A Modeling Tool for X-Window Application Software Development

by Joon Seok Lee
Sang Bum Lee
Dong Hae Chi

This paper presents a modeling tool, so-called X-ADD (X-window application Analysis & Design Diagram), which is useful for the understanding of existing X-window application program and the development of new X-window application software. This X-ADD tool is available for the X-window applications maintenance activities such as program understanding, and program analysis by the concept of reverse engineering. In addition, it supports the analysis and design for the new X-window application system development. Therefore, by the use of this tool, the visibility of the existing X-window application programs can be enhanced and modeling for the design of new X-window application systems can be easily made.

I. Introduction

As the needs of "Look and Feel" interface have increased, the GUI (Graphic User Interface) on the X-window system [1] with which the end users can access the computer easily without the knowledge of detailed operations has emerged. Usually the use of software having GUI provides much easiness compared to that of non-GUI software, but the development and maintenance of GUI-based application software still remains as a troublesome task to the programmer. Moreover, in spite that many GUI builder tools [1] which are useful for coding GUI programs have been appeared, they are not useful for analysis, design, and maintenance phases in the software life cycle because of their lack of supporting power for the understanding and analysis of GUI-based applications.

In general, X-window application software consists of three major parts; the user interface, the engine, and the callbacks [2]. While the user interface part (presentation layer) controls the appearance of somethings on the screen, the engine (application layer) has a set of the algorithms which perform applicative computation like other user-defined functions. Callbacks (dialog layer) help to communicate between the underlying application and the user interface. Fig. 1 shows

the general structure of X-window application. Since underlying modeling concepts differ from each other, i.e., the user interface part adopts the concept of object-oriented and the callback part follows the event-driven approach, most existing modeling techniques such as flow charts [3], structure charts [3] and object modeling diagrams [4] are not suitable for the modeling of X-window application software. In addition, many techniques of program understanding and reverse engineering techniques [5, 6, 7, 8, 9] are not helpful for the analysis and understanding of GUI-based applications. Because these techniques can cover the only part of engine in X-window application.

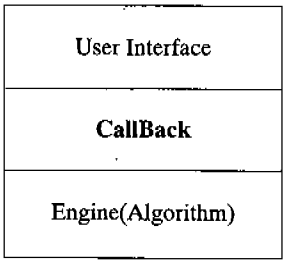

Fig. 1. The structure of X-window Application.

In practice, X-window application is programmed by X toolkits such as X intrinsics and widget sets [12]. X toolkit makes it simple to construct widgets such as buttons, dialogs, and menus by providing pre-defined functions. Each user interface object (or widget) is defined by a set of variables called resources. Usually a pre-defined function has a long complicated name and many parameters. It is not easy to define appropriate resources exactly since there are too many resources. In addition, the widgets follow the characteristics of an object in object-oriented approaches and the event occurred by the action of widget is similar to the event-driven approach. Hence, it is not easy to recognize the relationships among the user interface, the callback function and algorithm from the different points of view. The understanding of X-window application program is difficult as a program size becomes large.

In this paper, a new modeling tool for the use of understanding and development of GUI-based applications, so-called X-ADD (X-window application Analysis & Design Diagram) tool is introduced. The X-ADD automated tool which implements the X-ADD modeling technique supports re-engineering in X-window application software development from analysis to maintenance. The quick understanding of the existing X-window application program can be enhanced by using this X-ADD tool, i. e., an existing X-window program can be converted to X-ADD easily by applying this tool.

The outline of this paper is as follows. In chapter II, the modeling technique and symbols defined for this tool are introduced. The examples applying this X-ADD technique are discussed in chapter III. The features of X-ADD tool are described in chapter IV. Finally, the conclusion of this work is discussed in chapter V.

II. The X-ADD Modeling Technique

X-window programs like other ordinary application programs consist of a set of functions; some of which are defined to represent widgets or icons on the window and the others are used to perform general computing. It is possible that the whole program can be pictorially represented by a set of symbols, if each function has a corresponding pre-defined symbol. A set of symbols is a component of X-ADD. As X-window application

software is hierarchically constructed with three parts in Fig. 1, a set of symbols is hierarchically related with each other. Here, a set of symbols and their meanings are introduced in below.

