Implementing A Generalized Drag-and-Drop in X

Cui-Qing Yang and *Shrinand Desai*

Department of Computer Science
University of North Texas
Denton, TX 76203-3886
cqyang@cs.unt.edu

Abstract

Drag-and-drop operation is a well-recognized and widely applied mechanism in the development of graphical user interfaces (GUIs). Most of current approaches in design and implementation of such a operation are limited to some specific objects (such as icons or drag-and-drop widgets), and are usually complex, system-dependent, and time-consuming. This paper proposes a generalized semantic and protocol for drag-and-drop that makes the draggable and droppable as intrinsic properties of any GUI objects (just like the colors and other appearance of an object) and allows each object to customize its own semantic of drag-and-drop. Therefore, the drag-and-drop feature is integrated as an inherent part of a GUI window system and all details and complexities of its implementation can be hidden from end users. The framework of generalized drag-and-drop could be applied to any GUI system. A prototype implementation in the Xt Athena widget set of X11R5 on a Unix system is reported in the paper.

1 Introduction

As the use of computers continues to grow, so does the need for good and effective graphical user interfaces (GUI). Well-designed GUIs can make the learning process easier and aid in the use and understanding of applications. With support of different window systems on many machines, such as the Macinotsh, the MicroSoft Windows, the IBM OS/2 Presentation Manager, and the X Window System, applications of GUI are now based on the manipulation of various GUI items, e.g., windows, menus, icons,scroll bars and buttons. Among them, one of the most attractive operations in a window system is drag-and-drop, with which a user can use the mouse to pick up a source item, drag it elsewhere on the screen, and drop it on some destination site so that a predetermined operation can take place. In this way, drag-and-drop emulates interactions between different types of objects in a user application. For example, a data file (representing a source) can be dragged with the help of a mouse, and dropped on a destination site (probably an icon representing printer) in order to print the file. The advantage of this is that a general user does not have to be concerned with the nitty-gritty details of various printer command syntax. Thus, GUIs, with the aid of devices like a mouse and proper usage of menus and drag-and-drop operations, become more intuitive and more friendly for end users.

Due to their ease of use and inherently intuitive nature (the way we think), drag-and-drop operations have gained popularity in building friendly user interfaces. However, this advantage of drag-and-drop comes with its own stock of problems. Firstly, it is rather involved to integrate such operations of drag-and-drop in an application program, since this requires programmers to handle low-level details that a user should usually not be expected to know. These details prove cumbersome even for a skilled programmer. Secondly, as drag-and-drop demands a smooth communication among a wide variety of non-similar applications, an universal protocol needs to be followed among the applications participating in these operations. At present, many applications are available that provide consistent mechanisms for icon drag-and-drop, e.g., the HP VUE, the IXI X.desktop [2], and the OpenWindows of SunSoft [5]. Nevertheless, their implementations are more or less ad hoc, complex and not-reusable. There lacks a general framework in supporting the operations of drag-and-drop in an arbitrary client application of most window systems.

In this paper, we present a new framework of generalized drag-and-drop for GUI window systems. The scheme is based on the object-oriented programming paradigm and provides a generalized semantic and protocol for drag-and-drop operations in a GUI system. Instead of treating drag-and-drop as specific features of specific objects, we consider *draggable* and *droppable* as two of the basic properties of any GUI objects, just like other properties such as the colors and sizes, of an object. Furthermore, mechanisms are provided for individual objects to customize these properties with *True* or *False* and to specify the semantics of what is to be dragged and what action is to be taken when a drop operation is performed within an object. With such a generalized form, the feature of drag-and-drop is integrated as an inherent part of a window system and all details and complexities of its implementation can be hidden from end users. Some of the major advantages of this method over existing techniques in design and implementation of drag-and-drop operations are as follows:

- The treatment of drag-and-drop operation as the basic properties of a GUI object follows the same spirit of object-oriented programming paradigm for GUI applications, therefore, preserves the benefits of modularity, reusability, and maintainability.

- The properties of "draggable", "droppable" and their associated mechanisms for customization provides high flexibility in implementing operations of drag-and-drop, encapsulates low-level details from end programmers, hence, greatly eases user efforts in design and implementation of these operations.

- The high-level abstraction of "draggable" and "droppable" properties of GUI objects gives a uniform framework to handle drag-and-drop operations in various GUI window system and enhances the portability of GUI code between them.

