on keeping the design simple. With our process the interface designer or human factors specialist can make the final decision on interface design. While automatic design of an interface may save time, there might be room for debate on whether or not we should implement such a capability. If someone could automate a professional psychologist, would you want to see a computer or a real person?

One of the many goals of this project was to save some time and effort for the human factors specialist. Our approach was to visually present data the specialist typically uses when evaluating an interface. We were able to combine the interaction data from the testers and display the result on a duplicate of the GUI. For the X Window System and Windows 95, we created a bitmap of the GUI windows in memory and then traced the user's interactions onto the bitmap. The bitmap can then be saved to disk and accessed at a later time. This is simple for the interactions of a single user, but how would we handle 100 users? Although we did not create the file format, we could have stored the interactions in ASCII form on disk. All of the files could then be read into an application that could merge the data before it was displayed on the recorded bitmap.

Related Research

One goal of this project was to expedite getting UseIT into the hands of interface developers. We did not seek to copy any research, but to incorporate public domain software, where possible, and design the missing pieces when necessary. At the time of this writing we could not find any public domain software or research projects that focused on usability evaluation of GUIs for the Windows 95 operating system. With this in mind, the related research topics below cover research that was implemented using the X Window System (X11) release 6.1. Many of these topics were the motivation for the work completed by us for the Windows 95 operating system.

Numerous public domain applications have been developed to record and playback a user's interaction with a GUI (e.g., XRECORD [5] and XTEST [6]). These applications are more appropriate for regression testing than for performing a GUI usability test. Regression testing is used to check for software errors and performance testing of an application. Although regression testing should be part of user interface evaluation it does not evaluate an interface based on user interface standards (ie. style guides, corporate or international standards). Regression testing does not record the steps the user takes through an application to get a specific task accomplished. It basically runs a series of tests to put a load on the system and application to check if it will crash or meet a set of performance guidelines.

Layout Appropriateness (LA) is a metric developed to place a cost on a GUI design based on detailed task analysis. The appropriateness of a given layout is computed by weighting the cost of each sequence of actions by how frequently the sequence is performed. An oversimplified explanation of this metric would be to compare it to an architects design of a kitchen. An architect seeks to place the refrigerator, stove/oven, and sink in the smallest triangle as possible. With this as an example, one could think of LA as a metric to decrease the amount of time spent traveling between the most frequently accessed GUI widgets, while still allowing less frequently accessed widgets to be part of the design.

A metric-based tool called AIDE uses efficiency, alignment, and balance metrics to create initial designs. With this tool, a designer can base their GUI design on the end user's tasks while allowing for overall appearance of the GUI. This tool uses the LA metric as the basis for continued research. AIDE builds LA into a usable GUI application and allows the user of AIDE to establish weights on the metrics to help in the GUI redesign process.

The Remote Access Protocol (RAP) [7] provides a protocol which communicates information about GUIs between X applications. RAP can be used to get information about a GUI, monitor its status, and even cause changes in a GUI's status.

Motivation

The related research used several approaches that would work for the UseIt application. The RAP protocol was being used for a different human factors reason (make GUIs usable by the blind), but the protocol itself will make GUIs using X Windows easier to record human factors information. A standard software approach is valuable to get a product into the hands of users. Without using standard protocols it would be time consuming trying to convince an operating system vendor to include specific hooks into the GUI for use by your application.

Microsoft has done a great job of providing hooks into the interface to implement the research methods mentioned above. Microsoft does extensive human factors testing themselves and it seems that they have given other developers the tools necessary to to there own testing. We also implemented the UseIt application using Microsoft Windows 95. This application used the standard protocols the operating system had available.

Programming Methods

A focus of this project was to create an application to visually record the interactions of a GUI. The software used to create this application could be used for other usability software. It is our intent to explain the software methods in enough detail to give other software developers a head start in creating their own applications. We used the Remote Access Protocol (RAP)[7] for our implementation on the UNIX platform using the X Window System release 6.1 (X11R6.1) and the standard hook procedures available with the Win32 Software Development Kit (SDK) for personal computers. All personal computer software development was completed by using Microsoft VisualC++® version 4.2.

Design Considerations

Developing software for usability evaluation can prove to be quite painful. For example, when recording the interactions for use with a playback system you should not record the x and y locations of the mouse button presses as your means to playback the interaction. Depending on the implementation of the windowing system, a GUI's widgets (buttons, text fields, etc.) will move in relation to each other and to the screen. If you record the x and y locations, the particular widgets that were accessed will not be correct. If possible you should record the actual identifier of the widget. This identifier is different per operating system.

