BUILDING USER INTERFACES INTERACTIVELY USING PRE- AND POSTCONDITIONS

Martin R. Frank, J.J. "Hans" de Graaff, Daniel F. Gieskens and James D. Foley

Graphics, Visualization and Usability Center
College of Computing, Georgia Institute of Technology
Atlanta, Georgia 30332-0280
(404) 853-0672, {martin,graaff,daniel,foley}@cc.gatech.edu

ABSTRACT
A tool is presented which allows graphic layout of a user interface integrated with specification of behavior using pre- and postconditions.

KEYWORDS
User Interface Management Systems. Graphical User Interface Builders. Dialogue Sequencing.

INTRODUCTION
Development of user interfaces has traditionally been a tedious process, especially constructing non-textual interfaces using a textual programming language. Interface builders were developed to interactively design user interfaces. The Developer's Guide for the OPEN LOOK Graphical User Interface[3] is one of them. Like other builders, this tool is limited to specifying the appearance - not the functionality - of an interface.We have extended the Developer's Guide with a simple and powerful way to specify functionality.

METHODOLOGY
Pre- and postconditions are associated with widgets (sliders, buttons, ...). A widget has two preconditions - one for controlling widget visibility and one for controlling widget enablement. The postconditions describe how the state of the interface changes when the widget has been activated (e.g. a button pressed).The interface and application share a global state, the *blackboard*. Both can access predicates on the blackboard. They communicate by using predicate names. If either the application or the interface make changes to the blackboard the other component will be notified of the change. An application may have several different interfaces provided that the alternative interfaces use the same predicate names.

Postconditions of widgets can contain calls to a UNIX™ shell script defining application functionality. If all application functionality is contained in this shell script then there is no difference between the interpreted application in run mode and the compiled application (other than execution speed). However, the shell script method is not general enough to cover all possible interfaces (e.g. direct manipulation in an application window), so the designer is free to add code. Even in this case our tool assists in separating the interface and application provided that the programmer uses the blackboard mechanism in the intended way.

The description of user interface behavior through pre- and postconditions is used to automatically generate context-sensitive help. For example, help is given on why a certain command is disabled by examining which of the enabling preconditions are false. The system translates the missing preconditions into human-readable form using a forms-based approach and displays the help text in a pop-up window.

EXAMPLE
Let us go through the steps of designing a simple user interface to a hypothetical device consisting of two buttons, labeled "On" and "Off". The desired functionality is that these buttons are mutually exclusive, so that only one of them is enabled at any time.

First we create a base window and two buttons by dragging them from a palette. We use property sheets to enter the pre- and postconditions. The predicate *status(Device,On)* denotes that the device controlled by the interface is on, and that *status(Device,Off)* denotes that it is off. Thus the precondition for enabling the "On" button is that the device is off, in our notation *status(Device,Off)*. The postcondition of pressing the "On" button is that *status(Device,Off)* is taken off the blackboard and *status(Device,On)* is placed there instead. We enter equivalent conditions for the "Off" button. Finally we specify that the initial status of the device is off by specifying that *status(Device,Off)* is put on the blackboard at start-up time. Seven lines of text are required to specify the dynamics of this particular interface.

We can then immediately verify that we achieved the desired behavior by switching from "Build" to "Run" mode. A debugging window pops up that shows which predicates are currently on the blackboard and the interface we designed is now "alive": only one of the buttons is enabled and clicking on the enabled button will enable the other one. Moving the cursor over the disabled "Off" button and pressing the help

