Proceedings of the IASTED International Conference
Software Engineering and Applications
October 6–8, 1999, Scottsdale, USA

Integrating a GUI into a Command Driven Application

Brian A. Malloy*, John D. McGregor
Dept. of Computer Science
Clemson University
Clemson, SC 29634
{malloy,johnmc}@cs.clemson.edu

Shannon Hughes
Advanced Software Construction Center
Lucent Technologies
Cary, NC 27511
srhughes@lucent.com

ABSTRACT

Many applications in use today are driven by a command line interface rather than a graphical interface. Most of these applications are robust, popular and proven in general usage. The convenience and ease of use that a graphical interface provides might make these applications available to a wider range of users. In this work, we describe a technique that exploits ostreams *and* subtyping *to integrate a graphical interface into a command driven application. The technique that we propose does not create a new process to construct the communication linkage nor does it require a sentinel to terminate the linkage. We use the technique to integrate a graphical interface into legacy code, a grader program, that maintains a list of student grades. The grader program is coded in C++, exploits the* Command *and* Envelope-Letter *patterns and uses the standard library containers, algorithms and iterators to implement much of its functionality.*

Keywords: Design Patterns, class diagram, graphical user interface, component, architecture, standard C++ library

1 INTRODUCTION

There are many applications in common use today that are driven by a command line interface rather than a graphical interface. Most of these applications are robust, popular and proven in general usage. The uuencode or pkzip utilities that operate under the MS-DOS environment, are typical examples of these frequently used applications. However, the convenience and ease of use that a graphical interface provides might make these applications available to many people who would not otherwise use them. Thus, the integration of a graphical user interface into a command driven application represents a useful contribution to the computer industry.

Previous approaches to integrating a graphical interface into legacy code have focused on using a variation of a pipe to negotiate the communication between two processes where the master process is the graphical interface and the slave process is the command driven application; the vehicle of communication between the two processes is a file[1, 2]. The advantage of this approach is that, in most cases, the communication can be established without knowledge of the source code of the command driven application. The disadvantages of the technique are that the I/O in the command driven application must be limited to stdin, stdout and stderr and the linkage can be slow and, for some large applications, the system may stall[2].

In this paper, we present a technique that exploits ostreams and subtyping to integrate a graphical interface into a command driven application. An ostream is a mechanism for organizing and maintaining sequences of characters[4]; *subtyping* entails construction of a derived class that refines the behavior of the parent class[3]. The technique that we propose does not create a new process to construct the communication linkage and does not require the use of a sentinel to terminate the linkage. Further, our technique is not limited to communication through stdin, stdout or stderr.

We use our technique to integrate a graphical interface into legacy code, a grader program, that maintains a list of student grades. The grader program is coded in C++ and exploits the Command pattern and the Envelope-Letter pattern, and uses the standard library containers, algorithms and iterators to implement much of its functionality[1]. The graphical interface that we integrate into grader uses the V C++ GUI toolkit[6], a public domain package that supports

*Brian completed part of this work on sabbatical at the National University of Ireland, Maynooth, Ireland.

[1] Historically, the standard library has been referred to as the Standard Template Library (STL)

