



Visual Motif Code Generator
by Xiao Pu

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in
Computer Science
Montana State University
© Copyright by Xiao Pu (1995)

Abstract:

VMCG is the Visual Motif Code Generator, which is used to generate OSF/Motif C source code visually. Here VMCG uses a subset of OSF/Motif widgets, including representative Shell, Manager and Primitive widgets. VMCG also supplies widget hierarchy display, widget geometry management over its parent manager widget, widget resource editor and widget callback editor. Future directions for VMCG are discussed.

**VISUAL MOTIF
CODE GENERATOR**

by
Xiao Pu

A thesis submitted in partial fulfillment
of the requirements for the degree
of
Master of Science
in
Computer Science

MONTANA STATE UNIVERSITY
Bozeman, Montana

May, 1995

N378
P961

ii

APPROVAL

of a thesis submitted by

Xiao Pu

This thesis has been read by each member of the thesis committee and has been found to be satisfactory regarding content, English usage, format, citations, bibliographic style, and consistency, and is ready for submission to the College of Graduate Studies.

4/20/95
Date

J. D. Stanley
Chairperson, Graduate Committee

Approved for the Major Department

4/20/95
Date

J. D. Stanley
Head, Major Department

Approved for the College of Graduate Studies

5/3/95
Date

R. Brown
Graduate Dean

STATEMENT OF PERMISSION TO USE

In presenting this thesis in partial fulfillment of the requirements for a master's degree at Montana State University, I agree that the Library shall make it available to borrowers under rules of the Library.

If I have indicated my intention to copyright this thesis by including a copyright notice page, copying is allowable only for scholarly purposes, consistent with "fair use" as prescribed in the U.S. Copyright Law. Requests for permission for extended quotation from or reproduction of this thesis in whole or in parts may be granted only by the copyright holder.

Signature Xiao Pu
Date 4/20/95

TABLE OF CONTENTS

| | Page |
|--|------|
| APPROVAL PAGE | ii |
| STATEMENT OF PERMISSION TO USE | iii |
| TABLE OF CONTENTS | iv |
| LIST OF FIGURES | vi |
| ABSTRACT | vii |
| INTRODUCTION | 1 |
| X/Motif | 1 |
| Visual Motif Code Generator | 5 |
| VISUAL OSF/MOTIF GUI BUILDER DESIGN | 9 |
| Tree Widget Hierarchy | 9 |
| WYSIWYG Design | 9 |
| VISUAL RESOURCE BUILDER | 17 |
| Motif Widget Resources | 17 |
| Widget Resource Editor | 17 |
| Widget Callback Editor | 18 |
| VMCG USER INTERFACE | 22 |
| File | 22 |
| Run | 22 |
| Application | 22 |
| Compile | 22 |
| Widget | 22 |

TABLE OF CONTENTS - Continued

| | Page |
|--|-------------|
| Edit | 22 |
| CONCLUSIONS AND FUTURE DIRECTIONS | 25 |
| REFERENCES | 26 |

List of Figures

| | | |
|----|--|----|
| 1 | A Typical X Windows System Screen | 3 |
| 2 | X Windows System Hierarchy | 3 |
| 3 | Motif Class Hierarchy | 4 |
| 4 | A Label Widget | 7 |
| 5 | Examples of PushButton Widgets | 7 |
| 6 | A DrawingArea Widget | 8 |
| 7 | An MSU Student ID designed by VMCG | 8 |
| 8 | Tree Widget Hierarchy Interface | 12 |
| 9 | WYSIWYG Design Interface | 13 |
| 10 | Tree for Activated Widget Label1 | 13 |
| 11 | Activated Label Widget | 14 |
| 12 | Tree for Activated Widget Push1 | 14 |
| 13 | Moving A Widget | 15 |
| 14 | After the Moving | 15 |
| 15 | Resizing A Widget | 16 |
| 16 | After the Resizing | 16 |
| 17 | RGB Color Selector | 19 |
| 18 | List Color Selector | 19 |
| 19 | Font Selector | 20 |
| 20 | String Selector | 20 |
| 21 | Widget Callback Editor | 21 |
| 22 | VMCG MainMenu | 24 |
| 23 | VMCG File submenu | 24 |
| 24 | VMCG Exit DialogShell | 24 |

ABSTRACT

VMCG is the Visual Motif Code Generator, which is used to generate OSF/Motif C source code visually. Here VMCG uses a subset of OSF/Motif widgets, including representative Shell, Manager and Primitive widgets. VMCG also supplies widget hierarchy display, widget geometry management over its parent manager widget, widget resource editor and widget callback editor. Future directions for VMCG are discussed.

INTRODUCTION

X/Motif

The X Window System (or simply X) is a hardware- and operating system-independent windowing system. It was developed jointly by MIT and Digital Equipment Corporation, and has been adopted by the computer industry as a standard for graphics applications.

X provides a root window within which you can display smaller windows. You can run a number of applications simultaneously and each application may have any number of windows.

Figure 1 shows a workstation screen with a typical assortment of windows. The root window is the large window that contains all other windows.

X is composed of, among other things, a set of library functions. These functions are normally referred to collectively as Xlib. Xlib is the heart of X and it consists of a large number of C library functions. Unfortunately, programming using only Xlib can be tedious, so the X designers created a second set of functions that are collectively referred to as the Xt Intrinsics or the X Toolkit. The Xt Intrinsics provide a higher-level set of functions that make programming easier.

Although easier to use than Xlib, the Xt Intrinsics can also be tedious and cumbersome to use, so widgets and gadgets were designed to utilize both Xlib and Xt Intrinsics functions and relieve the programmer of much of the dirty work. A widget is a user interface component composed of data structures and procedures, which usually has its own window. A gadget is a widget that does not have a window.

Figure 2 shows the relationship among OSF/Motif, Xt Intrinsics, Xlib and operating system.

The Motif widgets and gadgets were designed under the principles of object-oriented programming. This means that they are organized in sets of different classes. A class is a set of objects with similar characteristics. One of the most important aspects of object-oriented programming is that of inheritance. Any class can inherit characteristics from higher class. For example, the PushButton widget can inherit characteristics of the Label widget, the Primitive widget, and the Core widget. Primitive and Core widgets are considered superclass widgets.

A gadget does not have a window of its own, but must display its contents in its parent's window. Gadgets are, therefore, more efficient.

Motif's object-oriented architecture groups widgets and gadgets into different classes. Each class has data structures and procedures that operate on the data. The classes are structured in a hierarchy, with the superclasses toward the top of the hierarchy.

Figure 3 shows the Motif and Xt widgets Class Hierarchy. Subclasses, to the right, inherit features from their superclasses, to the left.

