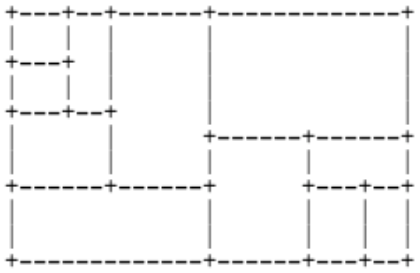
The MTI Panemanager Widget

A 2-d paned window for user configurable U/I

Brian Griffin

Mentor Graphics Corporation
8005 SW Boeckman Road
Wilsionville, Oregon 97070
503-685-0850



Abstract

This paper will present the basic functions of the Pane Manager widget, a megawidget implemented in incrTcl used as an application framework. The added features, such as Pane Docking, and Pane Zoom, make this a powerful tool for application frame development. Implementation details, problems encountered, and future development will also be discussed.

Keywords

Tcl, Tk, [incrTcl], TIP, mega-widgets, user interface.

1. Introduction

Several years ago we were presented with the problem of having too many toplevel windows in our application, cluttering the desktop, and management of these windows was difficult for the user. The decision was made to take an approach seen in other IDE-like tools where information was provided in tiled panes within the main application window. The large number of windows would be managed by controlling visibility and through the use of tabs. Also needed was a way to easily rearrange panes since the application tasks required a different focus of attention at different times. That coupled with user preferences required a flexible way to manage these configurations.

Our first approach used a grid that defined a maximum of 5 rows and any number of columns. Panes could be placed in a number of configurations, but this turned out to be too limiting. The second attempt uses nested Panedwindows where sets of panes can be grouped into larger panes. The Panemanager widget, presented here, is written in incrTcl and has an interface similar to the Tk Panedwindow widget.

2. Background

Model***Sim***® is a software tool in the Electronic Design Automation (EDA) industry used by digital hardware design engineers for design verification. The graphical interface for this tool is written using Tcl/Tk.

Hardware verification analysis involves many complex views since the problem is multi-dimensional. The user interface is very similar to software debuggers, such as Microsoft® Visual Studio® or the GNU Data Display Debugger (ddd), which includes features like source code viewing, structure browsing, and data viewing. In addition to traditional software debugging tools, hardware simulation involves concurrent processing and temporal sequences that are unique to the hardware debug process. This adds another dimension to the problem. Additional tools such as code coverage analysis and performance profiling add even more views to the U/I.

3. Application frame survey

Most applications limit their scope to a single activity requiring only one or two views. Editor applications typically have a main work area surrounded by tool-bars and palettes. Browsers have a main view with a possible sidebar of tool-like features. These tools may provide a tab mechanism for managing multiple views, but typically only a single view is necessary at any given point.

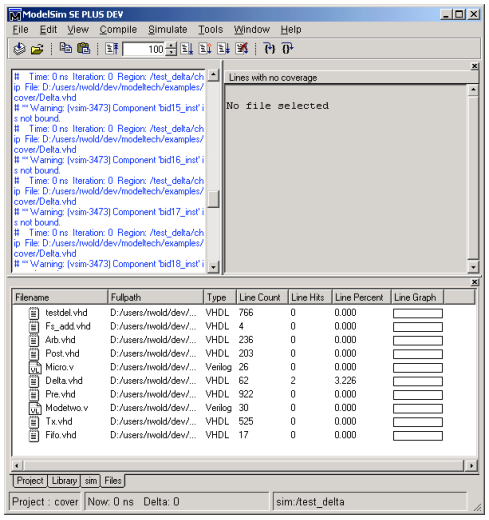
Mail applications usually involve 3 common views on the display at one time, a browser for organizing mail, a current folder content list, and current letter view.

More complex editors such as a 3D graphics editor may involve multiple views in addition to the tool palettes. Software used for analysis as in financial, scientific, and engineering areas require the ability to monitor multiple values simultaneously. Software debuggers typically monitor source code, memory contents, variable values, stack values, and register values.

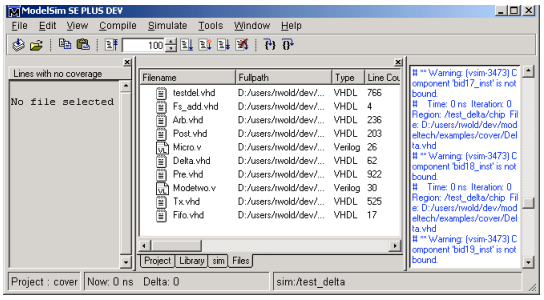
ModelSim is essentially a software debugger for hardware description languages (HDL), with one added dimension of complexity: concurrent execution.