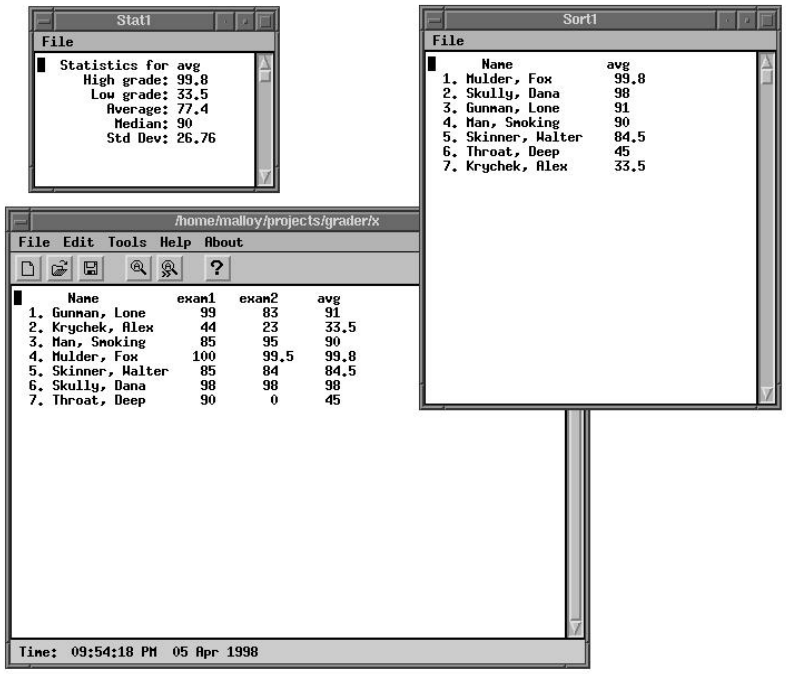

Figure 1: This figure illustrates the display of a grade list, in the lower left corner of the figure, together with a display of a statistics window in the upper left corner. The window in the upper right corner of the figure illustrates a sort window with the student list sorted by grade values in column avg from high grade to low grade.

most popular graphical objects found in GUI environments and performs on multiple platforms including Linux, Unix and Windows 95.

The remainder of this paper is organized as follows. In the next section we present an overview of the grader program followed by a review of previous techniques to integrate a graphical interface into legacy code. In Section 4 we present the design of the command line package that exploits object technology and the standard library and in Section 5 we present the design and implementation details of the graphical interface that we integrate into the grader program and in the final section we make some concluding remarks.

2 OVERVIEW OF THE GRADER PROGRAM

The *grader program* implements the functionality required to maintain a list of grades or *grade list*. Details of the design and implementation of the grader program are provided in subsequent sections; we now present a description of this functionality.

There are sixteen commands provided to users of both the command line version and the graphical version of grader. These commands enable functionality required to maintain a database of grades, including commands to display the grades, find particular students, delete students, delete columns of grades, create new columns of grades possibly by numerically combining existing columns and commands for performing statistical analysis of the database. The picture in Figure 1 illustrates three of these commands. The lower left section of Figure 1 illustrates the display command that displays all names and grades in the current grade list. The upper left section of the figure illustrates the stat command that, for a specified column, lists the highest grade, the lowest grade, the average of all the grades, the median and the standard deviation. The upper right section of Figure 1 illustrates the sort command that sorts a column of grades and prints the corresponding student name to the left of the grade. For example, the third command in the figure is *define* that permits the user to create a new column by (1) defining the new column by a formula that contains column names, constants or operators, or (2) defining a new column with each student's grade set to zero.

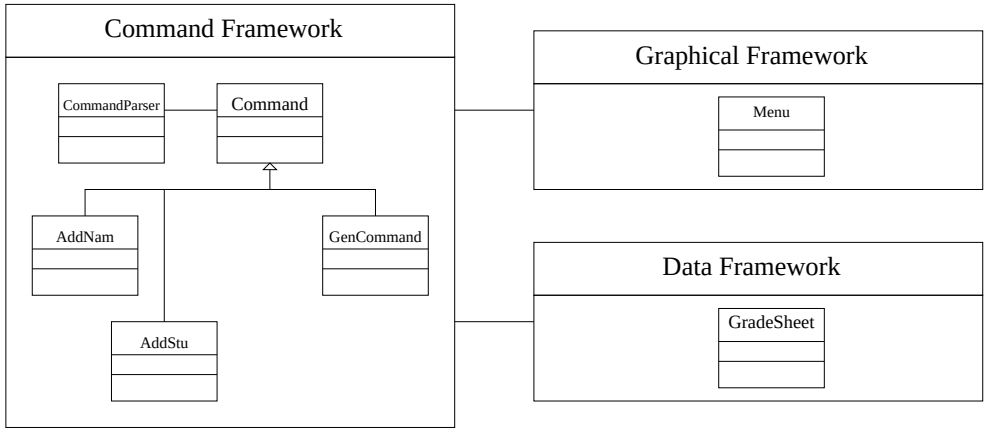

Figure 2: This figure illustrates the relationships among the class frameworks used in the grading program.

3 INCORPORATING A GUI INTO LEGACY CODE

There have been three important approaches to integrating a graphical interface into legacy applications that have been reported in the literature[2, 5, 7]. All three of these approaches use a pipe as the communication interface between the legacy application and the graphical interface; we review two of these approaches in this section.

Reference [2] presents an approach that uses anonymous pipes to link an MS-DOS program to a Windows program. Anonymous pipes are typically used to direct communication between a parent and child process; however, the technique presented in reference [2] permits the linkage to be accomplished without knowledge of the source code of the MS-DOS program. To establish an input linkage, the auxiliary pipe technique consists in (1) creating a pipe, (2) setting the read handle of the pipe to stdin, and (3) running the MS-DOS program as a new child process that inherits handles created by the parent (GUI) process. After the input pipe linkage has been established, all input to the MS-DOS program will be from the pipe rather than the console or keyboard. An output link is established in analogous fashion.