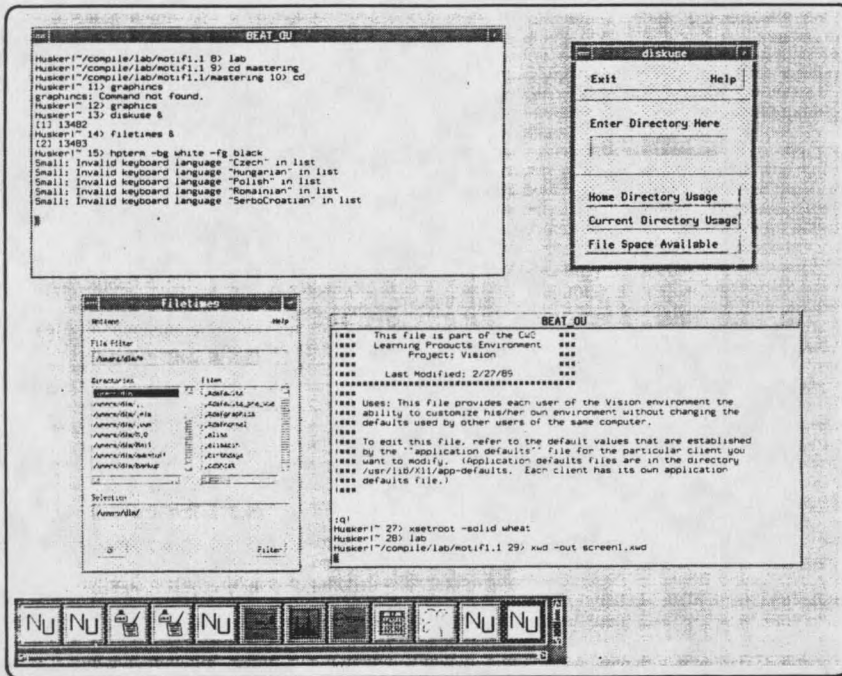


Figure 1: A Typical X Windows System Screen

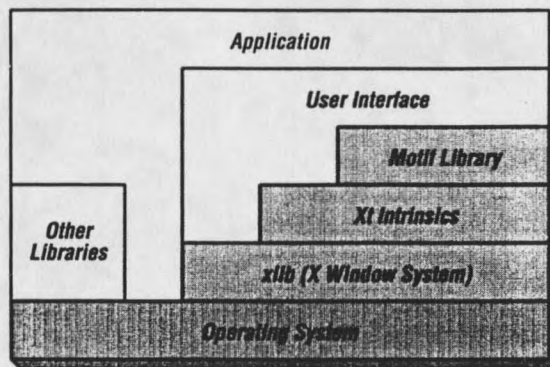


Figure 2: X Windows System Hierarchy

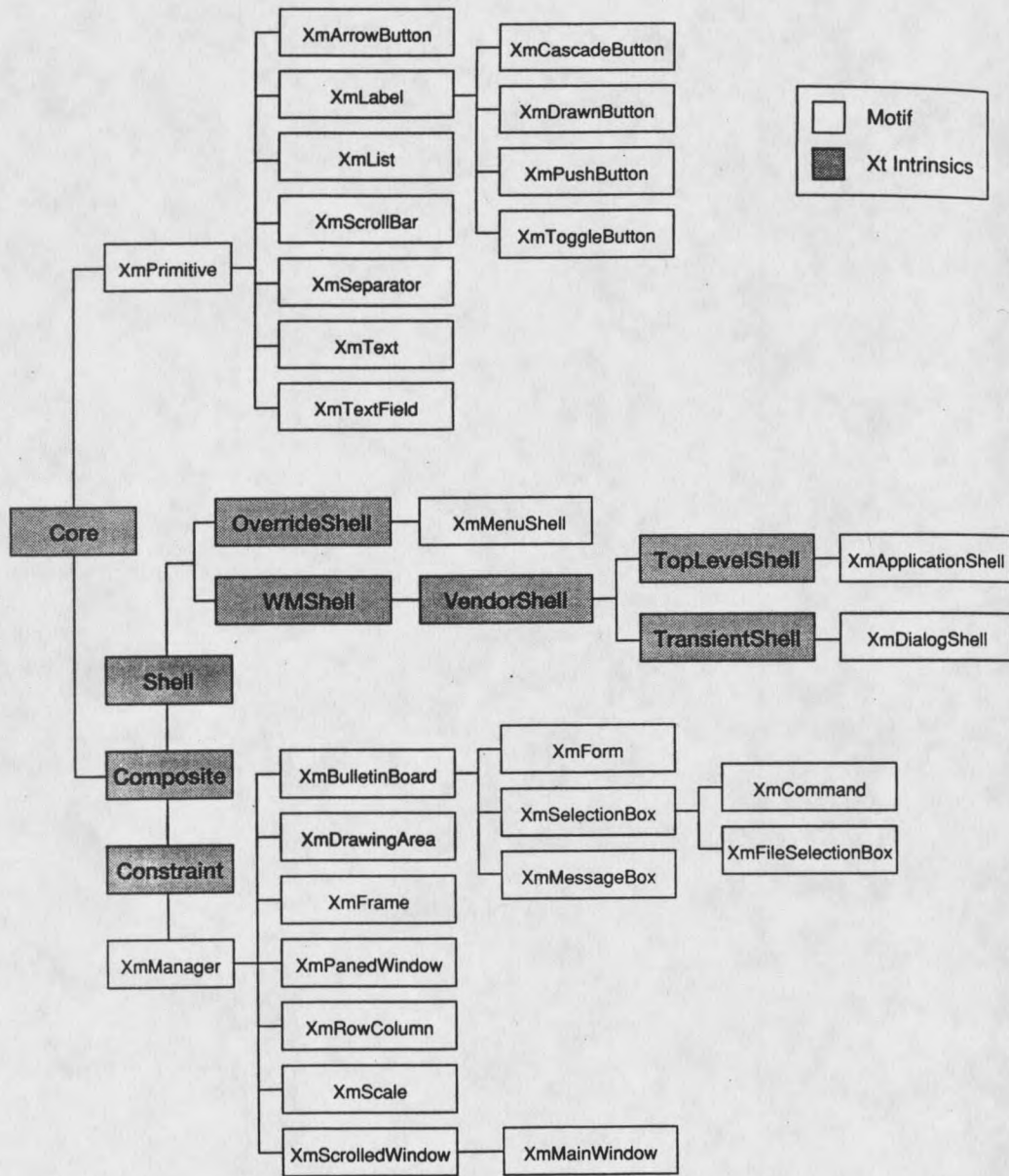


Figure 3: Motif Class Hierarchy

VISUAL MOTIF CODE GENERATOR

Developers have long known that the sophistication and power of their X/Motif GUI designs are restricted by the complexity of X/Motif unless they make available the many hours that it takes to create widgets. Here, we develop a Visual Motif Code Generator (VMCG) to give an easy and intuitive way to design X/Motif GUIs.

When we develop our X/Motif application, we need to know the hierarchy of the widgets used in our development intuitively. The best way is to supply a tree hierarchy for it; the second important thing is the geometry management for the child widget over its parent widget related to the issue of position and size; the third thing is to edit resource for the widgets; the fourth is to give C code for developers to enhance it.