4. Simple Grid

We decided to use Visual Studio 6 as a model for the single window U/I. We wanted to implement a similar interface, but also wanted it to be more flexible. Our initial approach settled on a grid of 5 rows with any number of panes in each row. The main feature of this interface is the ability for the user to rearrange the panes in whichever way was most useful as shown in Figure 1.



(a)



(b)

Figure 1. Grid based window pane manager, using two rows (a), or all in one row (b).

The implementation used the Iwidgets Panedwindow [1], packing 5 of these into rows in the main window. Using the Panedwindow megawidget provided dividers between window panes to allow resizing of the windows. Each window pane is then given a row and column position and is wrapped in a frame with a title bar at the top. This title bar can be dragged with the mouse to a different row/column location. The window locations are saved to a user preference file so that the layout is retained between invocations of the application.

This system worked well and provided all the features we were looking for. This approach was chosen by other IDEs because the environment centers around one primary pane, i.e., the source editor. With ModelSim, some users perceive more than one primary pane, or the primary pane changes. However, there were limitations in layouts, namely that there were only 5 rows and no window could span multiple rows. At this point, only a few of the many windows had thus far been moved into the main window and these limitations appeared to hinder effective use of the one window model once all the windows were brought in.

5. Extending Panedwindows

At the same time we were looking for an alternative layout model, we also moved to Tcl/Tk 8.4. This made available the Tk Panedwindow widget [2]. This widget provided better look and performance over the Iwidgets version. However, there were some limitations to the widget that needed to be addressed before it could be used as an effective replacement. What was missing was the ability to hide panes. This feature is used to control visibility of panes in the layout without having to add and remove them. Instead, all windows are added to the layout and then hidden when the user "closes" the window. This feature was addressed through TIP #179: Add -hide Option to Panedwindow Widget [3].

Another behavior of the Tk Panedwindow that was problematic was the stretch behavior. The Panedwindow, being a geometry manager, must decide how to allocate space to the various widgets under its control. Space is allocated to each widget as requested by the widget. Any remaining space, for example, if the geometry of the Panedwindow is larger than the sum of the required spaces, is allocated to the last window in the list. In most simple cases, this is the appropriate behavior. However, with complex layouts (multiple rows and columns) this behavior is not always what is expected. Sometimes the best behavior is to allocate the space evenly among all the windows, or proportionally. This led to TIP #177: Add -stretch Option to Panedwindow Widget [4]. The stretch option gives control to each pane to participate in the distribution of extra space. The values "always" and "never" enable or disable participation regardless of the window's position in the list, while the values "first", "last", and "middle" conditionally enable participation depending on the relative position of the window. The default behavior, as defined by the original Panedwindow behavior, is "last".

6. Nested panes

In order to address the layout limitations using a grid arrangement, the Panemanager uses an approach in which each pane is also a Panedwindow. For each window pane there is an outer Panedwindow and an inner Panedwindow with the opposite orientation as shown in Figure 2.

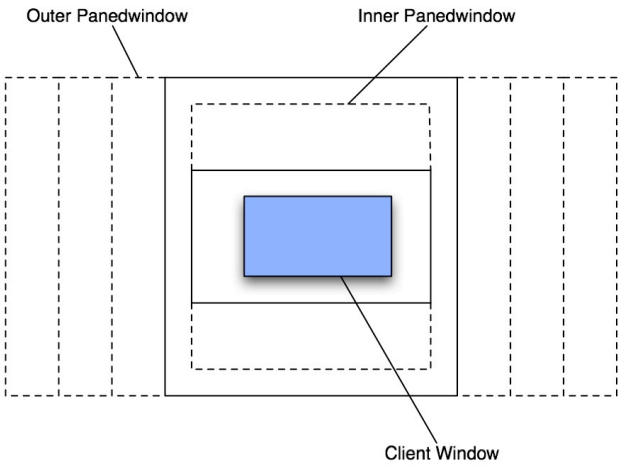


Figure 2. Nested Panedwindows

This allows adding a new window to the left, right, above, or below an existing window. The Panedwindow **add** command was modified to require 3 arguments: the window to be added, a target window within the Panemanager, and a location; above, below, left, or right. The remaining arguments are the same as the Tk Panedwindow **add** command.

pathName **add** *window target where ?arg arg ...?*

In addition to these basic placement locations, it is also possible to split a target window space so that the new window shares the space. To do this, the Panemanager replaces the current pane with a new Panedwindow, and then adds the target window and the new window to the Panedwindow. The *where* argument controls the location of the new window by specifying one of n, s, e, or w. Another way to think about this is as placing the new window inside (n, s, e, w) or outside (above, below, left, right) the target window cell (see Figure 3.) The e-w or n-s locations are redundant with the right-left or above-below locations, depending on orientation.