Reference [1] presents an approach that uses a *pseudo terminal* to integrate a legacy application into a Unix graphical user environment. A *pseudo-terminal* is a pair of devices, or files; one device behaves as a master and the other as a slave. Data written to the master appears as input to the slave and data written to the slave appears as input to the master. The slave portion of the pseudo-terminal can be used by a child process as a terminal device (stdin, stdout, stderr). To integrate the legacy application, the pseudo-terminal is used together with an xterm, telnet and a mediator process to give the user accelerators and macros. The disadvantage of using the pseudo-terminal approach is that older variants of Unix code can be difficult to implement. Also, the performance of pseudo-terminals can be sluggish.

4 THE DESIGN OF THE COMMAND LINE APPLICATION

Figure 2 illustrates the framework of classes and their relationships in the grader program. The figure includes three frameworks: the *command framework*, the *data framework* and the *graphical framework*. The command and data frameworks were included in the original command driven interface; the graphical framework was added after the command interface was fully implemented. The figure summarizes the classes in each of the frameworks; we provide details for the *graphical framework* in Section 5.

5 DESIGN OF THE GUI

In this section we present the class framework used to construct the graphical interface to the grader program. The GUI design is composed of V classes, or classes that comprise the V package, and classes derived from V that implement our graphical interface. Figures 3 and 4 illustrate both kinds of classes where classes starting with the letter "v" are the V package classes and the other classes are the ones that we have written to implement our graphical interface. We will not discuss those classes that comprise the V package; the interested reader can find details of these classes in

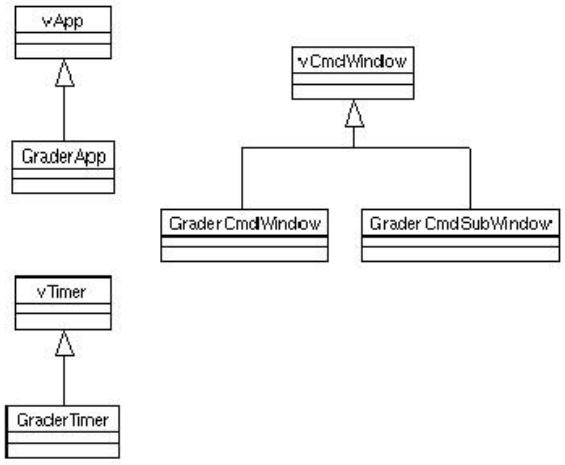

Figure 3: This figure illustrates the classes that create the top level window for users of the grader program.

the V Reference Manual[6]. Class GraderApp, depicted in Figure 3, is derived from the V class vApp; GraderApp contains function AppMain whose actions create the top level window that represents the graphical interface for the application currently in use. All information about active windows is maintained in GraderApp so that if a window is closed, or the application is closed, the memory for the window will be deallocated. Thus, all instances of windows are created and deleted in GraderApp.

Class GraderCmdWindow, also shown in Figure 3, is derived from vCmdWindow; an instance of GraderCmdWindow is the top level window that controls the menu bar, command bar (tool bar), and status bar. Class GraderCmdSubWindow is also derived from vCmdWindow; an instance of GraderCmdSubWindow is a sub-window that can be opened from the top level window. Sub-windows do not contain the same items that the top level window contains; our sub-windows contain two menu items and no tool or status bars. The purpose of our sub-windows is to display subsets of the text that is displayed on the top level window. For example, if the user sorts student averages, the results are displayed to a sub-window to allow the user to see the averages isolated from the top level window. Obviously the elaborate tool bar or menu bar, used in a top level window, is not needed in a sub-window to accomplish simple tasks such as painting subsets of text. The two menu items allow the user to close the sub-window or save the canvas' text to a file.

Figure 4 illustrates the classes that implement all dialogs in the graphical interface. Class StudentModalDialog, shown in the figure, constructs the modal dialog that permits the user to choose student names for modification or deletion. Class GradeModalDialog is the modal dialog that permits the user to change a student's grade. The GradeModalDialog instance allows a user to choose a student name from one list box and a grade item (test, program assignment, etc) from a second list box. After both list items are selected the current grade for the selected student and grade item is displayed as a label within the dialog window. Also included in this dialog window is an option to permit the user to enter a new grade in a text box updating the selected student's grade. The ColumnChoiceDialog class allows the user to pick a grade from a list of grades. This class is used to delete grades, to sort student grades in ascending order and to view statistics on a given grade (class average, highest grade, etc). The CreateListDialog class is a modal dialog that allows the user to create a new student list. To build a new student list the user begins entering student names in a text box and then uses the mouse to select a button to either continue adding students to the student list or to terminate the expansion of the student list.