A large problem, which the developers of this project have yet to overcome, is the inconsistent ways operating systems store widgets. In X11R6.1 a button is a button and a text field is stored as a text field (this explanation is moderately simplified). For our application, we centered the mouse button press on the widget. So, for X11 we were able to get the widget type and do the correct math to center the widget. For Windows 95, an application designer can choose to have a button be of the button type (class, etc.) or the button might actually be a window. To the user of the GUI the buttons look like buttons, but to the software developer a button can be created by many different methods and stored in memory as very different types. This proved to be a great headache when developing software that depended on certain widget types to calculate centers. If we looked to a window widget for our centering point we received more widgets of the window type than we wanted. For example, a drawing package uses windows to draw graphics in. If we wanted to see the drawing, we could not center on the window widget since the lines for link analysis would show up as a single point. In other words, we could not figure out the difference between a button and a canvas (drawing area). Menus are also stored as window widgets in Windows 95 and we would not be able to capture menu input.

Caution should be observed when trapping the mouse events of an application. Performance should not suffer while recording the user's interactions. Debugging your software will become close to impossible during the recording of the mouse events. Since the system is watching mouse events, it can cause errors in the mouse events of your debugging application. The only good work around is to send debug messages to the standard output for developer review.

X Window System Methods

A new protocol was released with the X Window System release 6.1 (X11R6.1) to facilitate the communication of information about graphical interfaces between X applications. This protocol, called Remote Access Protocol (RAP), can be used to script or test graphical applications. RAP was initially designed to translate graphical interfaces to a nonvisual presentation. This protocol proved to be very useful by the programmers of UseIt.

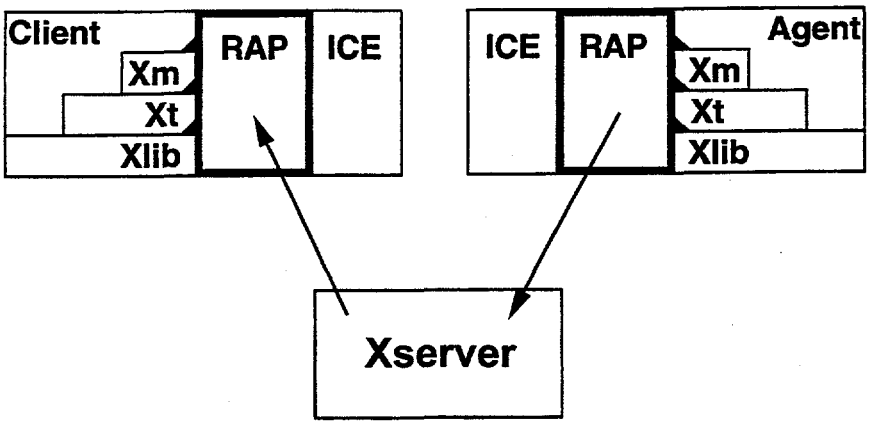

Figure 1: RAP Communication

In the RAP specification, a client is an X Window application that is compiled and linked with X11R6.1 include files and libraries. This application can use X widgets, Athena widgets, or be based on the popular Motif widget set. An agent is an external application that communicates with a client using the aforementioned protocols and RAP. A simple agent would request all resources and resource values from the client. An example of a resource would be BackgroundColor and a resource value would be Blue.

Callback List	Description
CreateHook	Called whenever a new widget instance is created.
ChangeHook	Called whenever a widget resource value is updated.
ConfigHook	Called whenever a widget's configuration (size or position) changes.
GeometryHook	Called whenever a widget's geometry is updated.
DestroyHook	Called whenever a widget is destroyed.

TABLE 1: HooksObject Callback Lists

RAP uses several standards in X11R6.1 and proposes that RAP be a standard in release 7. X11R6.1 provides the Xt library [8], the Inter-Client Exchange Protocol (ICE) [9], and ICE Rendezvous[9] used by RAP.

The basic technique RAP uses to capture interface communication is to monitor the exchange of X protocol packets between the X server and clients running on a user's machine.

RAP does this by using the ICE Rendezvous protocols. By tapping into the communication between a client and the X server, one can monitor all window creations, map and unmaps, text and graphics rendering and size changes. One major prerequisite of the RAP project was one of transparency. By being transparent, no modification of an application or the X server is necessary. One could immediately begin monitoring a GUI without having to recompile the software.