In VMCG, we give a solution for the above issue by using Application Shell widgets, Form widgets, PushButton widgets, Label widgets and DrawingArea widgets, which represent the Shell, Manager (also known as the Container) and Primitive widgets in X/Motif widgets.

The Label Widget provides a visual label as text (in any font or using multiple fonts) or as an image (in the form of a Pixmap). Its text is a compound string and can be oriented from left-to-right or right-to-left. Compound strings can also be multilined, multifont or any combination of these. Basically, the Label widget is only intended to be used to display labels or other visual aids. Figure 4 shows a Label widget with three different fonts and lines in a compound string.

The PushButton widget supports the same visual display capabilities as Labels, since it is subclassed from them. However, PushButton also provides resources for the programmer to install callback routines that will be called when the user "activates" the button (clicks on it). Additionally, the PushButton displays a shadow border that changes in appearance to indicate when the pointer is in the widget, and when it has been activated. When

the `PushButton` is not selected, it appears to project out towards the user. The button's border is highlighted when the pointer moves into the button; in this state, the button is said to be "armed". When the user actually selects the button by pressing the pointer on the armed button, it appears to be "pushed in". The user actually "activates" a `PushButton` by releasing the mouse button while the button is in a "selected" state. Figure 5 shows examples of `PushButton` widgets.

The `Form` widget is one of the `Manager` widgets which are used to handle the placement of multiple widgets in a single window. `Form` supports the capabilities of the class by introducing a sophisticated geometry management policy that involves both absolute and relative positioning and sizing of its children. For example, `Forms` may lay out their children in a grid-like manner, anchoring edges of each child to specific positions on the grid, or they may attach them to one another in a chain-like fashion. Figure 5 also shows a `form` widget with three `PushButton` widgets on it.

The `DrawingArea` widget is a widget that behaves like a drawing surface for X. Once you have established a `DrawingArea`, you can use any of the X `Drawing` functions to draw it. The X `Drawing` functions allow you to draw points, lines, rectangles, arcs, and so on. As with any other widgets, you can attach a `DrawingArea` widget to a form, make it sensitive and insensitive, resize it, and so on. Figure 6 shows a `DrawingArea` widget with a mouse signature on it.

The `ApplicationShell` widget is used to create an entirely new `Shell` from within an application. That is, you might want to call some code and have it create an entirely new and separate `Shell` into which you can add widgets. This new `Shell` has all of the attributes of the toplevel `Shell` and fulfills the same function. It can contain anything you would normally put in a toplevel `Shell`: any widget, form, and so on. The new `Shell` is a complete window, and you can resize it, minimize it, and maximize it just like any other window.

In Figure 8 the Tree Hierarchy Interface is a ApplicationShell from VMCG MainMenu's toplevel Shell.

Using VMCG, we can develop our GUI designs visually and intuitively. We just select the widget type we need and add it to the application hierarchy, also we can delete some unneeded widgets from the hierarchy, design their geometry management by mouse movement, edit their resources, add some callback routines, and finally we can get the needed C source code. For example, Figure 7 shows an MSU student ID form GUI developed by VMCG.