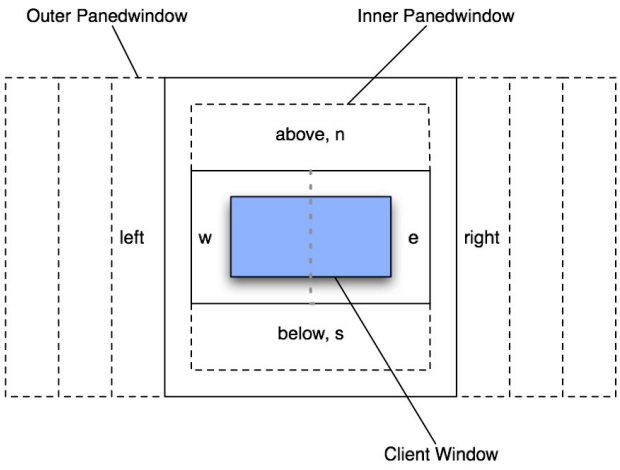
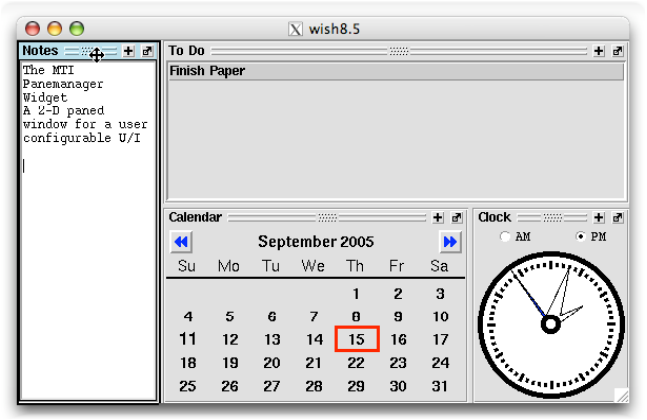


Figure 3. Placement locations.

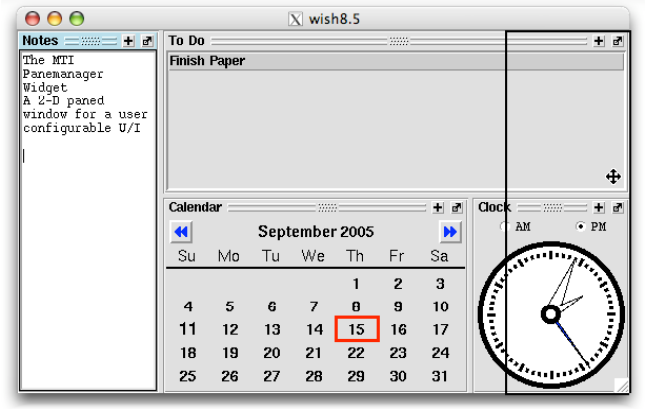
7. Pane placement

An application uses the Panemanager by adding all the available windows up front, and then hides the ones that may not be appropriate or that are not part of the initial default conditions. When the time comes, usually under user control, the hidden panes are made visible by simply modifying the **-hide** option for the pane.

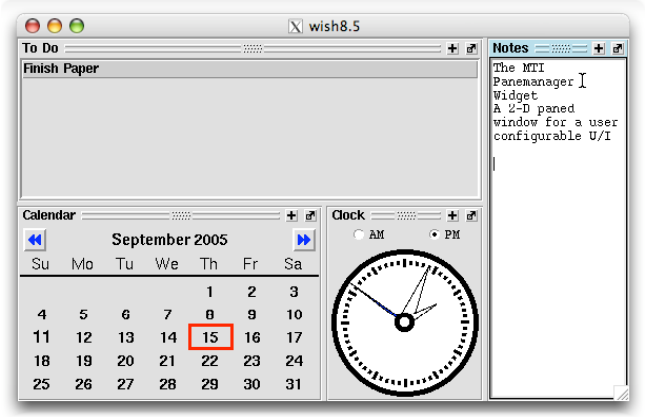
The initial default layout may not be to the user's liking. To make it easy to rearrange the windows, a title bar is given to each pane and these bars are used to drag the windows to new locations. When a drag operation is started, a simple wire frame is drawn on the screen to illustrate the target location of the window. Feedback is given to the user by snapping the wire frame to the potential target locations, illustrated in Figure 4.