The Xt library in X11R6 introduced an implementation-dependent widget called the HooksObject. This HooksObject is also used by the RAP. The HooksObject is associated with the application's display connection and maintains a set of callback lists (see Table 1, "HooksObject Callback Lists"). These lists hold routines which are called whenever certain changes occur to an application.

ICE provides common functionality for many protocols (opening connections, validating requests, etc.). Also, ICE handles generic protocol communication while RAP provides specific protocol messages and message handlers. ICE Rendezvous was designed to establish an ICE connection between an external agent and a client in a manner which will not require awareness of a specific protocol (i.e., RAP).

RAP uses the protocols and standards mentioned above to create its own protocol to communicate with a client. RAP can make specific requests about the GUI. For example, RAP could ask for the values of all the resources of the widget tree or ask to be notified when a button press happens.

As stated before, the UseIT team implemented a recording and playback mechanism to visually evaluate basic task analysis techniques. For the X Window System, we implemented a recording capability, playback capability, and visualization of the link analysis.

The recording mechanism was implemented to record mouse button presses of the GUI user. To communicate with a client, the UseIT agent must first establish communication using the ICE protocol. Once this connection is established, the agent can gain access to the actual window IDs of the GUI widgets. The window IDs will be used for playback instead of a x and y location. This will solve the problem of widget location relative to a window or screen. All events caused by a user's interaction can now be recorded until either the client or agent ends communication. A logfile is then saved to a hard disk on the computer.

Playback of the user's interaction helps the interface designer evaluate the design. The designer can playback the interactions in real time. Playback was implemented through a procedure similar to recording. The client and agent applications are started and communication is established via RAP. The logfile from the recording session is read into the agent. The actual events are then passed to the X server. The designer can then watch the use of the interface as if the user were videotaped.

To implement a link analysis capability, UseIT duplicated the GUI by taking a bitmap snapshot and used RAP to record the user's interactions. We then redisplayed the GUI (using the bitmap image) and drew lines between the different interactions with the GUI widgets. Sears displayed his efficiency algorithm [4], using this same method. We wanted to duplicate the GUI using Xtent [10], but we came across some problems with RAP that made the translation nearly impossible. Xtent is an executable specification system that creates GUIs from files similar to X resource files. If duplication of the GUI were possible, we could have a GUI builder running and the interface designer could make changes. This could greatly help in the interface design process.

Microsoft® Windows™ Methods

In the Microsoft Windows operating system, a hook is a mechanism by which a function can intercept events (messages, mouse actions, keystrokes) before they reach an application. The function can act on events and, in some cases, modify or discard them. Functions that receive events are called filter functions and are classified according to the type of event they intercept. For example, a filter function might receive all keyboard or mouse events. For Windows to call a filter function, the function must be installed--that is, attached--to a Windows hook (for example, to a keyboard hook). Attaching one or more filter functions to a hook is known as setting a hook. If a hook has more than one filter function attached, Windows maintains a chain of filter functions. The most recently installed function is at the beginning of the chain, and the least recently installed function is at the end.

When a hook has one or more filter functions attached and an event occurs that triggers the hook, Windows calls the first filter function in the filter function chain. This action is known as calling the hook. For example, if a filter function is attached to the mouse hook and a mouse click occurs, Windows calls the mouse hook by calling the first function in the filter function chain.

To maintain and access filter functions, applications use the SetWindowsHookEx and the UnhookWindowsHookEx functions. Hooks that are of interest for usability testing are: WH_KEYBOARD, WH_MOUSE, WH_JOURNALRECORD, and WH_JOURNALPLAYBACK. The WH_JOURNALPLAYBACK hook provides the only reliable way to simulate mouse and keyboard input. If you try to simulate these events by sending or posting messages, Windows internals do not update the keyboard or mouse state, which can lead to unexpected behavior. If hooks are used to playback keyboard or mouse events, these events are processed exactly like real keyboard or mouse events. Microsoft Excel uses hooks to implement its SEND.KEYS macro function.

The UseIT team did not implement a playback capability for their Windows 95 application. Although this feature could have been implemented, the team chose to focus their efforts towards properly recording a user's interactions.

