A Graphical User Interface Builder for Tk[1]

Stephen A. Uhler

Sun Microsystems Laboratories

ABSTRACT

I will demonstrate a prototype graphical user interface builder for **Tk** that provides a direct manipulation interface for building, editing and testing **Tk** applications. A novel dynamic "smart-grid" combines the familiarity of a spreadsheet-like WYSIWYG interface, with the powerful constraint based geometry management capabilities of **Tk** to provide automatic widget alignment, distribution, and resize management capabilities.

Introduction

Tk and **Tcl** are rapidly expanding beyond their initial user community of researchers and software developers. However, to be an effective tool for computer users as well as computer programmers, the generation and layout of user interface elements needs to be automated and simplified. The user interface builder attempts to combine familiar interface concepts from common user interface models with the inherent strengths of **Tk** to provide a powerful, easy to use system for designing graphical user interfaces.

Program description

The user interface builder provides a direct manipulation interface for the layout and placement of user interface elements (widgets). The interface layout is created by dragging widgets from a palette onto a layout area, or *table,* originally consisting of blank rows and columns. The geometry management is handled in **Tk** with *blt_table,* a table based geometry manager by George Howlett.[2] Once a widget is placed onto the *table,* it may be dragged with the mouse to a different row and column, or changed in shape or size to span multiple rows or columns. The user interface style is a combination of a spreadsheet, where entries are arranged in rows and columns, and a drawing program, where items are selected from a palette and displayed in a drawing area. The two interface paradigms are combined by automatically snapping the widgets to a grid, whose spacing changes dynamically, shrink-wrapping around the widgets as they are placed. This *smart grid* provides easy alignment and spacing for widgets of various shapes and sizes, while maintaining the familiar notion of a table.

The *smart grid* concept is extended to enable the table to be interactively resized by the user, while giving the table developer a flexible, yet easy mechanism for specifying how individual widgets grow or shrink as the entire table is resized. The resize behavior is specified first by identifying which rows and columns of the table can grow or shrink as the table changes size. Next, each widget is configured to specify how it *floats* or *sticks* to the sides of its row and column. If a widget *sticks* to the sides of a column, then it will grow as the column becomes larger. If it *floats,* then blank space will fill the column, as the widget's size remains same size.

For layouts that are not easily specified in terms of the rows and columns of a table, such as those interfaces where the widgets should not be aligned, an arbitrary rectangular region of the table may be treated as an independent sub-table widget, whose grid spacing need not line up with that of the main table. Widgets can be dragged between the main table and a sub-table, or between two sub-tables. The sub-tables can be copied or dragged like ordinary widgets, and can nest to an arbitrary depth.

Each widget has a property sheet containing all of the configurable options for the widget, which includes both the widget properties, and its geometric constraints. For those options in which the underlying **Tk** specifications are overly complex, such as font names, a simplified view is presented to the user, and the interface builder automatically translates the information into the format required by **Tk**. Although a complete user interface may be specified by filling out of the widget property sheets, a toolbar above the layout table allows rapid configuration of common options accessed by way of graphical pull-down menus. In either case, the effect of the option change is shown immediately, and the widget looks exactly as it will in the real application.

The property sheets for each widget are computed dynamically using the reflexive properties of **Tk** to determine which options are applicable to a particular widget. Similarly, upon startup, the user interface

builder automatically determines which **Tk** commands are widgets, and automatically places them on the widget palette. If **Tk** is extended by adding new widgets, the user interface builder will automatically include them on the palette, making them available to the user with no special action required.

Once the user interface form has been completed, it is saved to a file using a simple ASCII format which is interpreted and turned into **Tk** code by a separate **Tcl** procedure. With this technique, the user interface builder never needs to read and interpret **Tk** code, and the translation from the table layout into **Tk** can be changed without affecting the user interface builder. At any time, the user interface under construction may be tested with a single button press that saves it to a file, converts it into **Tk**, and runs it in a separate interpreter.

Design considerations and implementation issues

The primary design decisions involve the tradeoff between performance and portability. The current prototype of the user interface builder is written entirely in **Tcl**. When the Mac and PC ports of **Tk** are available, the user interface builder will run un-modified. If the **Tcl**- only solution was too slow to perform adequately on *middle-of-the-road* computing hardware, then its usefulness would be limited. The prototype carefully caches information to minimize the amount of **Tcl** code that needs to be executed at mouse-motion time. The user interface builder makes use of the new `after idle`[3] construct in **Tk** 4.0 to maintain a cache of relevant information when the application is idle. As a result, the performance is adequate, even on a mid-range "PC".

Prior work

Of the many user interface builders available, two are particularly relevant: Visual Basic[4] and XF.[5] Visual Basic, from Microsoft, provides a simple interface that has given a large number of non-traditional programmers the ability to write graphical user interfaces. Visual Basic provides a layout area to assemble widgets, and property sheets to specify their behavior. This prototype attempts to capture the simplicity that makes Visual Basic so accessible, without forgoing the sophisticated constraint based geometry management capabilities of **Tk**.

XF, by Sven Delmas, is a graphical user interface builder for **Tk**. It provides a forms based interface for building **Tk** applications. XF, however, does not attempt to hide the complexities of **Tk** from the user. Instead, it presents the language features as-is, in a graphical form, and relies on the packer for geometry management, whose geometry management model does not lend itself to a direct manipulation interface.

Summary and conclusions

The familiar user interface paradigms of spreadsheets and drawing programs are combined in a user interface builder for **Tk**. The interface builder supports a direct manipulation interface for laying out user interface elements, yet still takes advantage of the powerful constraint based geometry management capabilities of **Tk**. The current prototype verifies that sophisticated user interfaces may be constructed entirely in **Tcl**, without resorting to "C" code to provide adequate performance, as long as care is taken when coding time critical tasks.

References

1. John Ousterhout, *Tcl and the Tk Toolkit,* Addison Wesley, April, 1994.
2. George Howlett, ‘‘A Table Geometry Manager for the Tk Toolkit,’’ in *1993 Tk/Tcl Workshop Proceedings [online version]*, AT&T Bell Laboratories, 1993.
3. John Ousterhout, *Tk 4.0b3 Reference Manual,* Sun Microsystems Unpublished Draft, February, 1995.
4. Microsoft Corporation, *Visual Basic Programming System for Windows Programmer’s Guide,* Microsoft Corporation, 1993.
5. Sven Delmas, *XF - Design and Implementation of a Programming Environment for Interactive Construction of Graphical User Interfaces*, Technische Universitat Berlin, March, 1993.