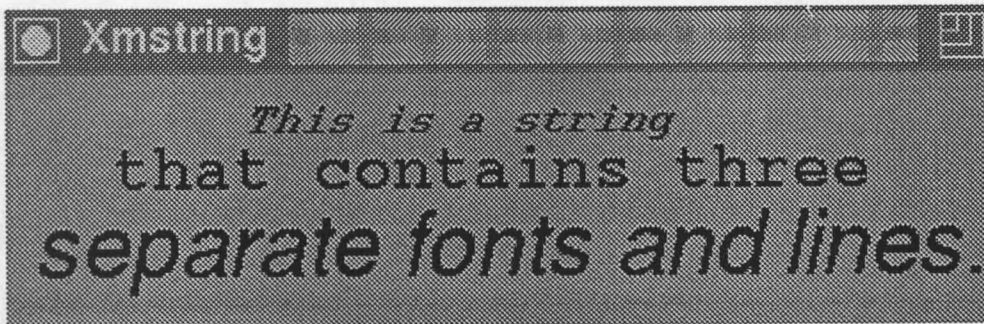


Figure 4: A Label Widget

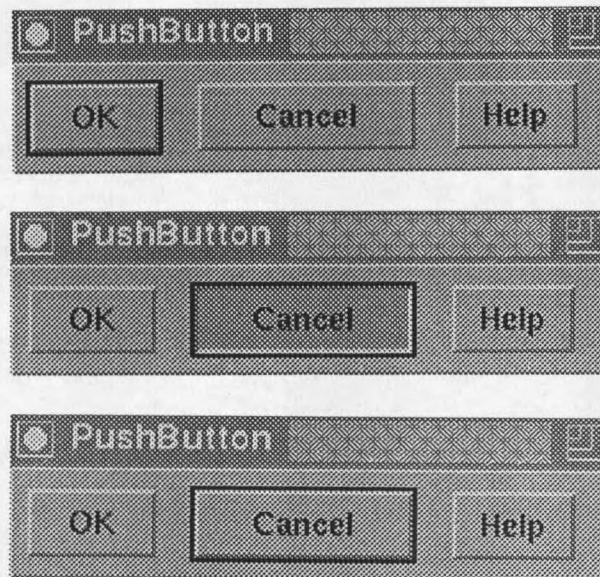


Figure 5: Examples of PushButton Widgets

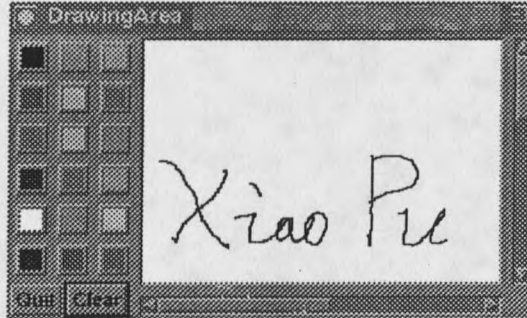


Figure 6: A DrawingArea Widget

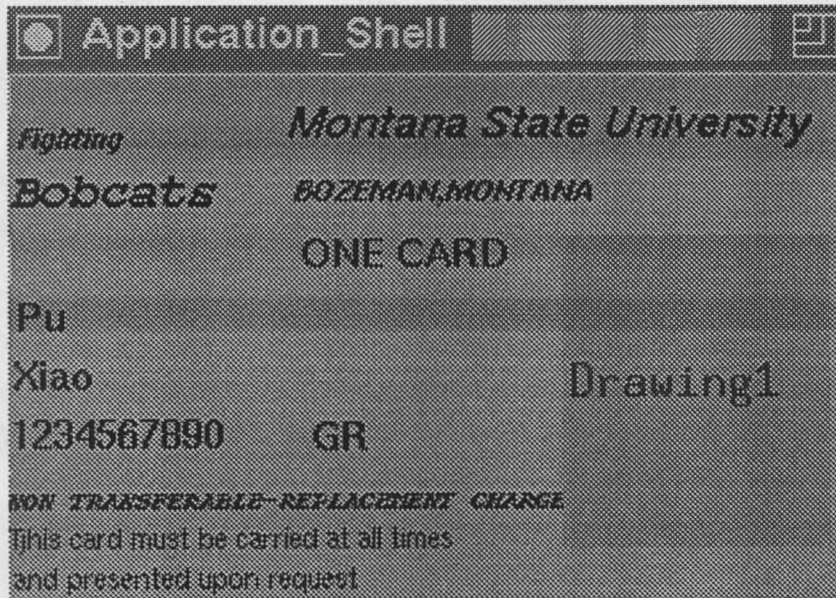


Figure 7: An MSU Student ID designed by VMCG

VISUAL GUI BUILDER DESIGN

Tree Widget Hierarchy Interface

The Tree Widget Hierarchy Interface displays the application widget hierarchy by a tree. Every node is a box, enclosing the widget name. The widget name is named for the widget type plus the serial number (1, 2, ...), such as Push1, Label1, Shell1, Drawing1, Form1, Push2, and so on. The widget is activated when user clicks inside the box containing the widget name. Not Active widgets will be displayed in black color, the active widget will be displayed in red color.

A current active object is served as the parent widget of any new widgets that are created based on it. For example, a Manager is served as the parent widget for some Primitive widgets on it; a Shell widget may be served as the parent widget for a Manager widget; a Primitive widget may be served as the parent widget for a Shell widget or a Dialog widget.

Figure 8 shows an example of the Tree Widget Hierarchy Interface, the interface is a scrolled DrawingArea, so user can scroll it when needed.

WYSIWYG Design Interface

The Application Design Interface is a what-you-see-is-what-you-get (WYSIWYG) interface. That means it reflects every change in our design, such as adding and deleting widgets, editing widget resources. Basically, it's a single window world unless we add a Shell widget to our application hierarchy, that will add one more window for the Application Design Interface (unless the user adds a Shell widget to a Shell parent, which is not encouraged). Finally, the application generated by VMCG generated C source code is exactly the same as that produced by the Application Design Interface.

Figure 9 shows an example of an Application Design Interface whose widget hierarchy is displayed in Figure 8.

Another important function for the WYSIWYG Design Interface is to implement Geometry Management for the widgets.

- **Widget Activated status**

When the user clicks inside the box containing the widget name in the Tree Widget Hierarchy Interface, the cursor becomes a hand cursor, the box of that widget in Tree Hierarchy becomes red while other boxes still remain black, a box appears in the real widget in the Application Design Interface with the same width and height as the real widget, and there is a small box in the right-up corner of the box with the 20 percent size of the box; this small box is served as resize box.

Figure 10 shows the Tree Interface after clicking inside the box of Label1. The Widget Label1 is activated and that box and the string Label1 change to red. Figure 11 shows a box that appears in the real widget of Label1, reflecting the real position and size of widget Label1.

- **Move widget**

After the widget is activated, the rubber-band box appears in the Application Design Interface and the cursor becomes a hand cursor, the user can press mouse button1 in box and not in resize box, and move the widget to the desired place in the Application Design Interface, then release the the mouse button1. The VMCG will move the widget to the new place in its parent container widget.

Figure 12 shows widget Push1 activated. Push1 in Tree Hierarchy is red, its XmString is OK in real widget. Figure 13 shows moving the widget. Figure 14 shows the moved widget.

- **Resize widget**

After the widget is activated and the rubber-band box has appeared in the Application Design Interface and the cursor becomes a hand cursor, the user can press mouse button1 in resize box, move the mouse and the user can get a new rubber-band drawing box illustrating the new size for the active widget. The box's right-up corner is changed by the cursor place of the mouse while the left-bottom corner place of the box is fixed. Move it to the desired place in the Application Design Interface and release the mouse button1. The active widget gets a new size which is the same as the new box, now, the resize action for the active widget is realized. The user can get the needed size and position for the child widget over its parent container widget by activating the child widget and move or resize it repeatedly.

Firstly, activating widget Push1, see Figure 12. Figure 15 shows resizing the widget. Figure 16 shows the resized widget Push1.