(a)



(b)



(c)

Figure 4. Wire frame drag targets shown with the starting position (a), the target location (b), and after the drop (c).

Implementing this interactive feature is simply a matter of removing the window from the Panemanager, then adding it back in the new location. Simple, except for determining the *where* argument for the **add** command. This is done by walking through the list of visible panes and computing a target drop zone for each location: above, below, left, right, n, s, e, or w. Some of these locations are redundant depending on the orientation of the Panedwindow containing the target window. The drop zones are

then drawn on a canvas as rectangles and an associative map is created, mapping each rectangle tag to a Panedwindow and location (see Figure 5.) During the drag (mouse motion) operation, the mouse location is mapped to the canvas to obtain the nearest rectangle. The rectangle's geometry is then used to define the wire frame's geometry. This makes the wire frame calculations simple and fast. When dropped (button release), the operation can be completed since all the necessary arguments have already been calculated. One final required step, however, is clean up. These operations can leave empty Panedwindows, so it is necessary to do some cleanup and coalesce or flatten these unnecessary widgets. Note that the canvas is never mapped to the display, so it remains hidden from view.

Any and all possible arrangements of *n* rectangles can be accomplished, however, it may take multiple moves to finalize a given layout. The intermediate steps are necessary to cause a change in the Panedwindow orientation.

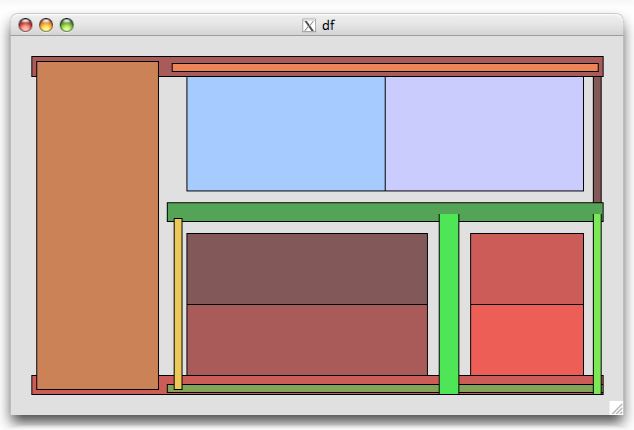


Figure 5. Drag map canvas showing drop zones of Figure 4(a).

8. Maximize & Undocking

Two powerful features of the Panemanager are window maximize and undocking. These operations are triggered by a click of a button on the window pane's title bar.

Window *maximize* is the ability to hide all the visible panes except for the "maximized" window. The net effect is that the targeted window now occupies the entire application frame. A second click will restore the windows to the previous layout. This is actually implemented by un-mapping the Panemanager's topmost frame and mapping the window pane in it's place. A temporary placeholder is put into the pane's location in the Panedwindow to hold its original location and attributes.

Docking and undocking of windows moves a window pane in and out of the main frame, making it a separate toplevel window when moved out. This can be achieved with either a toggle button on the title bar, or by dragging the window outside the main frame. This feature is implemented by using TIP #125 [5] which can convert a frame to a toplevel window (and visa-versa). The Panemanager maintains a placeholder for the absent window so when the window is later docked, it will return to its original location. The placeholder is not visible, but holds the location and attributes in the list of managed panes.

9. The Paneframe widget

The Paneframe widget works in conjunction with the Panemanager to handle various operations. It consists of a title bar and a childsite frame for locating window contents. The title bar contains, optionally, a title string, a drag handle, and up to three action buttons: close, maximize, and undock. The close button is used to hide or close the window depending on how the application chooses to use it. The maximize button performs the maximize operation to fill the window frame with the pane, and the undock button performs the undocking operation, removing the pane from the window and making it a separate toplevel. The drag handle is a small area in the center of the title bar used to drag the pane to a new location either inside the main window, or outside to undock the window. The drag handle was added later when it was discovered that using the entire title bar for dragging resulted in many unwanted or unexpected window relocations.

