A Truly Implementation Independent GUI Development Tool

ABSTRACT

Over the last few years, graphical user interface programming has become increasingly prevalent. Many libraries and languages have been developed to simplify this task. Additionally, design tools have been created that allow the programmer to "draw" their desired interface and then have code automatically generated. Unfortunately, use of these tools locks the programmer into a particular implementation. Even if the tool targets a multi-platform library (e.g. Tcl/Tk or JVM), the flexibility of the result is constrained. We present a truly implementation and platform independent solution. PROJECT_NAME generates Ada code targetted to an object-oriented set of graphical user interface specifications with absolutely no implementation dependent information. The pattern used to derive these specifications is an improvement over the "Abstract Factory" Pattern, as it allows both the specification and implementation to take advantage of inheritance. The user can then select an implementation (for example, Tcl/Tk or JVM) at compile time. PROJECT_NAME itself is also written using the same specifications, therefore it requires no modification to target a new implementation or to use a new implementation itself. PROJECT_NAME is currently being used to design the user interface for a satellite ground station.

1 INTRODUCTION

The combination of cheaper computing power and the introduction of the computer into the household has brought about a change in the nature of the user interface of most programs. The use of graphical user interfaces (GUIs) rather than text-based user interfaces has become increasingly widespead. Although graphical user interface programming was originally both highly complicated and system dependent, a large collection of widget libraries and GUI design tools have been created to simplify this task.

GUI design tools allow the user to visually create the graphical user interface for their programs, usually by clicking and dragging out the outline of a widget and then filling in the properties via a dialog. Once the design is complete, code is automatically generated which will create the user interface. This code usually targets a particular widget library, which also provides operations which allow the user to query the status of the widgets (to read the text in a text entry widget, e.g.) Unfortunately, many of these tools restrict the user to a particular platform (e.g. Windows).

Some GUI design tools target libraries that have been implemented across several platforms. While these tools allow a programmer to use the same generated source code on many different machines, they still constrain the user to a particular implementation. Ideally, we would like to be able to separate the design of the user interface from the selection of an implementation. PROJECT_NAME allows the programmer to do precisely that.

PROJECT_NAME generates code for an object-oriented set of graphical user interface specifications that contains no implementation dependent information. A particular implementation may be selected at compile time of the generated source code. Currently, Tcl/Tk and JVM implementations are provided. PROJECT_NAME is also implemented using the same libraries. This means that not only can the programmer pick an implementation for the output of the GUI design tool, but also the tool itself can be easily targetted to different implementations.

A new design pattern, the Peer Pattern, allows us to simplify the configuration management of mulitple implementations of the same specification. We describe the Peer Pattern, a novel solution to the same problem solved by the Abstract Factory Pattern, in Section 2. In Section 3, we describe the current functionality of PROJECT_NAME. We contrast PROJECT_NAME with prior work in this area in Section 4. Finally, we conclude and give directions for future work.

