The following paper was originally presented at the
Third Annual Tcl/Tk Workshop
sponsored by Unisys, Inc. and USENIX Association
Toronto, Ontario, Canada, July 1995.

A Table-based Layout Editor

George A. Howlett

For more information about USENIX Association contact:

1. Phone: 510 528-8649
2. FAX: 510 548-5738
3. Email: office@usenix.org
4. WWW URL: http://www.usenix.org

A Table-based Layout Editor

George A. Howlett
ghowlett@fast.net

Abstract

This paper describes a user-interface layout editor built upon the BLT library. It offers a drag-and-drop graphical model where new layout designs can be created or existing designs can be tuned.

Introduction

One reason why graphical user interfaces (GUI) are so hard to design and implement is that their requirements are so hard to pinpoint. Initial specifications are usually insufficient or wrong and correct specifications only slowly become apparent through usage. Interface design, implementation, and testing phases become intertwined. The traditional "waterfall" approach to software engineering fails as there is a need to successively iterate through design, implementation, and testing. "The only reliable way to get good interfaces is to iteratively re-design (and therefore re-implement) the interfaces after user-testing, which makes the implementation task even harder."[1]

The *Tk*[2] toolkit's use of a high-level programming language (*Tcl*[3]) has several features which benefit iterative design. Tcl is interpreted so there is no compilation or linking, thus shortening the iterative design cycle. In Tk, all widget resources and toolkit intrinsics (e.g. geometry managers) are available at the Tcl programming level. This means that the entire interface can be dynamically queried and modified from the interpreter while the application is running. In addition, these changes can be made from another program using Tk's "send" facility, which allows Tcl commands to be executed in another interpreter. Successive iterations of the design cycle can be performed without terminating and restarting the application. The best example of such a development tool is the widget resource editor *HierQuery*[4].

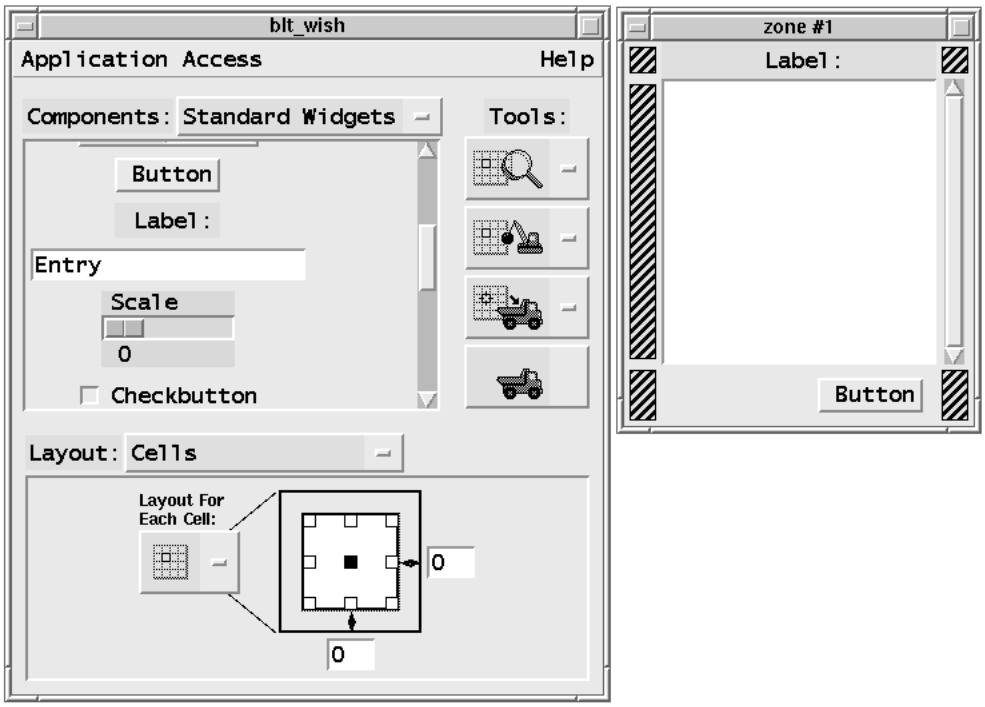
However, the problem with a strictly language-based approach is that not everyone is a programmer. An application's interface may be designed by graphic artists or human-computer interface specialists. Even for programmers, there is a great deal about placement, format, and design of the user interface which must be specified. Re-programming the user interface (via Tk commands) can quickly become tedious. Interface designers and end users need a simpler design mechanism.

The following describes a window layout design tool, built upon the *BLT* library†, called *Ted* (Table EDitor). It attempts to address the iterative design problem by marrying a layout model (table-based geometry manager) with a direct graphical specification (i.e. drag-and-drop) paradigm. New and existing GUI layouts can be manipulated and modified by the mouse, while instantaneous feedback is provided. Designers can create new interfaces by simply dragging and dropping‡ Tk widgets (windows) onto a grid. Like *HierQuery*, *Ted* exploits Tk's powerful send mechanism so that existing designs can be iteratively tuned and modified.

Ted is not (yet) a full-fledged interface builder. There are a number of open problems regarding interface builders. While an interface builder may be useful for generating dialogs and menus, it is not practical for specifying

†. The BLT library is available by anonymous FTP from `ftp.aud.alcatel.com`.