Another duty of the Paneframe, with its title bar, is to identify the active window. Like the highlight ring in the Tk widgets, the title bar will change color when the window pane has focus, thus indicating the active window in a fashion similar to desktop window managers. The Paneframe widget also supports focus redirection. When a window pane is made active by clicking on the title bar, the B1 event will redirect focus to an inner widget identified by the application. This is similar to what Tk does when focus moves from one toplevel window to another; Tk remembers which internal widget had focus last and will return focus to that inner window when the toplevel window gains focus.

10. Persistence

What good are all this fancy wiz-bang layout features if the application can't restore the layout the next time it's run? The Panemanager provides a way to serialize the layout into a list that can be saved by the application and used later to recreate the layout. The restore operation requires that all the window panes exist beforehand. It does not provide any mechanism for recreating the individual window contents. This task is left up to the application. However, the ability to serialize a layout means an application can provide multiple pre-defined layouts that can be applied at the click of a button or menu pick.

The serialized form is a nested list where each entry is either a window with attributes, or another list. The nested list is itself a list of windows or another list. Each nesting represents a nested Panedwindow with a rotated orientation. The initial element of the serialized layout defines the initial orientation. The window attributes include the docked window size, undocked geometry, and Panedwindow configure options, including the **-hide** option.

The size value defines the sash location for the window. Percentages are also accepted. This is useful for defining an initial default layout for an application that is not dependent on a given screen size. The serialize process, however, does not generate percentage based sizes. An example serialized layout is shown in Figure 6. This layout represents the display shown in Figure 4(c).

```
horizontal {
  {
    .mp.f2 {-minsize 40 -stretch never} 157 na
    {
      .mp.f3 {-minsize 40} 292 na
      .mp.f4 {-minsize 40} 444 na
    } {-minsize 40 -stretch always} 345 na
  } {-minsize 40 -stretch always} 446 na
  .mp.f1 {-minsize 40 -stretch always} 582 na
}
```

Figure 6. Example serialized layout.

11. Problems and limitations

There are a few remaining problems with this megawidget. One issue is performance. Loading a layout currently requires multiple redraws as windows and sash locations settle out. The forced redraws have been the only way to get the sash locations to restore properly. Tk is designed to work best if geometry is determined from the bottom up. The Panedwindow widget assumes that initial sash placement is solely based on requested slave window geometry. The Panemanager is designed to manage pane sizes in a top down fashion. We are still looking for the right solution to this problem.

When rearranging windows, they sometimes end up with unexpected sizes. The complexity of both removing a window from one location and placing it in a different location resulted in several windows changing size all at once.

Ideally, the undocking feature would work in conjunction with the window manager so that the docked and undocked window title bars were the same or similar. Also, a window move using the window managers title bar would recognize a drop over the main window causing a dock to occur. However, there is currently no support for this. Consequently, the undocked windows end up with a double title bar, which does not look very professional. This feature would be more difficult to implement than just handling each platform because of the plethora of window managers. The good news is that an application does not have to use the Paneframe class for the docked windows. The Panemanager can manage any widget, so an application can use a different paradigm for the docking/undocking maneuver.

12. Future work

Most recently, we have been contemplating overlaying panes and introducing tabs, in other words, making it a tabbed notebook. A tabbed notebook would be created when one pane is dropped onto the center of another. When a pane is dragged away, its tab is removed and when only one pane is left, the tabs disappear as well.

The performance and layout problems could be easily addressed by writing the widget in C, using the Panedwindow widget as a starting point. This would eliminate the need to manage the Panedwindow widgets, as the window pane management would be handled directly.

References

[1] Ulferts, Mark L. [incr Widgets] - panedwindow manpage <http://incrtcl.sourceforge.net/iwidgets/manpages/iwidgets3.0/panedwindow.n.html>. 10 September 2005.

[2] Melski, Eric. Paned Window Tk Widget. 04 July 2001. <http://www.tcl.tk/cgi-bin/tct/top/41.html>. 10 September 2005.

[3] Griffin, Brian S. Add -hide Option to panedwindow Widget. 22 March 2004 <http://www.tcl.tk/cgi-bin/tct/tip/179.html>. 10 September 2005.

[4] Griffin, Brian S. Add -stretch Option to panedwindow Widget. 17 March 2004 <http://www.tcl.tk/cgi-bin/tct/tip/177.html>. 10 September 2005.

[5] Griffin, Brian S. Converting between Frame and Toplevel Windows. 20 January 2003 <http://www.tcl.tk/cgi-bin/tct/tip/125.html>. 10 September 2005.