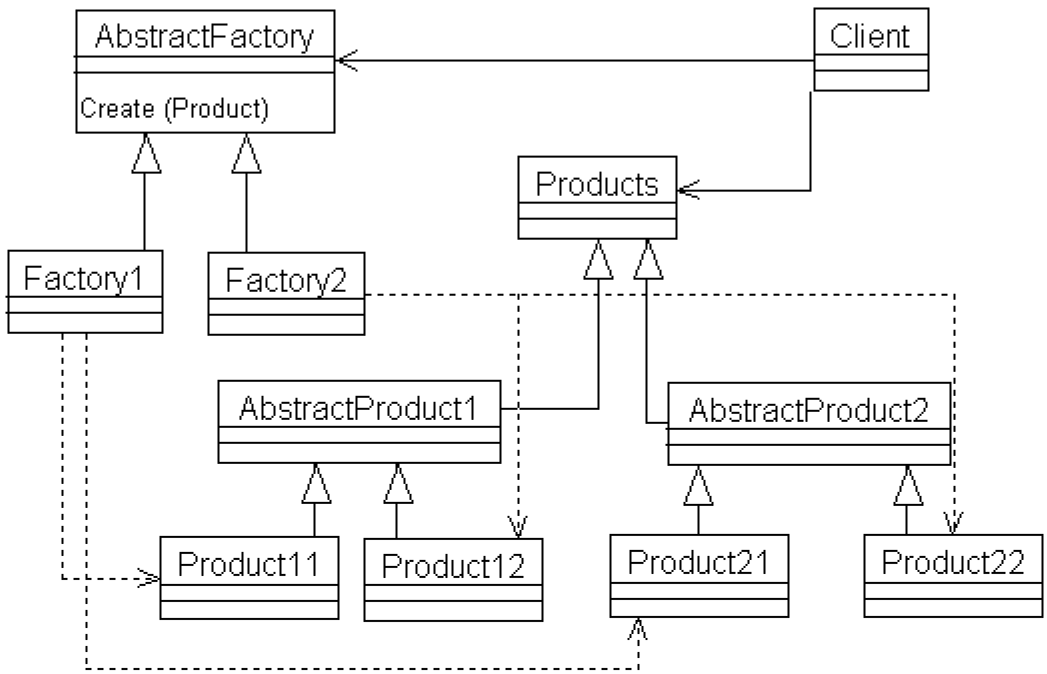

Figure 1: Abstract Factory Pattern class hierarchy

2 MODERNIZING THE ABSTRACT FACTORY

Gamma et al describe the Abstract Factory Pattern, which is useful, e.g., for "a user interface toolkit that supports multiple look-and-feel standards, such as Motif and Presentation Manager" [GHJV94]. Although the example is specified as providing different look-and-feel standards, ET++ [WGM88] used the same pattern to achieve portability across different window systems. In the Abstract Factory Pattern, each user interface item is defined as an abstract class, and the various implementations (in this example, Motif and Presentation Manager) are defined as children of the abstract class. Then, a factory is defined. The factory merely calls the appropriate creation methods depending on which implementation is currently selected. Matthew Heaney [He99] has implemented the Abstract Factory Pattern in Ada in two ways. The first is exactly as described by Gamma et al; he also notes that you can accomplish the Abstract Factory simply by doing static package renaming. Figure 1 shows the class hierarchy of the Abstract Factory Pattern obtained from the SIGAda Patterns web site [He99]. The triangular arrows point from a child class to its parent class. The dashed lines point from a client to the package whose objects it instantiates. The solid arrows point from a client to the packages it utilizes. Although it is not shown in the diagram, the client would also need to access AbstractProduct1 and AbstractProduct2 to use the associated methods.

The problem with the Abstract Factory Pattern becomes evident when we attempt to extend an abstract product. For example, suppose we wish to create AbstractProduct3, which extends the functionality of AbstractProduct2. It is a simple matter to create the abstract class, by simply extending AbstractProduct2. When we implement Product31, it is likely the case that we would like to extend the class Product21. Unfortunately, Product31 is already a child of AbstractProduct3. Many important object-oriented languages (such as Ada and Java) do not allow multiple inheritance; therefore, we are required to reimplement the functionality of Product21 in Product31.

One solution is to simply dispose of the Abstract Factory altogether, and implement two separate hierarchies, making sure that each has the same methods and class names. The user then selects an implementation by including the appropriate set of files in the project (via a makefile, compiler flags, or similar). This has the disadvantage that multiple copies of the specifications are created, each differing only in the representation of the data.

We instead solve the problem by creating the *Peer Pattern*. In the Peer Pattern, one hierarchy gives the specification of the classes. The client sees only this hierarchy. A second hierarchy actually implements the specification. This is illustrated in Figure 2.