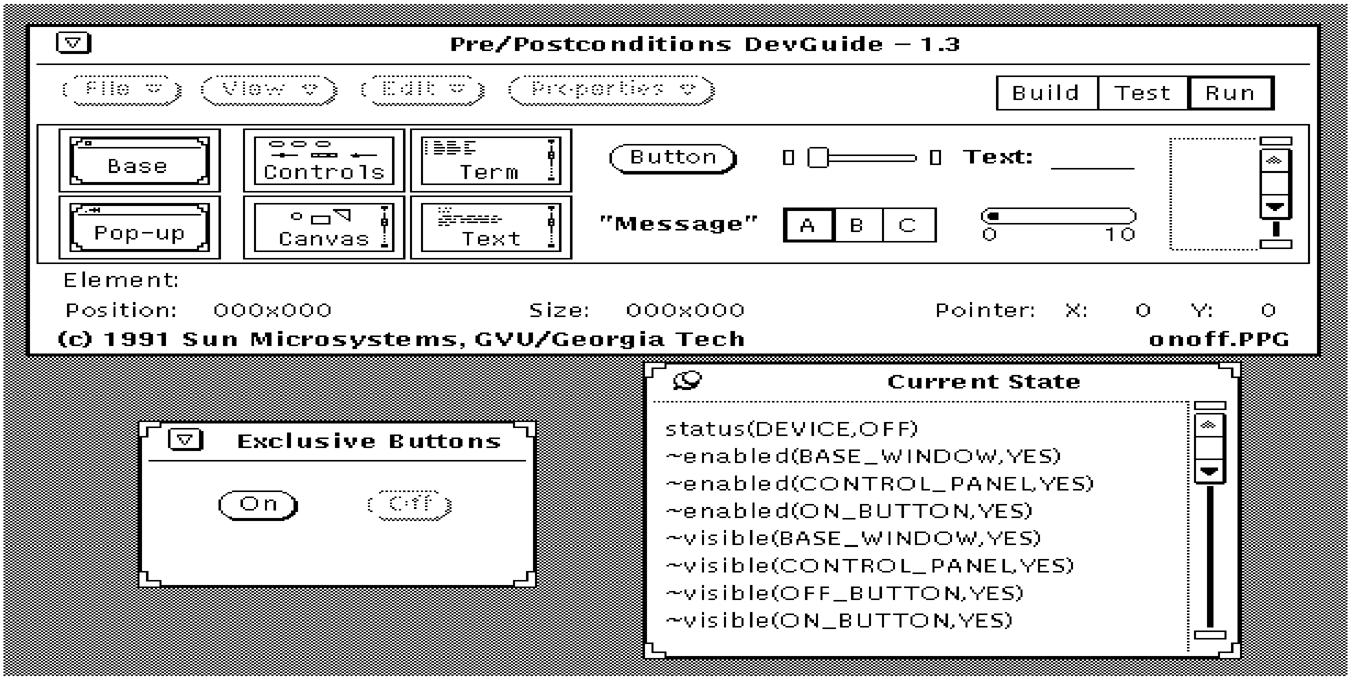

Figure 1: The Example Interface in Run Mode

key pops up the message “The button Off is disabled because the Status of the Device is Off”.

In the figure the Developer’s Guide is shown in “Run” mode. One of the buttons is greyed out as desired and the blackboard is shown. The tilde (~) before some of the predicates denotes that they are automatically generated system predicates controlling the visibility and enablement of interface objects. The other predicates are designer-defined and are used to communicate between interface and application or to control sequencing. Here, the current state is that the device is off (the first predicate). Both buttons, their control panel and the base window are visible (the last four predicates). Only the “On” button, the control panel and the base window are enabled.

EXTENSIONS

The help facility will be extended so that help is given not only on why a widget is disabled but also on how to *enable* it. This is done in two steps. First, the system finds out which predicates are missing for the precondition of the disabled command to evaluate to true. Second, it finds out which commands or command sequences would set these predicates by examining the postconditions. This search recursively extends over several levels through backchaining.This form of help has already been implemented in the framework of the User Interface Design Environment [1], but has not been integrated with this tool so far.

We will add a graphical front-end tool that can generate some of the typical pre- and postconditions automatically. The tool will translate graphical connections between user interface objects into corresponding pre- and postconditions. This will make designing interfaces with our system more comfortable. However, we do not intend to hide the underlying pre- and postcondition engine from the designer since there are many constraints which cannot easily be expressed in a graphical way.

CONCLUSIONS

Relieving user interface designers from the burden of low-level programming is bound to boost productivity. While we have not conducted formal studies on the usage of our enhanced Developer’s Guide versus the original and C or C++ to add functionality, we noticed that we ourselves started to experiment with a variety of interfaces and interface alternatives simply because our tool makes it so simple - and fun!

ACKNOWLEDGEMENTS

Partial funding was provided by Sun Microsystems’ Collaborative Research Program and by National Science Foundation #IRI-8813179.

REFERENCES

[1] Foley, J., W. Kim, S. Kovacevic, and K. Murray, “Defining Interfaces at a High Level of Abstraction”, IEEE Software, (6)1, January 1989, pp. 25-32.

[2] Gieskens, D., and J. Foley, “Controlling User Interface Objects Through Pre- and Postconditions” in Proceedings of CHI’92, Monterey, CA, May 1992.

[3] Sun Microsystems, Inc., “Open Windows Developer’s Guide 1.1, Reference Manual”, Part No. 800-5380-10, Revision A, of June 1990.