We believe that the concept of generalized drag-and-drop should be applicable to many different GUI systems. A prototype implementation of the proposed scheme has been completed in X11 R5 using Xt Intrinsics and Athena widget set [15]. Applications based on properties of new widget classes demonstrate the ease with which customized drag-and-drop operations are realized in GUI applications.

The paper is organized in the following manner. After a brief review of the related work in Section 2, Section 3 address the main issues in defining the generalized drag-and-drop. The implementation details of the widget hierarchy, the new properties of widget classes, and the protocols for inter-client communications are discussed in Section 4. Section 5 presents some testing results, and finally, Section 6 concludes the paper.

2 Related Work

The idea of using icons to represent objects in a programming environment, and the implementation of icon drag-and-drop started since the early days in the development of GUIs. The pioneering work of the Xerox Star system in the early 80's [13,7]. introduced the concept of "Desktop", which resembles the top of an office desk representing the working environment of users of computer systems. All objects, such as documents, folders, file drawers, in-baskets, and out-baskets, are displayed as small pictures called "icons". The icons are visible, concrete embodiments of the corresponding physical objects. Various operations are supported for the icons, including drag-and-drop. For instance, a document icon can be "dragged" (moved) around the Desktop and "dropped" onto the out-basket icon, which will mail the content of the document to its destination. The design principles of the Star user interface greatly simplify the human-machine relationship, making computer systems seem familiar and friendly to most users, and open a new direction for computer system design.

The success of the Apple Macintosh system and its user interface is a historical evolution in the innovative concept of desktop and icon manipulation originating from the Star system. It is the Macintosh system that fully explores the potentials of the desktop metaphor and puts it into massive implementation. As the focus of the Macintosh user interface, the Macintosh Finder [3] supports a desktop environment, which manages all the resources and objects an application programmer needs. All objects are represented by icons, including the documents, folders, applications and disks. Operations on the objects, such as open, copy, move, discard, rename, and lock, can be performed via the manipulation of their icons. Among them, the icon drag-and-drop is one of the best features.

Both Star DeskTop and Macintosh Finder support the usage of icon drag-and-drop for user applications. Mechanisms are available for users to specify customized properties for the appearance and behavior of icons representing their applications. However, the implementation of icon drag-and-drop are proprietary of the companies, and are deeply embedded in the systems. No general framework is provided for the adoption of icon drag-and-drop operation for arbitrary applications in these systems.

With all of its power and features for GUI development, the X Window System offers a perfect vehicle for the implementation of GUI applications. There are X-based desktops and applications available on many different systems that provide operations of icon drag-and-drop. Some of them are HP VUE, IXI's X.desktop [2,6], and the OpenWindows of the SunSoft [10]. The OpenWindows is a development tool designed for building graphical user interface for user applications [16]. It is based on the X Window System, and supports programs in three different Toolkits of Sun Microsystems – the XView Toolkit, the NeWS Toolkit, and the OPEN LOOK Toolkit. With the help of glyphs palette, panes, and menus in the OpenWindows, users can easily select various UI elements (User Interface elements) and assemble them together into an interface for their application. This gives programmers the freedom to create and test user interfaces without writing any code.

The OpenWindows supports nice features for implementing drag-and-drop in general user applications. For that purpose, *drag* and *drop targets* are defined as regions in control areas where drag and drop operations can be performed. Users can simply choose the instances of these targets in their targeted interfaces. Edit windows are provided in the OpenWindows that allow users to specify a variety of properties for the selected drag and drop targets. These properties, such as the label, the location (x, y coordinates), the color, and the cursor, define the appearance and behavior of the drag and drop targets. In this way, users can easily customize drag-and-drop operations in any of their individual applications. This makes the OpenWindows different from other desktops like the Macintosh Finder and the Star system.

3 A Generalized Drag-and-Drop

In many GUI systems, drag-and-drop is implemented as specific properties of only limited GUI objects, such as icons or windows. Also, the semantics and implementations of drag-and-drop have always been system specific. Moreover, many GUI systems support the concepts of "clipboard" and "cut-and-paste" that can be used to move around some text or graphics contents from one application to other applications. Both of these features have similar semantics and functionalities as the drag-and-drop operation, however they are usually handled differently from the implementation of drag-and-drop. There lacks a uniform model in providing the semantics and functionalities of drag-and-drop in different window systems. This leads to the difficulty and complexity in the design and implementation of drag-and-drop operation in general GUI applications. The basic considerations of our framework are as follows:

- *Drag-and-drop as a basic property* [1] - Drag-and-drop is considered as a basic property of any GUI (widget) classes and their objects just like other properties as colors and sizes. For this purpose, two Boolean flags: *draggable* and *droppable* are defined in every widget class, and each object instance of a class can customize these properties with *True* or *False*, which indicates whether or not the object supports the operation of drag-and-drop.