Finally, the GraderCanvasPane and GraderSubCanvasPane classes handle any text displayed to the screen and redraws a window's canvas after maximizing or resizing the window. The GraderTimer class is used to update the system time every second and displays the time on the status bar.

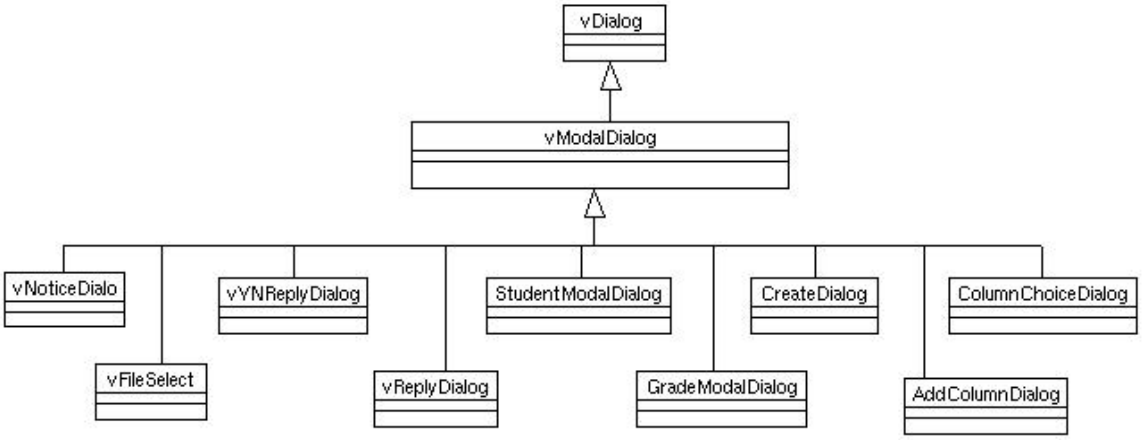

Figure 4: This figure illustrates the classes that are used to implement dialog boxes in the graphical interface of the grader program.

6 CONCLUDING REMARKS

In this paper, we describe our technique for incorporating a graphical interface into a command driven application. Our technique exploits ostreams and subtyping and requires minimal alteration to the application source code. We have used our technique to incorporate a graphical interface into a grader program written in C++ that uses the Command pattern and the Envelope-Letter pattern. The table in Figure 5 summarizes the technique that we used to incorporate the graphical interface into each command. The first column in the figure lists the commands that were integrated into the graphical interface using a direct call to the instance of the *command letter*, the second column lists the commands that were integrated using sub-classing, the third column lists the commands that were integrated using ostreams, and the fourth column lists those commands that were integrated using the facilities provided by the V package. Storage of data in the grader program is accomplished using containers, algorithms and iterators in the standard library.

Direct Call	Sub-classing	Ostream	V Library
addnam	new	sort	help
addstu	define	stat	find
delcol	modify	display	
delnam	formula	define	
exit		modify	
save		formula	
load			

Figure 5: This table contains columns that summarize the technique that we used to incorporate the graphical interface facilty into each command. Some of the commands required the use of both sub-typing and ostream techniques.

References

[1] W. L. Crowe. A pseudo-terminal class for unix. *C/C++ Users Journal*, pages 21–29, March 1998.

[2] D. Klementief. A windows shell for legacy ms-dos applications. *C/C++ Users Journal*, pages 71–74, June 1997.

[3] J. Rumbaugh, M Blaha, W. Premerlani, F Eddy, and W. Lorensen. *Object-Oriented Modeling and Design.* Prentice-Hall, 1991.

[4] B. Stroustrup. *The C++ Programming Language.* Addison-Wesley, 1997.

[5] U. Vahalia. *UNIX Internals: The New Frontiers.* Prentice-Hall, 1996.

[6] Bruce Wampler. The v c++ gui framework. http://www.objectcentral.com.

[7] D. A. Young. *The X Window System Programming and Applications with Xt: OSF/Motif Edition.* Prentice-Hall, 1997. ISBN: 0-13-497074-8.