The recording mechanism exclusively used the WH_JOURNALRECORD hook to record the mouse and keyboard events. Filter function documentation for this hook can be found by searching for *JournalRecordProc* in any of the online development help books from Microsoft. The scope of a hook depends on the hook type. Some hooks can be set only with system scope; others can also be set for only a specific thread, as shown in the following list:

Hook	**Scope**
WH_CALLWNDPROC	Thread or system
WH_CBT	Thread or system
WH_DEBUG	Thread or system
WH_GETMESSAGE	Thread or system
WH_JOURNALPLAYBACK	System only
WH_JOURNALRECORD	System only
WH_KEYBOARD	Thread or system
WH_MOUSE	Thread or system
WH_MSGFILTER	Thread or system
WH_SHELL	Thread or system
WH_SYSMSGFILTER	System only

The WH_JOURNALRECORD hook has only system scope. Although most system hook filters must reside in a separate Dynamic Link Library (DLL), the WH_JOURNALxxx hook filters can reside in your application. For an explanation of DLLs see your development documentation (ie. *Visual C++ User's Guide*). *Warning*: much of the documentation on this hook contains conflicting information on where the filter function must reside. Through trial and error, it was discovered that the filter function can reside in the application that you create to monitor a GUI.

Similar to the application created for X11R6.1, we created a bitmap in memory for the representation of the GUI and drew lines from the different button presses. This will give the user interface designer a visual cue for link analysis. Any Windows 95 development documentation contains information on capturing and saving bitmap information. One must be aware of all the different graphics systems available for Windows 95 and do lots of testing.

Conclusions

This document was not meant to be a tutorial for developing software for usability testing or a human factors reference guide. We wanted to present a general overview of programming methods to implement usability software. The X Window System and Microsoft® Windows™ have similar libraries and functions to implement the recording and playback of users' interactions with a graphical user interface. These libraries and functions need to be explored further in order to discover their complete potential.

We learned that although automated testing tools are a valuable asset to testing GUI applications, there seems to be a lack of software protocols to implement such software. An area of

further research might be in modifying how a human factors specialist gathers information. This may be needed to better develop software for this specialist. The demand for tools might also increase enough to put pressure on the operating system vendors to make modifications so that tools can be better developed.

Microsoft's implementation seems to be much stronger than X Window's. The X Window System consortium is still waiting to officially release many of the Remote Access Protocol features. Microsoft's features have been available since Windows 3.1. The names and uses of the functions have only changed slightly between releases of the operating system.

Information on Microsoft development tools and procedures can be found at http:// www.microsoft.com/devonly/. Information about the X Window System and related research can be found at http://www.x.org.

References

1. Nielson, J. and Mack, R.L. *Usability Inspection Methods*. John Wiley & Sons, Inc., New York, New York 10158, 1994.

2. Gettys, J. and Scheifler, R.W. Xlib - C Language X Interface. *X Consortium Standard.* X Version 11, Release 6.1. pp. 378-395.

3. Sears, A. Layout Appropriateness: A metric for evaluating user interface widget layout. *IEEE Transactions on Software Engineering*, 19, 7, 707-719, 1993.

4. Sears, A. AIDE: A step toward metric-based interface development tools. *Proceedings of the ACM Symposium on User Interface Software and Technology* (Pittsburgh, PA, USA, November 14-17, 1995). New York: ACM, pp.101-110.

5. Zimet, M. Record Extension Protocol Specification, Version 1.13. *X Consortium Standards.* X Version 11, Release 6.1.

6. Drake, K. X11 XTEST Extension, Version 2.2. *X Consortium Standard.* X Version 11, Release 6.1.

7. Edwards, K.E., Liebeskind, S.H., Mynatt, E.D. and Walker, W.D. A Remote Access Protocol for the X Window System. *The X Resource*, Issue 13.

8. McCormack, J., Asente, P. and Swick, R.R. X Toolkit Intrinsics - C Language Interface. *X Consortium Standard.* X Version 11, Release 6.1.

9. Scheifler, R. and Brown, J. Inter-Client Exchange (ICE) Protocol, Version 1.1. *X Consortium Standard.* X Version 11, Release 6.1.

10. Blewett, D., Addessi, B., Anderson, S., Clark, C., Hicks, K., Kilduff, M. and Udovic, S. *Xtent Release 3.2: A Messaging Protocol and Specification Language for X Toolkit Based Applications.* Unpublished manual, AT&T Bell Laboratories.

DISTRIBUTION:

5	MS 0829	H. W. Allen, 12323
1	MS 0829	K. V. Diegert, 12323
10	MS 1138	T. L. Edwards, 6532
1	MS 1138	B. N. Malm, 6532
1	MS 9018	Central Technical Files, 8940-2
5	MS 0899	Technical Library, 4414
2	MS0619	Review and Approval Desk, 12630 For DOE/OSTI