- *A generalized semantic of drag-and-drop* - With our approach, the drag-and-drop concept becomes very generalized and more flexible. It lets applications customize the drop operation as well as the drag operation in any specific widget instance (object). The semantic provides a general framework for dragging and dropping an object from a drag site to drop site, but it does not restrict the functionality or operations. These are completely left to the application programmer in customizing the operations at each specific object. For example, in one instance, the user may decide to specify the operation of dragging a file name in a directory object and dropping it on another directory object as to move a file from one directory to another, and on another instance, the user can decide to customize the same drag and drop site so that only the color of one directory window is moved (copied) to another directory window. In order to support such a generalized drag-and-drop, two callbacks (procedure properties) are added to each widget class: *dragcallback* and *dropcallback*, with which applications can customize operations of dragging and dropping in each individual widget object.

- *A general protocol for drag-and-drop* - One important component of the generalized drag-and-drop is the protocol for communications between the source (dragging) and

[1] Properties of GUI objects are often referred as *resources* in many GUI systems. In this paper, we use both term interchangeable unless it is specifically noted.

destination (dropping) objects. Generally, the protocol requires the sharing of certain information between source and destination objects, including the identification of the dragging object, the draggability of the object, the source data associated with the dragging object (that is generated from the dragging callback), the location and identification of the dropping object, its droppability, and the operations (callback) of the dropping object.

There can be different approaches for protocols sharing information between source and destination objects. A straightforward way is to have source and destination communicate directly using messages (such as the *ClientMessage* in X) [6]. That is, during the course of dragging a source object, the source can first initiate a message to the destination with the information of its identification, and then the destination will acknowledge a message back to the source with some additional information. More messages can be exchanged between source and destination if they are needed. Another way to arrange communication between source and destination is not to have them exchange messages directly. Instead, a third party, such as the window manager in X, can be involved as an intermediary between the source and destination for exchanging information. There are some more ways to arrange the drag-and-drop protocol in which the messages between the source and destination can be reduced and even eliminated. One of such a protocol is to have a shared data structure (database) between the source and destination. Each destination is required to pre-register its information in the database (the identification, the location, and the callback), and the source can search for the related information after it is dragged [6].

The framework of the generalized drag-and-drop imposes no constraints to the specifics of the protocol as far as there is such a protocol to support communications between the source and destination objects. It is up to the implementors for the decision of general strategy and detailed steps in a protocol. Certainly, a standardized protocol will help the portability of the generalized framework, which is beyond the scope of this paper. A specific protocol based on a shared database of dynamic registration was chosen in our testing implementation presented in the paper.

Also, a complete drag-and-drop protocol specifies all details for the interaction and feedback between the source and destination objects, such as the change of placement and visuality of the source during dragging, and the change of visuality of destination at dropping. For simplicity, our discussion ignores some of those details and only concentrates on the main interactions (the data exchange and callback invocation) of the protocol.

The framework of our generalized drag-and-drop does not depend on any specific GUI system and the hardware platforms. As our model is based on the object-oriented approach and the abstraction of GUI classes, it can be integrated to the most of current GUI window system without much difficulty and provide a common ground in the design of drag-and-drop operations.

4 Prototype Implementation

In this section we present a prototype implementation of a generalized drag-and-drop. For simplicity the prototype was limited only to certain necessary resources. The following subsections go through the widget hierarchy, resources and protocol typical to this prototype.

4.1 Widget Class Hierarchy

We choose to use Xt Intrinsics and the Athena widget set (Xaw) [15, 5] as the testing ground for our prototype implementation. Xaw has a fairly complete sets of widget classes. More importantly, it comes with a full source code, which makes it easy to alter it and test our implementation.

Generally, Xt Intrinsics provides basic mechanism to create new widget classes. Xt Intrinsics defines the Core class as the basis of all widgets in its toolkit. Thus, the class record of the Core class embodies information common to all widgets [1]. Certainly, we could have implemented our prototype in the Core widget. However, as our first effort, for simplicity we limited our hierarchy to the Simple widget in Xaw, a direct subclass of the Core class (in this way, we avoided to alter any code in Xt Intrinsics). Simple widget class is a very general widget class that has a rich set of children classes such as List, Text, and Label. All of its children inherit properties like window and cursor from the Simple widget class. Figure 1 shows the class hierarchy of the Simple widget class.