1. Symbol of an invisible X functions

There are pre-defined X functions in X-window program whose execution does not affect to display somethings directly on the screen but they are necessary to build windows. The example functions include the composite functions in Xt intrinsics and the so-called container widget functions such as **XmCreateForm, XmCreateBulletinBoard** and **XmFrame** in Motif/X [10, 11]. Due to their property of invisibility on the screen, these functions might be easily ignored and bring the difficulty to find errors when there are errors. A symbol similar to a box is defined in Fig. 2 to represent such an invisible X function. In general, X functions consist of function variable name, X function name, and resource names. There are three sub parts in the box: The top row denotes a user-defined function variable, the middle row contains X function's pre-defined name such as Xlib functions, X widget functions, and X intrinsic functions and the bottom row has the resources such as the size and the position of a window, and etc.

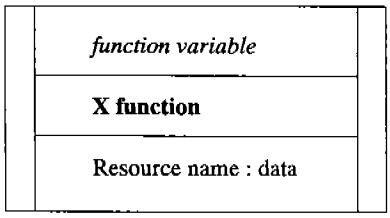

Fig. 2. The symbol of an invisible X function.

2. Symbol of a visible X functions

The visible X function means a function which directly displays something on the screen. In Motif/X, most widget functions such as **XmCreatePushButton, XmCreatePulldownMenu**, and **XmCreatePromptDialog** are included in this category of functions [10, 11]. A symbol representing a visible X function is defined in Fig. 3. There is no big difference with the symbol of an invisible X function except vertical boundary lines. The contained information is also the same as those of the symbol in Fig. 2.

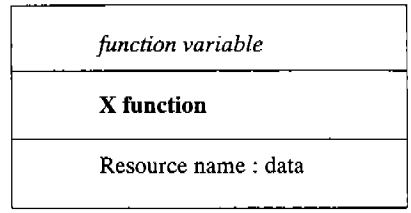

Fig. 3. The symbol of a visible X function.

3. Symbols of callback functions

A callback function is a part of the X-window application program which enables to communicate between the user interface and the computational part. Usually it is regarded as the most hardest part to manage the X-window program. Callbacks are usually written by attaching them to a specific widget event. The connection function, for example **XtAddCallback** [10, 11] enables to connect a visible X function and a callback function through parameters. A circle-like symbol in Fig. 4 is defined to represent the callback function. To make it simple, only the name of the callback function is specified in a circle. A double circle-like symbol is defined to represent a terminated callback function, which is a pre-defined function, for example **XtUnmanageChild** [10, 11] in Motif/X, and makes the specified widget disappeared from the screen.

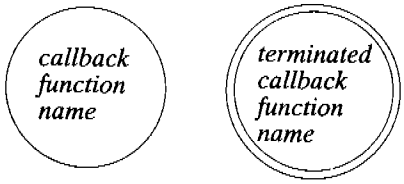

Fig. 4. The symbols of callback function.

4. Symbols of general function

A general function is an ordinary user-defined function which performs computing. We modify an ellipsoid to represent the symbol of this category function in Fig. 5. Only the name of the function is specified in the center of the ellipsoid like a symbol of the callback function in Fig. 4. The double lined ellipsoid represents a terminated general function which makes the specified widget removed from the window.

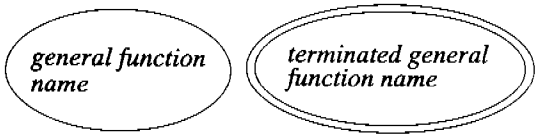

Fig. 5. The symbols of general function.

5. Symbol of a call relationship

An arrow with a parameter on the top is defined to specify the parameter passing in Fig. 6. It represents the relationship between a visible X function and a callback function, or between a callback function and a general function with client data. If the left hand side of an arrow is connected to the visible X function, the right hand side should be directed to a callback function. In case the left hand side of an arrow is connected to the callback function, the right hand side needs to be directed to a general function. If there is no client data on an arrow, it represents the connections between an invisible X function and a visible X function.