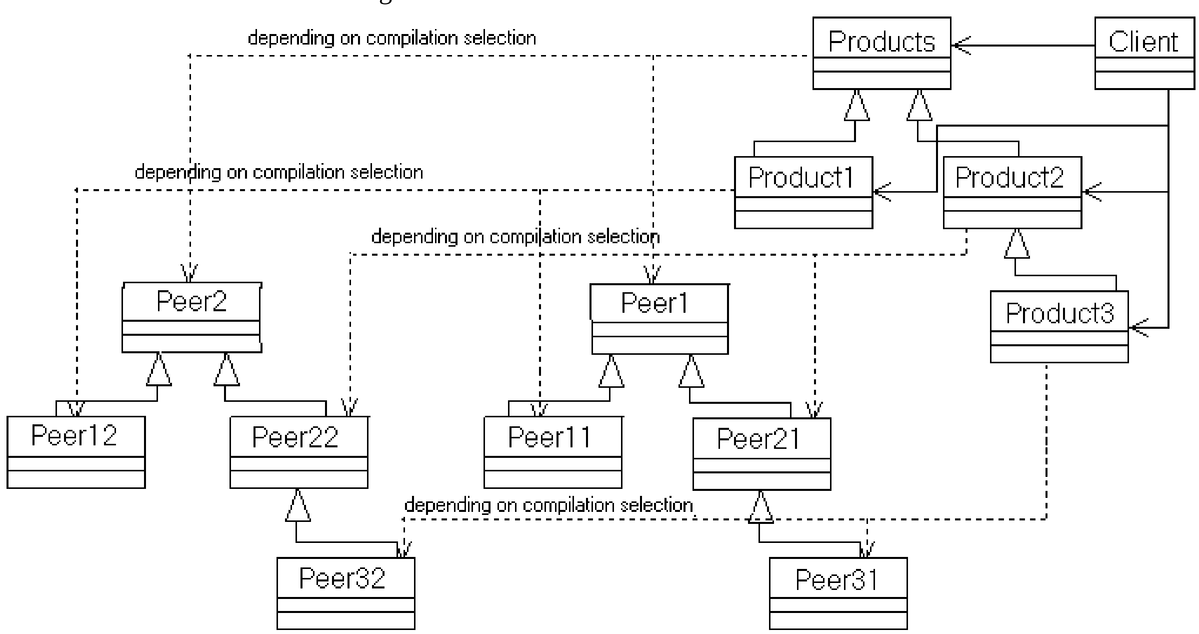

Figure 2: The Peer Pattern class hierarchy

In Figure 2, the dashed lines leading from the products to the peers are labeled “depending on compilation selection.” The reason for this is that while the specifications of the products are exactly the same across all implementations, the bodies of these packages are different. To accomplish this, the root level object of Product has the following declaration:

```
type Object is tagged record
   My_Peer : Peer.Peer;
end record;
```

The peer type can then be defined on an implementation-specific basis. For the Tcl/Tk implementation, the Peer contains a string pointer giving the name of the Tk widget. For the JVM implementation, it is a classwide pointer to a Java object. To simplify the selection of an implementation at compilation time for PROJECT_NAME, we organize the files into the following directories:

- Gui: contains the package specifications of the "products", i.e. the widgets, windows, etc. These are named: Gui, Gui.Container, Gui.Widget, Gui.Widget.Radio_Button, etc.
- Lib: contains package specifications and bodies that are useful across all implementations of Gui.
- Tcl_Peer: contains both the specification and body of the package Peer for the Tcl/Tk implementation. It also contains a set of package bodies, based on this Peer package, for the specifications in the Gui directory.
- JVM_Peer: similar to Tcl_Peer, but instead uses Java components to implement the Gui specifications.

This mechanism totally separates the specification from the implementation (since the Gui files contain only a reference to the implementation's representation). Additionally, each implementation is free to create a separate class hierarchy. This solution does require some support from the compilation environment; however, it solves configuration management issues pertaining to having multiple copies of specifications that differ only in the private section (where the representation is stated).

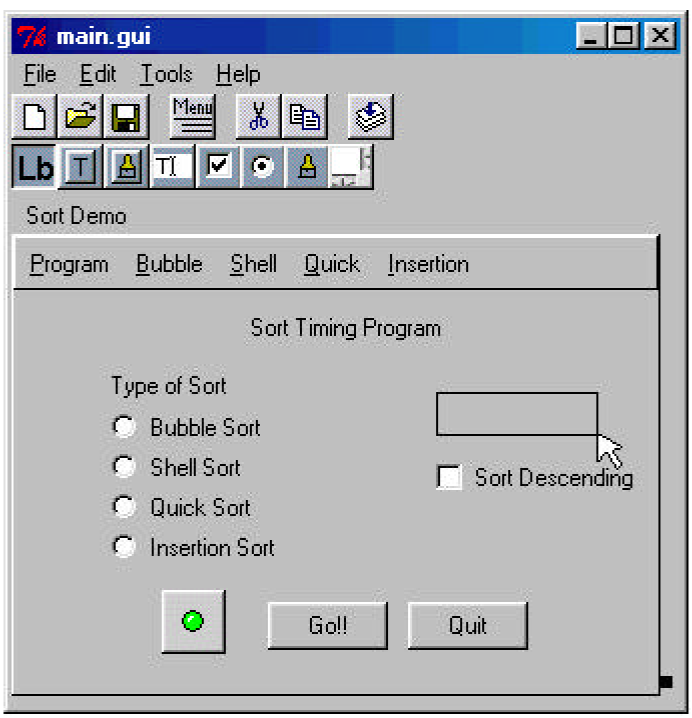

Figure 3: The PROJECT_NAME window after opening a GUI file.