Figure 1: Simple Widget Class Hierarchy in Xaw

Besides, in order to further test our implementation, we also made a whole new widget class called Icon widget class as a child of Simple widget class. The Icon widget inherits properties of drag-and-drop from Simple widget class, and can be used for testing drag-and-drop operations of icons, which is popular in many GUI systems.

4.2 New Properties (Resources)

Since the Simple widget class is directly derived from the Core widget class, it automatically inherits all the properties defined in the Core. Additionally, we need to define new properties that are specific to the Simple widget class for the sake of drag-and-drop operations.

Four public resources are added to Simple widget class: 1) draggable, 2) droppable, 3) dragCallback, and 4) dropCallback. (Figure 2 shows some selected resources used by Simple widget class). These resources can be set by an application to inform a widget if it is draggable or droppable or both, and to define the functions associated with dragging and dropping operations. Since the dragging and dropping feature is treated as resources, it provides all the advantages and flexibility of resource settings.

Name	ClassRep	Type	Default Value
background	Background	Pixel	XtDefaultBackground
border	BorderColor	Pixel	XtDefaultForeground
borderWidth	BorderWidth	Dimension	1
cursor	Cursor	Cursor	None
cursorName	Cursor	String	NULL
destroyCallback	Callback	Pointer	NULL
height	Height	Dimension	0
insensitiveBorder	Insensitive	Pixmap	Gray
pointerColor	Foreground	Pixel	XtDefaultForeground
sensitive	Sensitive	Boolean	True
width	Width	Dimension	0
x	Position	Position	0
y	Position	Position	0
droppable*	Droppable	Boolean	False
draggable*	Draggable	Boolean	False
dragCallback*	Callback	Pointer	NULL
dropCallback*	Callback	Pointer	NULL

Figure 2: Resources of Simple Widget Class
(Note: * indicates the new resources added)

Some private variables are also added to Simple widget which are not visible to user applications. These resources (e.g. source, destination, source and destination type and size) are used internally by Simple as well as the subclasses of Simple widget. These resources are available in a read-only form to an application when the user sets callback routines for draggable and droppable widgets. Some information available to a user through a callback are 1) source (window) id, 2) destination (window) id, 3) source data pointer, and 4) mouse position (x and y coordinates) where the source was dropped, which can be useful when a user wants to pop up a selection box at the drop site. The source type can be either text or widget ID. At this point the drag and drop feature is truly a general feature. In future, when different types of data can be dragged and dropped, more fields of type information need to be added in the class record.

4.3 The Protocol

In order for any drag-and-drop operation to take place successfully every source object must have information about all the potential drop sites or destination objects. Moreover, potential dropsites may be in different applications that have no knowledge of one another therefore there must be some kind of protocol support for cooperation between source and destination objects. That is, information must be available to every source from each destination and must be dynamically updated from time to time. Based on this information, when a source is dragged , it can be determined whether the pointer is in a dropsite, whether a drop would succeed, and what the resulting operation would be.

In our implementation, this form of information-sharing is achieved by defining an information database and an incorporative protocol between source and destination. Because the X server is shared by all applications displayed at the workstation, they can use the server as a conduit to pass information among one another. Actually, X provides a general mechanism, *properties*, for sharing of information among different applications. A property is a collection of named data items. Applications in X can communicate by hanging properties on windows, as if the properties were tags. Any application can come along and write information on any property, or can read any property. The X server does not show the properties to the user, but merely stores them so applications can share them. Every property has a name associated with it. A property name is an arbitrary character string. The X server keeps each window's properties separate from those of all other windows. Each property also has a data type. Before communicating via properties, applications must agree about the name and type of the property they will use to pass information. Applications must also agree about the window on which the property they will use to pass information.

We decided to use the property with the root window as a shared database. The data consists of a collection of widget identifiers of all instances of potential destination objects that are created by applications. This database is built through the procedure of "destination registration" that is a part of the methods in the Simple widget class. Since this property resides in the root window associated with the X server, it is visible to every application.

Whenever an instance of a destination widget class (with its *droppable* property as *True*) is created, it registers to the database with its widget identifier, which is unique for every widget. This is done by having the object call a method which sets a property associated with the root window. Similarly, whenever a destination object is destroyed it deletes its identifier from the database. In this way the database is built dynamically and always reflects the identifiers of the most current dropsites. When a source is dragged, it calls a method in the widget class that extracts all current widget identifiers from the database. This is called binding. Once the client has all the widget identifiers it can easily retrieve the geometry and other information from the X server and know which dropsite the pointer is located in. Since the database is located in the root window, this scheme can support drag-and-drop across two different applications. Once the source object has the identifier of the dropsite, a *ClientMessage* event can be sent to the destination with source's own identifier. In this way, a two-way communication between the source and destination can be created.