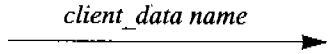

Fig. 6. The symbol of a call relationship.

6. Symbol of an assembly relationship

If one window contains a set of widgets, the relation of assembly of widgets is specified with a symbol in Fig. 7, which looks like a diamond. For instance, a container widget such as the bulletin board widget can contains several primitive widgets such as pushbutton, scalebutton, etc.

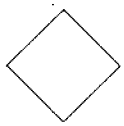

Fig. 7. The symbol of an assembly relationship.

7. Symbol of an either-or relationship

As discussed early, the callback function sets up the connection routine for an X-function and a non X-function. In that case, there must be several computing functions connected to the callback function through *client data* and only one function is selected at a given event. The symbol in Fig. 8 is defined to represent the either-or relationship.

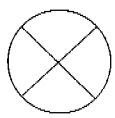

Fig. 8. The symbol of an either-or relationship.

III. Examples

In this chapter, we introduce two simple examples to model, and analyze the X-window application software by using of X-ADD technique. Through these examples, the relations between X-ADD symbols and the meanings of symbols can be clear.

The first example is shown for the understanding of the existing X-window application program. An application program in Appendix referred from [10] is converted to a diagram in Fig. 9 by X-ADD modeling technique. The description of this example is as follows :

- There is a push button named ' *Button* ' in a window.
- When the button is pushed, an event is occurred to created the message dialog that has two buttons named ' *OK* ' and ' *CANCEL* '.
- If the button ' *OK* ' is pushed, a message appears on the screen. If the button '*CANCEL* ' is pushed, the message dialog disappears.

In Fig. 9, the *toplevel* widget (invisible X function) contains a *button-widget* (visible X function) which is connected to *buttonCB* (callback function) whose client data is ' *OK* '. Consequently, *buttonCB* creates a message dialog widget (visible X function) which calls *dialogCB* (callback function). If the client data is ' *OK* ' , the function *A* (general function) executes and the message dialog disappears. If the client data is ' *CANCEL* ' , the message dialog disappears immediately. *NULL* means that there is no function. The symbol ① in Fig. 9 is defined to represent the connectivity of separated diagrams.

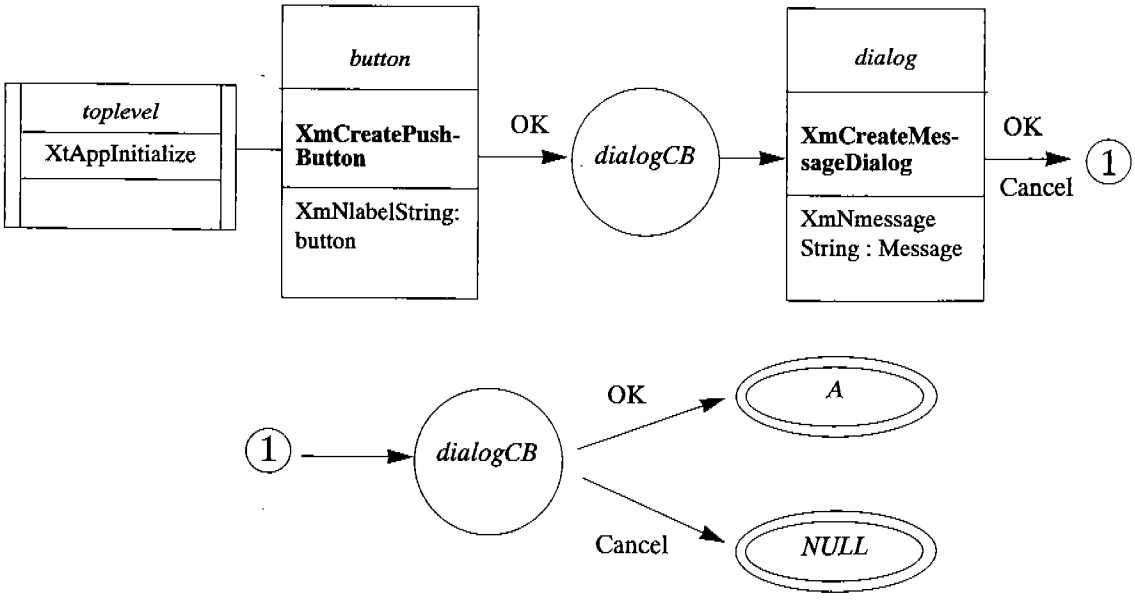

Fig. 9. The X-ADD of the first example.