‡. The BLT drag-and-drop command was created by Michael J. McLennan.



interfaces beyond the widget level. For example, a builder would not normally help with the design of the contents of a canvas widget. A bigger problem is how the new interface is incorporated into the existing application. Does the application need to be restarted to view the changes? Is easily is the code generated by the builder merged with the existing application, or is programming necessary? How is consistency maintained?

The much more modest task of *Ted* is to edit window layouts. Its main focus is on *existing* applications, to assist iterative design by making it simple to experiment with different window layouts on running applications. The goal is to provide much more than a graphical representation of the table geometry manager, but a higher level view of layout design to make it easy to create, save, and apply new layout designs. *Ted* is more than a wrapper for table configuration options.

The above figure displays *Ted* and a simple window layout generated from it. Tk widgets have been dragged from the palette (listbox) of simple widgets and dropped on the 3x3 grid. Windows are arranged onto a grid representing the rows and columns of the table. Windows may also span multiple rows or columns. Grid configuration and window alignment is controlled from the lower panel. This too works by drag-and-drop.

Table-based Model

The most interesting aspect of *Ted* is how it embraces the use of the table geometry manager[5] from the start. Many interface builders give the user too much freedom in window placement. They ignore geometry management issues, simply positioning windows at specific screen coordinates. While this is sufficient when it can be assumed that window sizes will never change, the diversity of screen sizes and resolutions today make it unrealistic that “one size will fit all”. In addition, because there is no underlying geometry model, the interface builder does nothing to repair or improve the user’s design.

An alternative approach is to retrofit the placement of a set of windows to a particular layout mode after the user has arranged them. But this can easily be a source of confusion, especially when the layout model fails to accurately reflect the actual design problem. It is important that the model be clear to the interface designer as to how it will affect the design. The model’s layout policies should be intuitive and its design constraints obvious, so that the expected results occur when windows are resized. The layout model at the same time must be flexible and powerful enough to design almost any layout.

Ted utilizes the layout model of the BLT table geometry manager. The table geometry manager has several characteristics which make it highly suitable for this purpose.

1. Gridding

A table-based geometry manager naturally supports gridding. Gridding is a common tool, ubiquitous in graphical design. It is extremely useful for defining relations among graphical components. These relations include balance, symmetry, grouping, and scale. Unlike more complicated "spring" models, gridding should be well understood by most interface designers.

2. Flattened Hierarchy

The logical view of a graphical interface is flat and 2-dimensional. It is much easier to work with a layout design which is essentially flat. The table layout model makes it easy to align windows without resorting to creating layers of window hierarchy (e.g. a frame used to position windows in a column).

3. Layout Description

The simplicity of the table layout model has several advantages. Layouts can be described compactly. The entire table arrangement can be described in a single Tcl command. Parts of the table can be reconfigured without re-specifying the entire layout description. This is important because the table layout may change many times during an editing session.

Ted saves "undo" and "redo" information for each editing operation so it is necessary that the state of the table is easily queried and the description can be compactly saved. The table geometry manager has several options to query how the table is configured and what windows are arranged inside of it. *Ted* extracts this information to construct each view of the table grid.

4. Order Independence

The order in which windows are packed and repacked should not affect their final layout. The packing order of windows may change many times in the process of editing a layout. It is too difficult for the end user to redesign a layout while considering how reordering slave windows will change the packer's eventual layout. This was the major reason why the Tk packer geometry model was not used.

Editor Output

The output of *Ted* is Tcl code. Since the major focus of *Ted* is for redesign of the layouts of existing applications, it can not be assumed that they will contain any extra code to process layout information. *Ted* interacts with the application's table geometry manager to update the layout.

An open issue in *Ted* is how to merge the new Tcl code with the application when the new interface is finally saved. Tcl does not maintain the line numbers of the code as it is executed. It is not possible to know where in what file the table geometry manager was created. *Ted* saves the Tcl code in a separate file. It is left for the user to integrate the saved piece of code with the original files.

Conclusion

Ted is an layout design tool which provides a graphical interface to create layout designs and edit existing table-based designs. Its modest goal is to provide a platform for iterative design of window (widget) layouts. It embraces a table-based geometry model which combines simple and intuitive layout policy with the power and flexibility to describe most layout designs. *Ted* will hopefully evolve into a full-fledged interface builder.

References

1. B. A. Myers, *User Interface Software Tools*, 1994.
2. J. K. Ousterhout, *An X11 Toolkit Based On the Tcl Language*, Proceedings of the 1991 Winter USENIX Conference, 1991, pp. 105-115.
3. J. K. Ousterhout, *Tcl: An Embeddable Command Language*, Proceedings of the 1990 Winter USENIX Conference, 1990, pp. 133-146.
4. D. Richardson, *Interactively Configuring Tk-based Applications*, Proceedings of the 1994 Tcl/Tk Work shop, New Orleans LA, pp. 21-22.
5. G. A. Howlett, *A Table Geometry Manager for the Tk Toolkit*, Proceedings of the 1993 Tcl/Tk Workshop, University of California at Berkeley, pp. 27-30.