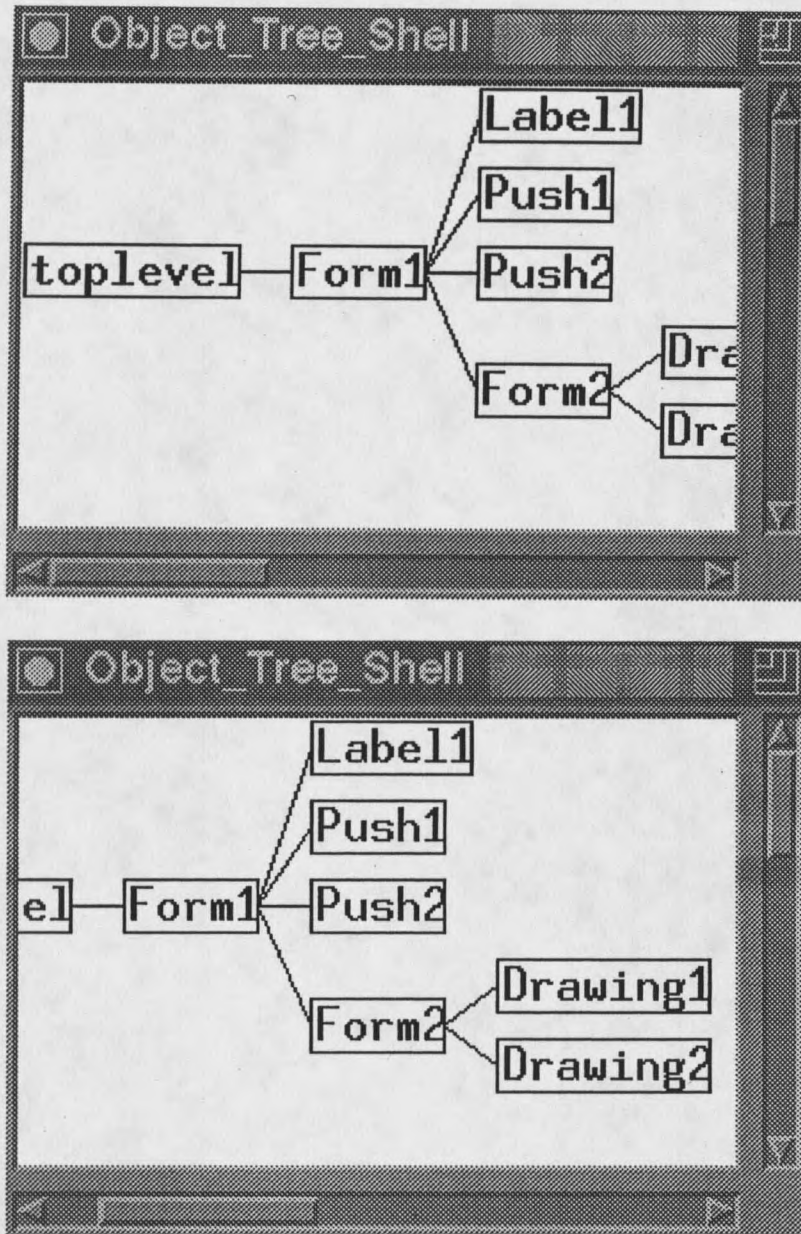


Figure 8: Tree Widget Hierarchy Interface

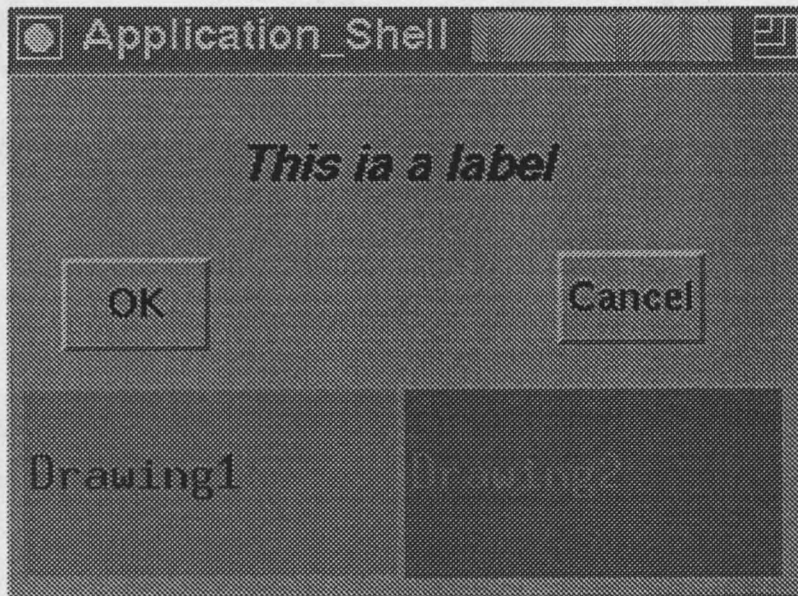


Figure 9: WYSIWYG Design Interface

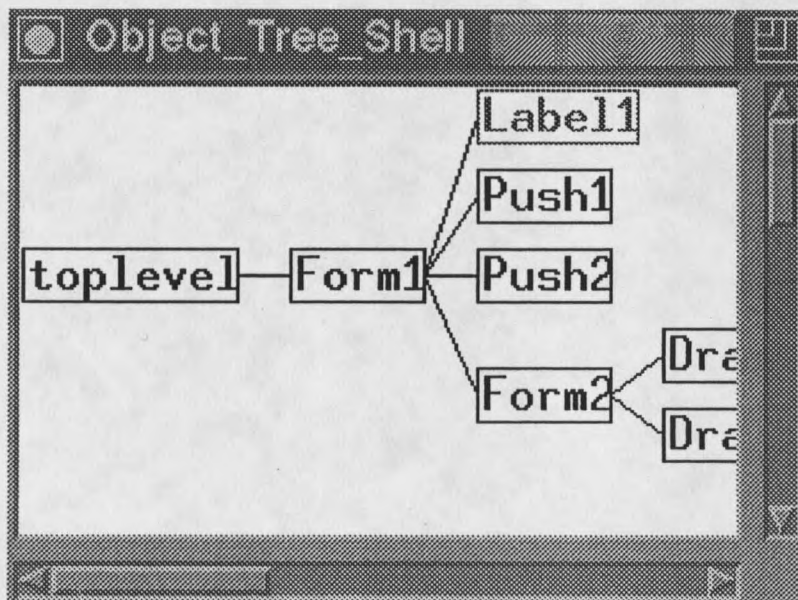


Figure 10: Tree for Activated Widget Label1

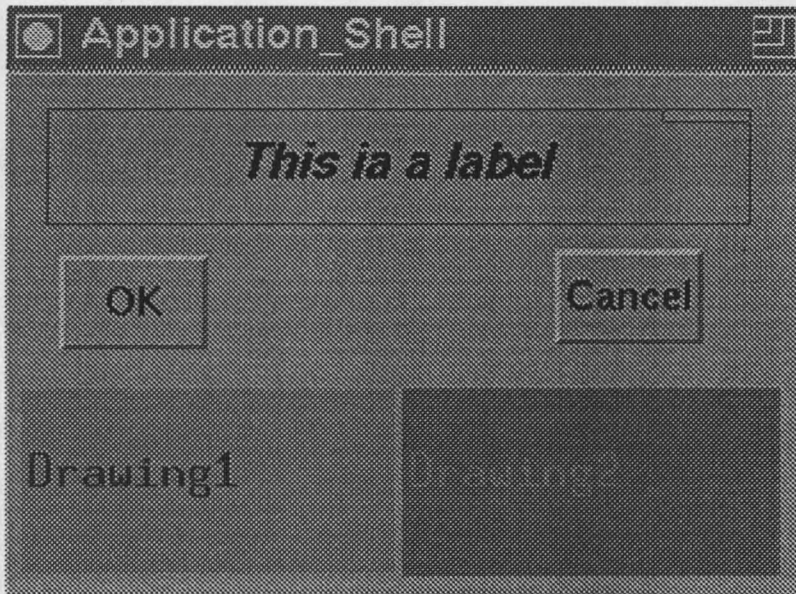


Figure 11: Activated Label Widget

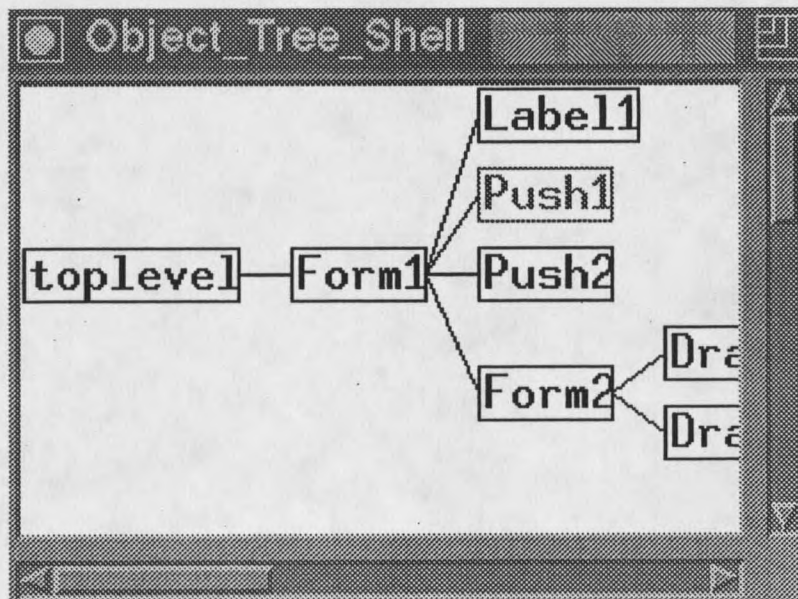


Figure 12: Tree for Activated Widget Push1

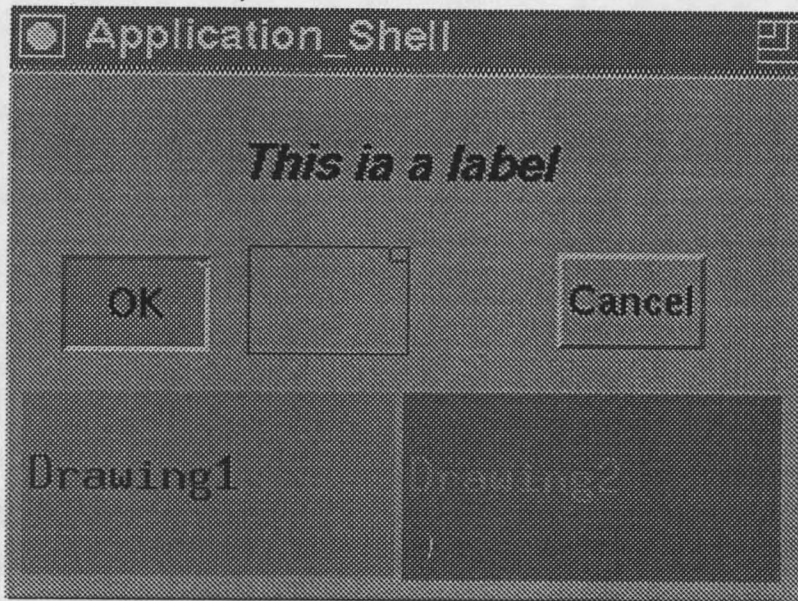


Figure 13: Moving A Widget

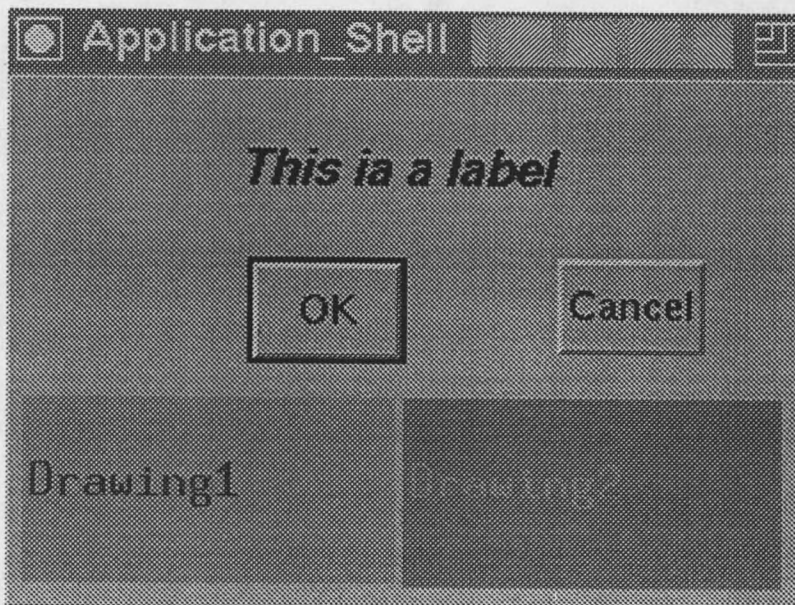


Figure 14: After the Moving

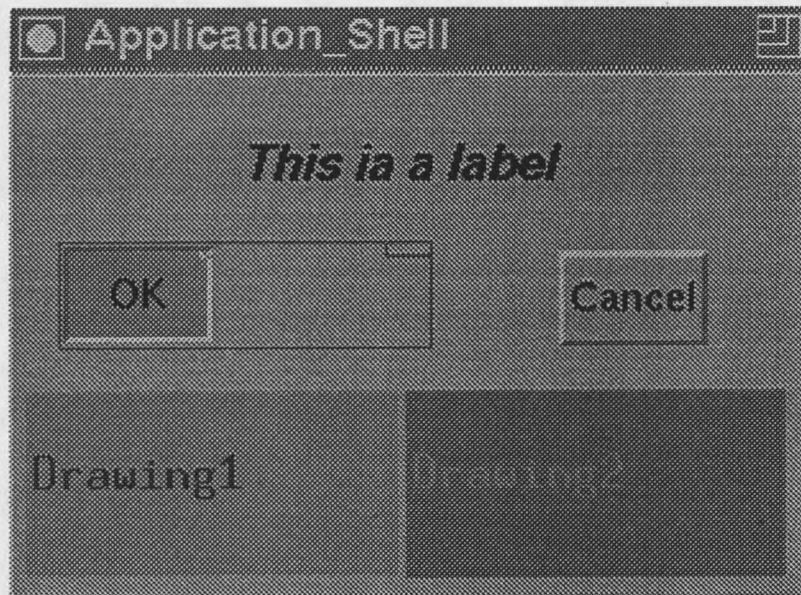


Figure 15: Resizing A Widget

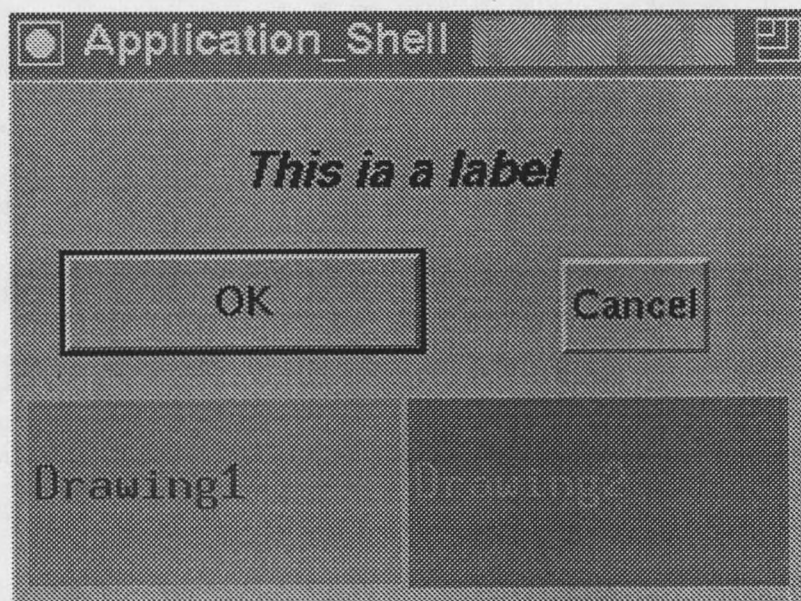


Figure 16: After the Resizing

VISUAL RESOURCE BUILDER

Motif Widget Resources

One of the keys to understanding Motif programming is understanding the concept of resources. The programmer designs a Motif application by selecting a set of widgets to compose the user interface. Every widget, in turn, has a set of associated resources that control its appearance and behavior. Resources are much like normal variables, except that you must access them in a special way. For example, a label widget has resources that determine such features as the string displayed by the label, the font used to display the string, and the margins around the string. These resources can be read or set to new values. The programmer changes the widgets in the user interface by adjusting resource values.