The second example is to draw the X-ADD for the design of an X-window system. The system requirements of the second example are described as follows:

- A bulletin board contains several widgets such as one label widget, three button widgets, one text widget.
- Each button is connected to the specific function through a callback function, *doCB*.
- While the *do-button* widget is chosen, the function *do* executes, the function *help* begins to execute when the *help-button* widget is selected. Otherwise, the bulletin board disappears from the window, i.e., there is no corresponding function (*NULL*) when the *cancel-button* widget is pushed.

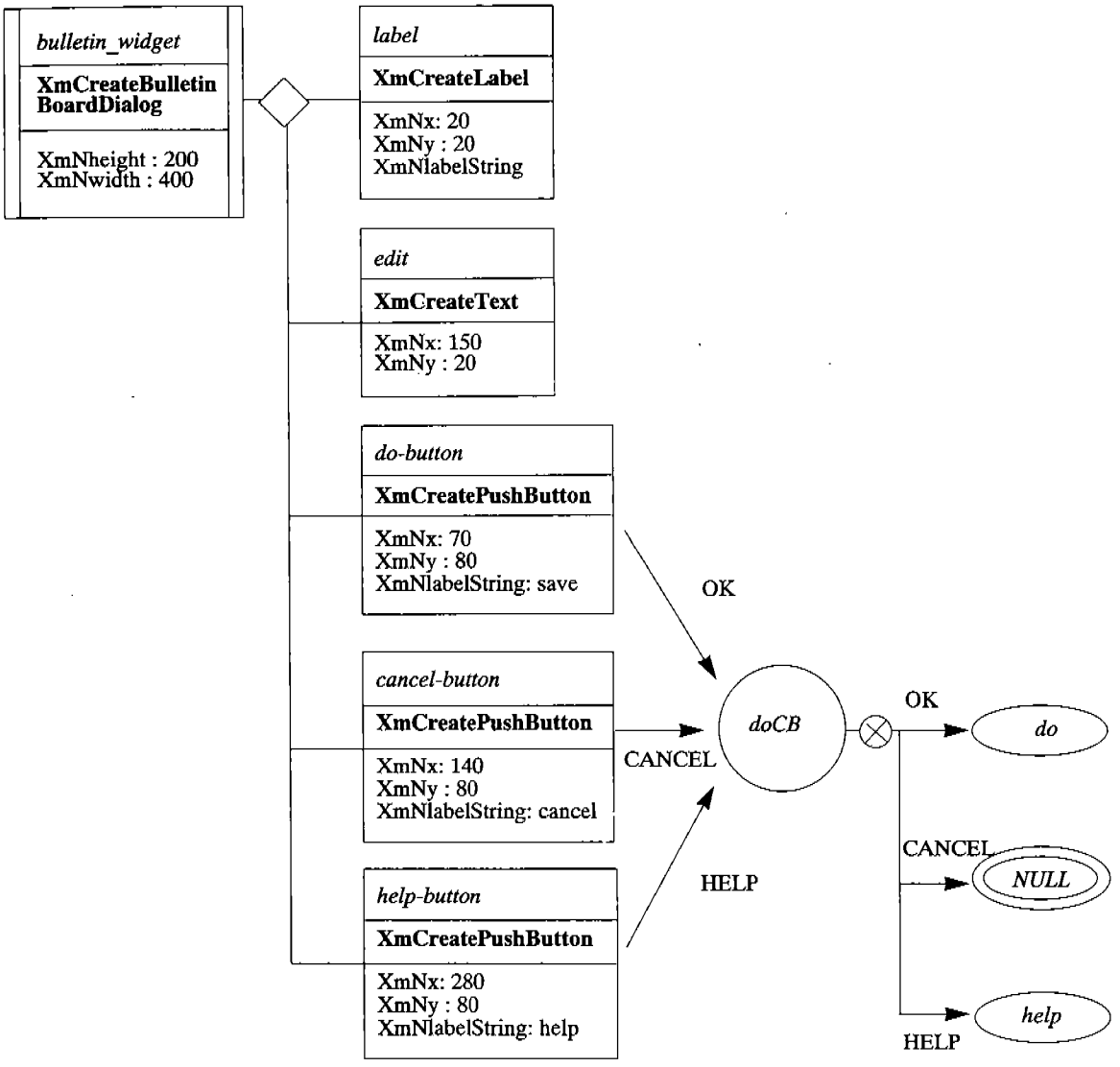

Fig. 10. The X-ADD of the second example.

The diagram in Fig. 10 represents this situation. In this figure, *bulletin-widget* (invisible X function) contains one label widget, one text widget and