The following pseudocode for the draggable and droppable feature added in Simple widget class:

```
/* The droppable part of the implementation: */

    IF a widget is droppable

        - At the creation: pre-register its widget ID
        - At the deletion: withdraw its registration
        - Solicit ClientMessage event for messages from source
        - After receiving message from a source:
            - Identify the source, and receive the data
            - Invoke the dropCallback installed by user
            - Acknowledge the source

    /* The draggable part of the implementation: */

    IF a widget is draggable

        - When a pre-assigned mouse button is pressed and dragging:
            - Get widget IDs of potential dropsite
            - Check the current mouse position
        - When the mouse button is released for dropping:
            - Identify the destination, if it is droppable:
                - invoking the dragCallback for source data
                - Send a message to the destination with the data
                - Receive the acknowledgment
```

Figure 3: Pseudocode for Drag-and-Drop Operations in Simple

5 Testing

For testing purpose, we implemented the drag-and-drop feature in the List widget, which is a sub-class of the Simple class. Also, we implemented an icon widget class (called Ficon for Function Icon). The icon widget inherits the drag-an-drop feature from its superclass Simple widget, and also provides resources for customizing its appearance, such as its pixmap and label. With our approach, implementing the drag-and-drop in rest of the subclasses of the Simple widget becomes trivial. All the implementation details are transparent to users. All that a user needs to do is to set the corresponding draggable and droppable resources, and to install the drag and drop-callbacks. Since resources are used in setting the drag-and-drop operation, a resource file can be used to change these settings without re-compile the program.

To test our implementation, we made a simple file manager. As shown in Figure 4, two list widgets (in the left) and five icon widgets (in the middle and right) are created in our testing. The list widget in the upper left (List1) contains a list of file names and is only set *True* to *draggable*, whereas the list in the down left (List2) contains a list of directories and is set *True* to both *draggable* and *droppable*. The drag-callbacks in List1 and List2 set the source data to be the list that is currently selected in the widget (the selection operation is one of the features provided by the List widget class; in our testing, we only use the single selection). The drop-callback in List2 set the destination also to be the currently selected list.

The five icon widgets are customized with their own labels and pixmaps. Each of them is set *True* to *draggable* and the drag-callback sets the source data to be the widget ID of the icon. Only three icons: "XEDIT", "FILELIST", and "QUIT", are set *True* to *droppable* and their drop-callbacks set the destination functions as invoking a simple X editor for a named file of the source data, displaying a window with the directory contents of the source data, and destroying a widget with the ID in the source data, respectively.

Figure 4: Screen Dump of the Testing Drag-and-Drop Application

In our testing, files (in List1) can be dragged and dropped to directories (in List2) as a copy operation, files can be dropped on an icon drop-site (in XEDIT) invoking edit operation on the file. Directories (in List2) can be dragged and dropped to itself (List2) as a copy operation to other directories. Directories can also be dragged and dropped on an icon (FILELIST) to show the files contained in the directories. An icon widget can be destroyed by dragging and dropping into the QUIT icon. In the same way, a file or a directory can be dragged from their list widget and be destroyed by dropping it into the QUIT icon (some sort of polymorphism is supported there).

The drag-and-drop feature worked without any problem in both List and Icon widgets. Overall, the whole system behaved well with almost instantaneous response time. Even with as many as thirty source and destination objects the time taken was much less than the tolerable limit. The drag-and-drop operations also worked across different applications. The most interesting point is that all these various forms of drag-and-drop can be achieved with the single framework of generalized drag-and-drop.