Widget Resource Editor

- **RGB Color Resource Editor** : The RGB Color Resource Editor lets the user specify a color by using a set of RGB sliders. The RGB value of the selected color is dynamically displayed. Sliders let the red, green and blue values be changed so the user can select any color available on the system. Figure 17 shows the RGB Color Resource Editor.
- **List Color Resource Editor** : The List Color Resource Editor lets the user directly access an alphabetically-sorted list of hundreds of the most common colors. Figure 18 shows the List Color Resource Editor.
- **Font Resource Editor** : The Font Resource Editor lets the user specify a font by selecting the font family and directly access the font name list in that family. On the top of Font Selector, there are font family name and two arrow buttons allowing the user to specify the font family by toggling the family names. On the bottom, there are

detailed font name and two arrow buttons allowing the user to specify the exact font name in that font family by toggling the font names in that font family. In the middle, there is a WYSIWYG dynamic displaying for the font name. Figure 19 shows the Font Resource Editor.

- **String Resource Editor** : The String Resource Editor lets the user specify an XmString resource for the Label and PushButton widgets. Figure 20 shows the String Resource Editor.

Widget Callback Editor

All Motif widgets have callbacks, which they use to trigger specific actions in response to user events. The idea behind callbacks is extremely straightforward. If a user manipulates a widget on-screen, something needs to notify the program of the change. For example, if an application displays a PushButton widget, the user will eventually click the PushButton and expects some specific actions to result. For example, the user will expect the program to quit after clicking the button labeled "quit". The program needs to know about the click so that it can generate the appropriate action.

The Widget Callback Editor in VMCG lets the user specify the callback action, also the user can enhance the callback action when he/she edits the VMCG generated C source code. Figure 21 shows the Callback Editor.