three push button widgets. All of these button widgets call the *doCB* (callback function), but each of them has different client data. If the client data is ' *OK* ', *do* (general function) executes. If the client data ' *CANCEL* ' is chosen, the *bulletin-widget* board disappears immediately without any execution. If ' *HELP* ' is selected, *help* (general function) executes.

IV. The Features of X-ADD Tool

The various features of X-ADD CASE tool that implements the modeling and analyzing of the X-window application program are introduced in this chapter. The features of X-ADD tool are summarized as follows:

- This tool has a graphical editor which handles the creation and revision of various symbols (X function, callback function, etc) with ease.
- Callback functions and general functions are

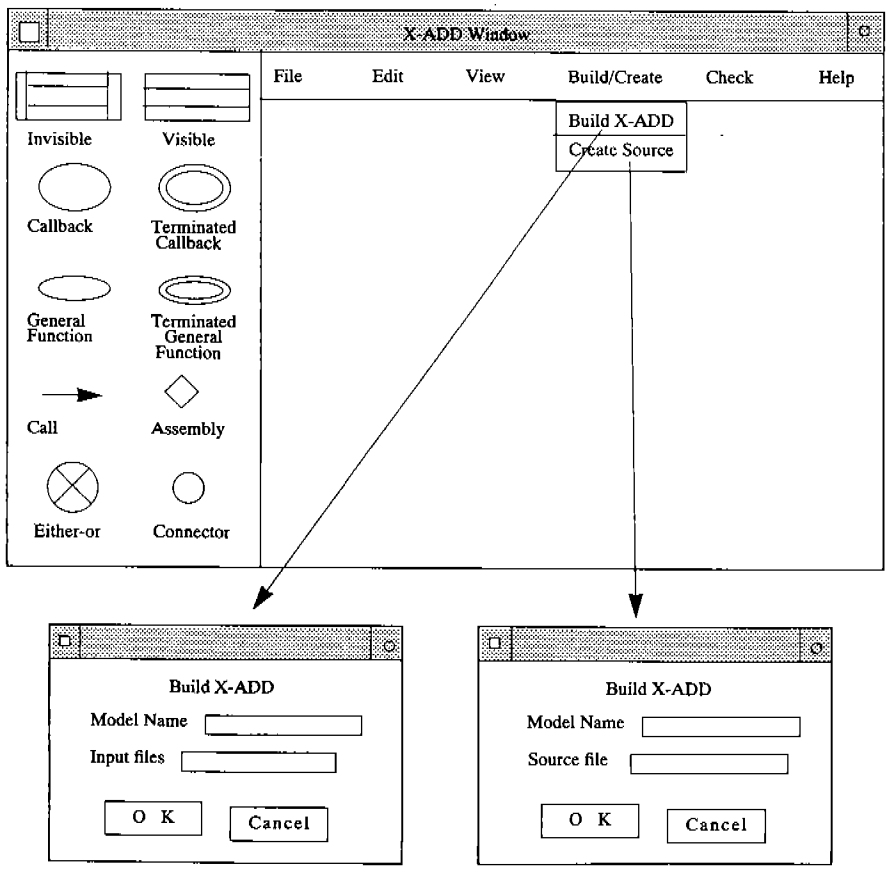

Fig. 11. The windows of X-ADD tool.