3 PROJECT_NAME FEATURES

PROJECT_NAME provides an intuitive interface for designing a graphical user interface. Figure 3 shows what the Tcl/Tk implementation of PROJECT_NAME window looks like while editing a file. The first row of buttons is a toolbar. These buttons allow the user to create a new window, open a previous window, save the current window, delete or duplicate the selected widget, start the menu editor, or compile the GUI to Ada code. The second row of buttons is used to select what type of widget will be added (currently text labels, text buttons, picture buttons, and text entry widgets, check buttons, radio buttons, static pictures and listboxes are supported, and more are being added.) After selecting a widget type, the designer uses the left mouse button to click and drag out a new widget (as shown in Figure 3).

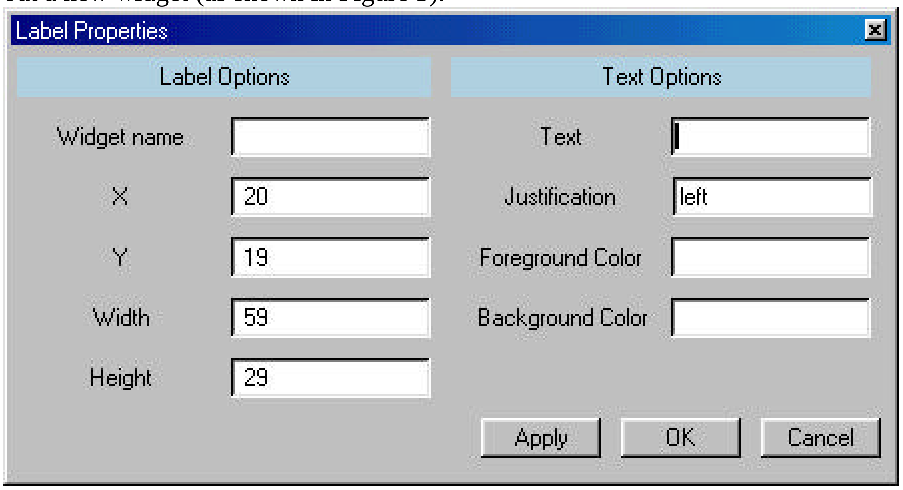

Figure 4: Dialog for entering the properties of a label widget

Once the user releases the left mouse button, a dialog box appears that asks the user to fill in the rest of the properties of the widget. The location and size of the widget are automatically filled in. Figure 4 shows the dialog for a label widget. The user is asked to give a name to the widget. This name is used as a variable name. For a label, the user selects the text that will be displayed, its justification, and also the foreground and background colors. Some widgets, such as buttons, also have actions associated with them. These actions indicate what should happen when the button is pushed, the user presses a key in a text entry widget, etc. The user specifies an action by giving a fully qualified Ada procedure name (with package name, e.g. `Actions.Ok_Button`).

Additionally, PROJECT_NAME has a menu editing tool, whose visual interface is modeled after a Windows-based file browser. Arbitrarily nested menus can be created using this tool. The menu is then displayed in the window when the menu editor is closed (as shown in Figure 3). Menu items can also be associated with accelerator keys by typing the shortcut that will appear in the menu (e.g. "Ctrl+X"). PROJECT_NAME then also generates code for the window that will invoke the action associated with the menu choice when the key sequence is pressed.

The PROJECT_NAME GUI designer allows the user to generate a simple graphical user interface without any knowledge of GUI programming. Once they are pleased with their design, pushing the compile button will generate all of the necessary Ada code to display the interface, and handle all of the events. The designer can then focus on the functionality of the program.

4 PREVIOUS WORK

There are several efforts in progress to provide GUI libraries and design tools for Ada. Both the Aonix GUI Builder [Ao97] and the CLAW Application Builder [BM97] generate Ada code that uses the Win 32 libraries to implement the user interface widgets. (Note CLAW claims to be "portable," but this portability refers to its ability to be used with several compilers, not on several platforms). This has advantages if you are only interested in that particular platform (as you can take advantage of the unique features of the Win 32 libraries), but requires you to entirely redesign your application's user interface to port it to a new platform.

Additionally, several projects provide bindings for Ada to libraries that run on many different machines. First, TASH [We96] provides a thin binding to Tcl/Tk [Ou94]. Tcl/Tk implementations are available on Windows, Macintosh, Linux and UNIX. Westley is also creating an object-oriented thick binding to the widgets in the Tk toolkit. Ada compilers are also beginning to target the Java Virtual Machine [Ao97, CDG97]. Using Tcl/Tk to provide the user interface has a speed advantage for the code outside the user interface (since it will be compiled to native machine code) and also makes it easier to interface with native code, as the Java Native Interface [Su97] is quite complex compared to Ada's interfacing pragmas. Java [GJS96], however, has a far richer set of graphics primitves available.