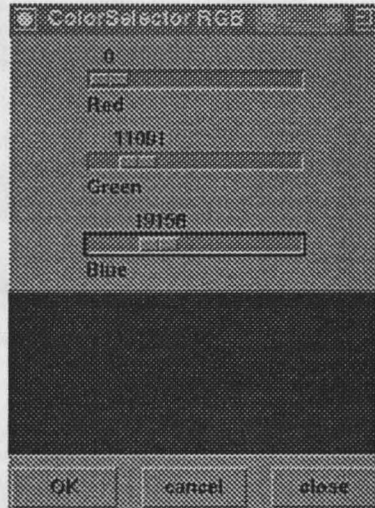


Figure 17: RGB Color Selector

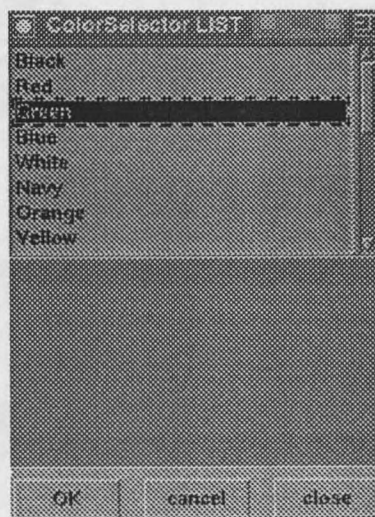


Figure 18: List Color Selector

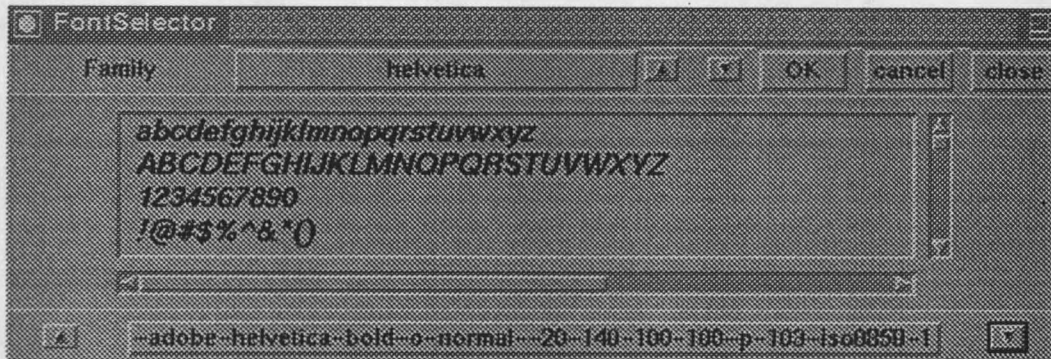


Figure 19: Font Selector

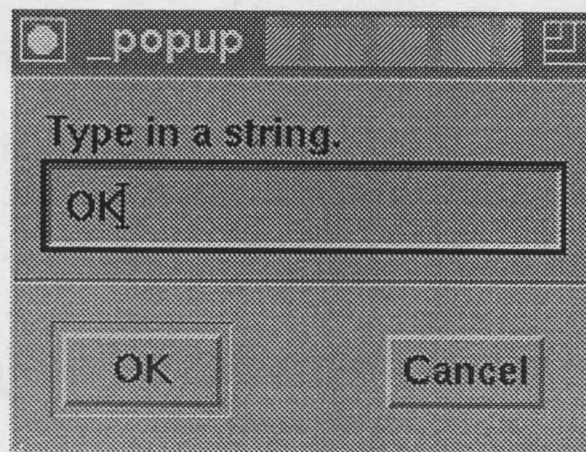


Figure 20: String Selector

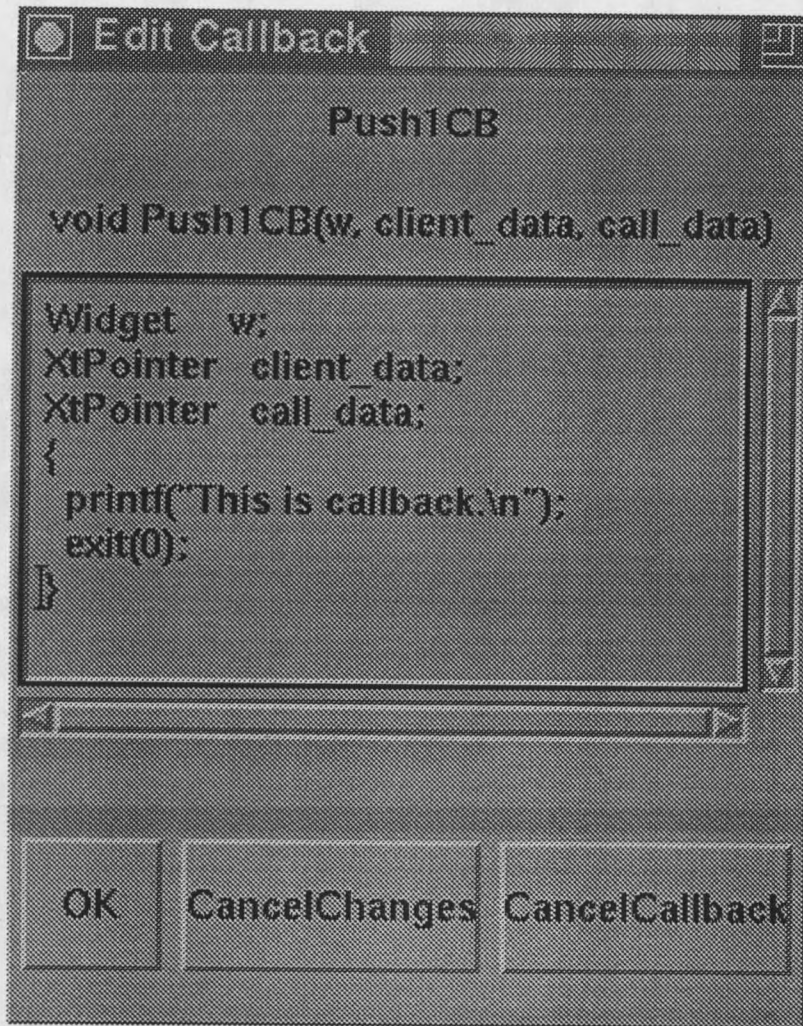


Figure 21: Widget Callback Editor

VMCG USER INTERFACE

File

All information in VMCG's current application's widget can be saved to the diskfile and can be retrieved for the next design. The File menu supplies New, Open, Save and Save As options to manage the application, as well as supplying the Exit option to exit VMCG.

Figure 22 shows the MainMenu of VMCG; Figure 23 shows the File submenu from the MainMenu; Figure 24 is a DialogShell to confirm the action of exit in the File submenu.

Run

Run is used to perform the real-time execution of the compiled C source code within the VMCG package.

Application

Application is used to generate a C source code for the current application.

Compile

Compile is used to compile the current application generated C source code to an executable file (.exe).

Widget

Widget contains a submenu which includes all widget types. Selecting one of them from the submenu will set the current temporarily widget type to the chosen widget type, which can be served as the child widget as the current active widget, which is shown in the Tree Hierarchy Interface.

Edit

Edit contains the Edit Active Object and Edit Temp Object submenus.

- **Edit Active Object** : The Edit Active Object is used to edit the current active widget.

Edit Resource : Edit Resource is used to pop up the RGB Color Selector, List Color Selector, Font Selector, and Labelstring Selector options to edit the resources for the current active widget.

Edit Callback : The Edit Callback is used to pop up an Edit Callback interface to edit the callback for the current active widget.

Delete : The Delete is used to delete the active widget and its children widgets. The Tree Hierarchy Interface and Application Design Interface will be dynamically updated after the deletion.

- o **Edit Temp Object :** The Edit Temp Object is used to add one child widget to the current active widget.

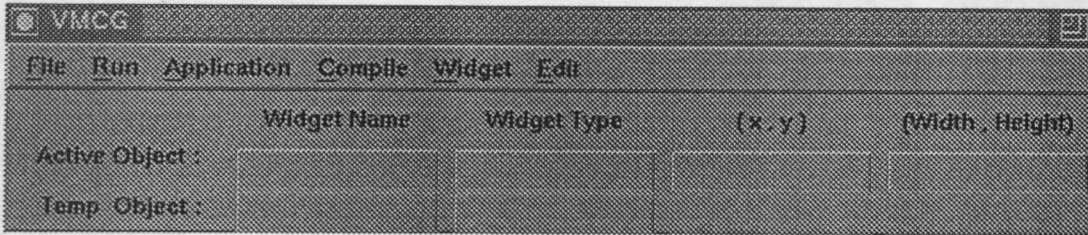


Figure 22: VMCG MainMenu

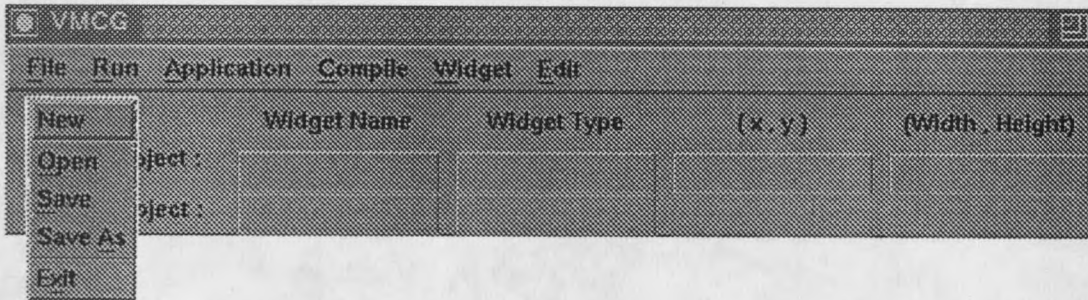


Figure 23: VMCG File submenu

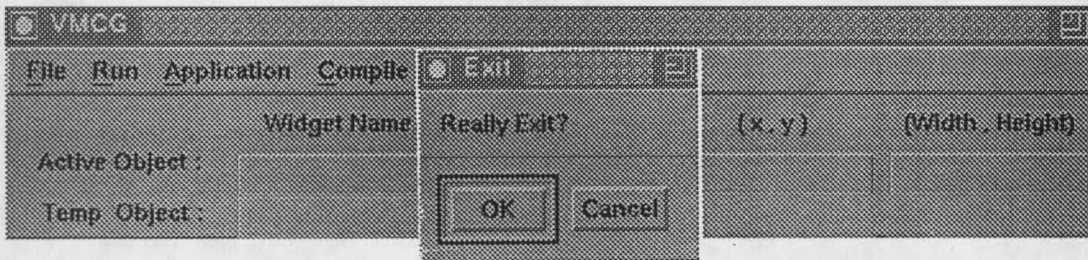


Figure 24: VMCG Exit DialogShell

CONCLUSIONS AND FUTURE DIRECTIONS

VMCG supplies an intuitive and easy way to design an X/Motif GUI by supplying a Tree for the widget hierarchy, a mouse movement based geometry management for child widget over its parent container widget, resource editors, callback editor, and C source code generator, its success in designing X/Motif GUIs visually. The future work is to introduce more widget types and more fancy resource selectors.

References

- [1] Marshall Brain, *Motif Programming*, Digital Press
- [2] Dan Heller, *Motif Programming Manual*, OSF/Motif Edition, O'Reilly&Associates Inc., 1991
- [3] Adrian Nye and Tim O'Reilly, *Xtoolkit Intrinsic Programming Manual*, OSF/Motif Edition, O'Reilly&Associates Inc.; 1990
- [4] Adrian Nye, *Xlib Programming Manual*, OSF/Motif Edition, O'Reilly&Associates Inc., 1992
- [5] Nabajyoti Barkakati, *X Windows System Programming*, SAMS
- [6] Douglas A. Young, *X Window System Programming and Application with Xt OSF/Motif Edition*, Prentice-Hall, Englewood Cliffs, NJ 07632
- [7] Integrated Computer Solutions, Inc. *ICS Widget Databook*, Integrated Computer Solutions, Inc., 201 Broadway, Cambridge, MA 02139, 1992

MONTANA STATE UNIVERSITY LIBRARIES



3 1762 10254486 1

HOUCHEM
BINDERY LTD
UTICA/OMAHA
NE.