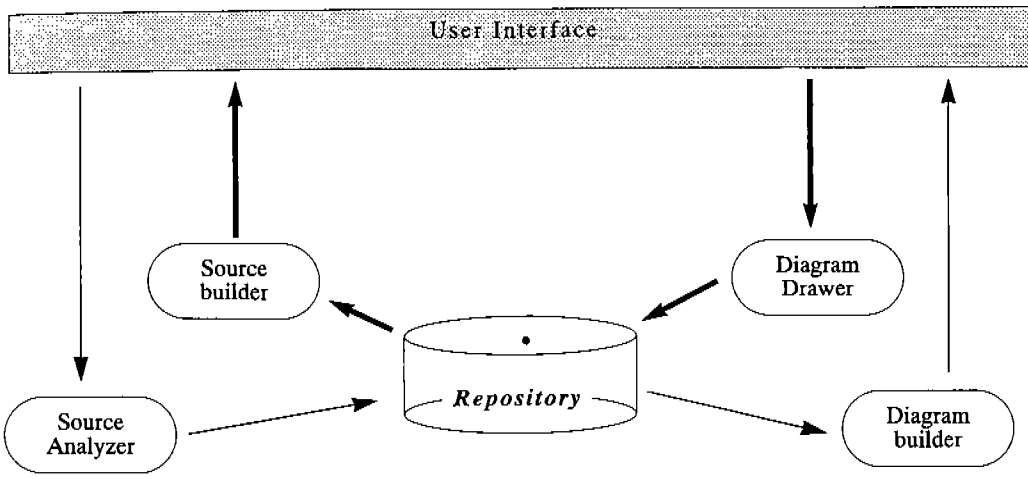

Fig. 12. X-ADD tool system.

registered in and retrieved from the predefined library.
- The drawn X-ADD can be converted to codes of a program description language easily.
- The existing X-window program can be converted to an X-ADD automatically.
- This tool enables to check completeness and correctness of the development system using defined rules.

This tool includes various sub-internal tools, such as graphical editor, source builder, source analyzer, diagram builder, and so on. While Fig. 11 represents the part of user interface windows in X-ADD tool. Fig. 12 describes the X-ADD tool system and describes the relation among the internal tools. The input diagram of the diagram drawer is parsed and stored in the repository, and this parsed information of diagram is created to an X-window based program by the source builder. Also, the information of existing X-window application program is stored into the repository in the form of widgets (libraries) and stored information is shown in user interface window by the diagram builder.

V. Conclusions

In general, there is a lack of modeling technique in the understanding and development of X-window application software. We have presented a modeling tool (technique) which helps the user to understand the existing X-window application programs, analyze and design the new X-window application software. Primarily, this tool supports for the general forward engineering of new X-window application software development from the analysis to the design, and maintenance. In addition, the use of this tool in the reverse engineering enhances the visibility of X-window application i.e., this X-ADD modeling tool helps to understand the existing an X-window application program by converting it to the diagram. The major benefit of this diagram technique is that it is very simple, and easy to apply to the real systems. We

are now working on the development of X-ADD CASE (Computer-Aided Software Engineering) tool which helps to draw X-ADD diagram easily, and convert X-window program to X-ADD diagram automatically, and vice versa. Moreover, this tool has the facility to check the completeness and correctness of an X-ADD diagram with defined rules.