GtkAda [BBC99] provides a object-oriented binding to the Gtk+ toolkit [Ma99]. Additionally, a graphical user interface designer, GLADE [Gl99], is available for Gtk+. Gtk+ currently runs on many flavors of UNIX, and is also being ported to Win 32. The implementation of Gtk+ is in C, however, the Ada binding cleanly obscures this from the user. Gtk+ is free software distributed under the GNU Library General Public License [LGPL99].

This work is a direct extension of RAPID [CM98]. RAPID targetted only the TASH binding to Tcl/Tk and did not provide an object-oriented library of functions for clients to use; however, we now allow the user to select from multiple implementations and also provide an object-oriented interface for clients, which simplifies using the generated code.

5 CONCLUSIONS AND FUTURE WORK

In conclusion, PROJECT_NAME allows an Ada programmer to add a GUI to his program in a very simple and portable way. The GUI design tool uses a very intuitive visual process to create the desired interface. Not only is the user given portability across several platforms (as both Tcl/Tk and JVM implementations are provided), but also the user has the ability to use the same design tool with different implementations.

Since PROJECT_NAME is freeware and will run on a variety of computers, this makes it an attractive tool for use in an educational setting. At a recent SIGCSE conference, it was pointed out that CS curricula should address human-computer interface issues and visual programming [Si98]. PROJECT_NAME allows students to experiment both as an implementer and client of graphical user interface libraries, and also further demonstrates the utility of Ada both as a commercial-use language and a teaching language. PROJECT_NAME is currently being used to create the user interface for a satellite ground station.

Additionally, the source for PROJECT_NAME is available for download via ftp from the Internet. This provides an opportunity for others to contribute to the product by adding additional widgets, additional functionality to the existing widgets, or additional implementations. We also intend to continue to improve the product based on our observations from using it, and input from others. Since PROJECT_NAME uses the object-oriented features of Ada 95 in its design, adding widgets is a straightforward process consisting of creating a new type and overloading the appropriate methods. The PROJECT_NAME design process also greatly speeds expansion via bootstrapping and code reuse.

Finally, we also present a new solution to the configuration management problem of having multiple implementations of a single specification via the Peer Pattern. This is an improvement on the Abstract Factory Pattern as it allows both the specification and the implementation to have separate hierarchies. In the future, we hope to provide a new pattern that has all of the functionality of the Peer Pattern, while allowing the program to use multiple implementations simultaneously.

REFERENCES

[Ao97] Aonix Inc. *Object Ada*, 1997.

[BBC99] E. Briot, J. Brobecker and A. Charlet. "GtkAda : an Ada95 binding for Gtk+", http://www.ada.eu.org/gtkada.

[BM97] R. Brukardt and T. Moran. "CLAW, a High Level, Portable, Ada 95 Binding for Microsoft Windows," *Tri-Ada '97*, pp. 91-104, ACM, 1997.

[CM98] M. Carlisle and P. Maes. "RAPID: A Free, Portable GUI Designer for Ada," *SIGAda '98*, ACM, 1998.

[CDG97] C. Comar, G. Dismukes, and F. Gasperoni. "Targeting GNAT to the Java Virtual Machine," *Tri-Ada '97*, pp. 149-161, ACM, 1997.

[GHJV94] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1994.

[Gl99] "GLADE: Gtk+ User Interface Builder", http://glade.pn.org.

[GJS96] J. Gosling, B. Joy, and G. Steele. *The JavaÔ Language Specification*, Addison-Wesley, 1996.

[He99] M. Heaney. "Abstract Factory Pattern" and "Abstract Factory Revisted", ACM SIGAda Patterns WG Archive, http://www.acm.org/sigada/wg/patterns/patterns/index.html.

[LGPL99] "GNU Library General Public License", http://www.fsf.org/copyleft/lgpl.html.

[Ma99] P. Mattis. "The GIMP Toolkit", http://www.gtk.org/docs/gtk_toc.html.

[Ou94] J. Ousterhout. *Tcl and the Tk Toolkit*, Addison-Wesley, 1994.

[Si98] SIGCSE Town Meeting, Atlanta GA, February 1998.

[We96] T. Westley, "TASH: A Free Platform-Independent Graphical User Interface Development Toolkit for Ada," *Tri-Ada '96*, pp. 165-178, ACM, 1996.

[WGM88] A. Weinand, E. Gamma, and R. Marty. "ET++--An object-oriented application framework in C++." In *Object-Oriented Programming Systems, Languages, and Applications Conference Proceedings*, pp. 46-57, ACM, 1988.

NOTE TO REVIEWERS: The JVM implementation is in progress. It is expected to be completed by the end of August.