Figure 5 shows the sample code of our testing program. This demonstrates that with the support of generalized drag-and-drop, implementation of drag-and-drop operations in a user application becomes simply a matter of setting some resources and callbacks in GUI objects.

```
/* Declare all necessary widgets */

Widget toplevel,form_widget,list1,list2,ficon1,ficon2,ficon3;

/* write necessary callback routines */

void l1_drop_callback(w,client_data,call_data)
Widget w;
caddr_t client_data;
XawDropReturnStruct  *call_data;
{
/* Code for drop operation of List1 */
}

void l2_drag_callback(w,client_data,call_data)
Widget w;
caddr_t client_data;
XawDragReturnStruct  *call_data;
{
/* Code for drag operation of List1 */
}

main(argc, argv)
int argc;
char *argv[];
{
    /* Initialize the shell widget here */
    ...

    /* Create new widget instances */

    n = 0;
    XtSetArg(args[n],XtNdragable,TRUE);n++;
    ...
    list1 = XtCreateManagedWidget("List1",
              listWidgetClass,form_widget,args,n);

    n = 0;
    XtSetArg(args[n],XtNdropable,TRUE);n++;
    XtSetArg(args[n],XtNdragable,TRUE);n++;
    ...
    list2 = XtCreateManagedWidget("List2",
              listWidgetClass,form_widget,args,n);

    n = 0;
    ...
    ficon1=XtCreateManagedWidget("Ficon2",
              FiconWidgetClass,form_widget,args,n);

    n = 0;
    ...
```

```
    ficon2=XtCreateManagedWidget("Ficon2",
             FiconWidgetClass,form_widget,args,n);

    n = 0;
    ...
    ficon3=XtCreateManagedWidget("Ficon3",
             FiconWidgetClass,form_widget,args,n);
    ...

    XtAddCallback(list1,XtNdropcallback,l1_drop_callback,NULL);
    XtAddCallback(list2,XtNdropcallback,l2_drag_callback,NULL);
    ...

    XtRealizeWidget(toplevel);
    XtMainLoop();
}
```

Figure 5: Sample Code of a Drag-and-Drop Application

6 Conclusions

Due to its interactive and visual nature, we argue that the operation of drag-and-drop should be an intrinsic part of any GUI systems. Unfortunately, most of current GUI systems treat drag-and-drop as an exception rather than a norm. This leads to difficulty and complexity in implementing operations of drag-and-drop in user applications. This paper presents a framework of generalized drag-and-drop that integrates the drag-and-drop operation as part of a GUI system and supports various forms of drag-and-drop in a broad range. A prototype implementation in X window system shows that the framework is both feasible and beneficial. Certainly, more issues need to be addressed before this scheme can be applied in real production systems, including the definition of drag-and-drop semantics in various GUI objects, the support of polymorphism for different data types in the operation, and the design of a complete protocol.

References

[1] M.S.Ackerman, D.Converse, and R.R.Swick, "Widget Internals," *Tutorial Notes of 6th Annual X Technical Conference*, January 1992.

[2] Alan Southerton, and Steve Mikes, "Friendly Desktops", *UnixWorld*,8(9), Nov. 1991.

[3] Apple Computer, Inc., "Inside Macintosh," Vol. III,1988.

[4] Apple Computer, Inc., "Macintosh System Software User's Guide," Version 6.0, 1991.

[5] P.A.Asente and R.R.Swick, "X Window System Toolkit," Digital Press, 1991.

[6] G. Beged-Dov and E. S. Cohen, "Implementing Drag & Drop for X11," *The X Resource*, Issue 1, Winter 1992, pp. 169-190.

[7] J.D.Foley, A.van Dam, S.K.Feiner, and J.F.Hughes, "Computer Graphics Principles and Practice," 2nd Ed., Addison-Wesley, 1991.

[8] J. McCormack and P. Asente, "Using the X Toolkit or How to Write a Widget," *Proc. of the Summer 1988 USENIX Conference*, 1988, pp. 1-13.

[9] S. Mike, "New Ways to Program in Object-Oriented X," *UNIX World*, Vol. 8, No. 8, August 1991, pp. 103-108.

[10] Sun Microsystems, Inc., "OPEN LOOK Graphical User Interface Functional Specifications," Addison-Wesley, 1989.

[11] Neuron Date, "Open Interface Reference Manual, Version 1.03," Neuron Data, Inc., 1991.

[12] Open Software Foundation, "OSF/Motif Programmer's Guide," Release 1.1, Prentice Hall, 1991.

[13] D. Smith, C. Irby, R. Kimball, and B. Verplank, "Designing the Star User Interface," *Byte*, Vol. 7, No. 4, pp. 242-280, April 1982.

[14] J. D. Smith, "Object-Oriented Programming with the X Window System Toolkits," John Wiley & Sons, 1991.

[15] R. Swick, and M. Ackerman, "The X Toolkit: More Bricks for Building User Interfaces," *Proc. of the Winter, 1988 USENIX Conference*, pp. 221-233, 1988.

[16] D. A. Young, and J. A. Pew, "The X Window System Programming & Applications with Xt, OPEN LOOK Edition," Prentice Hall, 1992.