[Appendix]

```
/* message.c */

#include <Xm/Xm.h>
#include <Xm/PushB.h>
#include <Xm/MessageB.h>
#define OK 1
#define CANCEL 2
XtAppContext context;
XmStringCharSet char_set = XmSTRING_
   DEFAULT_CHARSET;
Widget toplevel, button, dialog;
void dialogCB(w, client_data, call_data)
        Widget w;
        int client_data;
        XmAnyCallbackStruct *call_data;
{
        switch(client_data)
                {
                  case OK:
                       printf("OK selected\n");
                       break;
                  case CANCEL:
                       break;
                }
        XtUnmanageChild(w);
}
void buttonCB(w, client_data, call_data)
        Widget w;
        XtPointer client_data;
        XmPushButtonCallbackStruct*call_data;
{
        XtmanageChild(dialog);
}
```

```
void  main(argc, argv)
                int argc;
                char *argv[];
{
        Arg al[10];
        int ac;
        toplevel= XtAppInitialize(&context, "",
          NULL,0,&argc,argv,NULL, NULL,0);
        ac=0;
        XtSetArg(al[ac], XmNmessageString,
                 XmStringCreateLtoR(" Mess-
                    age ", char_set));ac++;
        dialog = XmCreateMessageDialog(top-
          level,"dialog",al,ac);
        XtAddCallback(dialog, XmNokCallback,
          dialogCB, OK);
        XtAddCallback(dialog, XmNcancelCall
          back, dialogCB, CANCEL);
        XtUnmanageChild(XmMessageBoxGet-
          Child(dialog, XmDIALOG_HELP_
          BUTTON);
        ac=0;
        XtSetArg(al[ac], XmNlabelString,
                 XmStringCreate ("Button ",char
                   _set)); ac++;
        button = XmCreatePushButton(toplevel,
          "button",al,ac);
        XtManageChild(button);
        XtAddCallback(button, XmNactivateCa-
          llback, buttonCB, OK);
        XtRealizeWidget(toplevel);
        XtAppMainLoop(context);
}
```

References

[1] James C. Armstrong Jr., " Six GUI builders face off ", *SunWorld*, 1992. 12.

[2] James C. Armstrong Jr., " ABCs of Programming on X", *SunWorld*, 1992.12.

[3] Richard E. Fairley, *Software Engineering Concepts,* McGraw-Hill, 1985.

[4] James Rumbaugh et al, *Object-Oriented Modeling and Design* , Prentice-Hall, 1991.

[5] Y. F. Chen, M. Y. Nishimoto, "The C Information Abstraction System," *IEEE Transactions on Software Engineering,* Vol. 16, No. 3, 1990.

[6] L. Clevelands, "A Program Understanding Support Environment," *IBM Systems Journal,* Vol. 28, No. 2, 1989.

[7] T. A. Corbi, "Program Understanding Challenge for the 1990s," *IBM Systems Journal,* Vol. 28, No. 2, 1989.

[8] A. R. Henver, R. C. Linger, "A Method for Data Reengineering in Structured Programs," *Proc. 22nd Annual Hawaii Int'l Conference of Systems Science, 1989.*

[9] D. J. Robson, H. H. Bennett, B. J. Cornelius, and M. Munyo, "An Approaches to Program Comprehension," *Journal of Systems and Software,* Vol. 14, Feb. 1991.

[10] Marshal Brain, *Motif Programming.* Digital Press, 1992.

[11] Donald L. McMinds, *Mastering OSF/Motif Widgets*, HP Press, 1992.

[12] Douglas A. Young, *The X Window System Programming and Applications With Xt OSF/Motif Edition,* Prentice-Hall, 1990.

[13] T. J. Biggerstaff, "Design recovery for maintenance and Reuse," *IEEE Computer*, July 1989.

Joon Seok Lee received the B.S. degree in industrial engineering from HanYang University, in 1985, the M.S. degree in industrial engineering from Korea Advanced Institute of Science and Technology(KAIST), in 1990. He is a researcher in the System Engineering Section at ETRI, where his research interests are in the configuration management, the modeling of software system, and the system development methodology.

Sang Bum Lee received the B.S. degree in mechanical engineering form HanYang University, in 1983, and the M.S. and the Ph.D. degrees in computer science from Louisiana State University, in 1989, 1992. He is a senior researcher in the System Engineering Section at ETRI. His research areas are software engineering, knowledge-base system, and system modeling.

Dong Hae Chi received the B.S. degree in chemical engineering from KyungHee University, in 1982, and the M.S. and the Ph.D. degrees in industrial engineering from Iowa State University, in 1985 and 1989, respectively. Dr. Chi is currently a head of the System Engineering Section at ETRI and an Associate Professor of Computer Engineering at Chungnam National University. His research interests include software reliability, object-oriented development methodology, and programming environment.