

WebWiseTclTk: A Safe-Tcl/Tk-based Toolkit Enhanced for the World Wide Web

Hemang Lavana Franc Brglez

CBL (Collaborative Benchmarking Lab), Dept. of Computer Science, Box 7550

NC State University, Raleigh, NC 27695, USA

<http://www.cbl.ncsu.edu/>

Abstract

The WebWiseTclTk toolkit is an enhancement of the existing feature set of Safe-Tcl and Safe-Tk that does not compromise security. The toolkit re-defines the functionality of the auto_load mechanism in Tcl such that it works for packages located anywhere on the World Wide Web. It also re-introduces several commands not available in Safe-Tk such as toplevel and menu to provide a much richer feature set of Tk commands. The toolkit is written entirely in Safe-Tcl/Tk and uses the home policy for running applications as Tcl-plugins.

The toolkit supports (1) creation of new Web-based Tcl applications with greatly enhanced functionality, and (2) migration of existing Tcl applications to the Web by merely writing an encapsulation script. We demonstrate the capabilities of the WebWiseTclTk toolkit by readily creating an encapsulation script for Web-based execution of the Tk Widget Demonstrations, distributed with the core Tcl/Tk.

Keywords: plugins, Web browsers, security, scripting, encapsulation, GUI.

1 Introduction

The last few years have seen an explosive growth of the usage of Tcl (Tool Command Language) [1, 2, 3] and its popularity can be easily gauged by the large number of postings in the Tcl newsgroup `comp.lang.tcl`. Scripting languages such as Tcl are designed for ‘gluing’ applications and encourage rapid application development as compared to system programming languages, and hence are very important for applications of the future [4]. The emergence of organizations such as the recently formed *Scriptics* [5] and the *Tcl/Tk Consortium* [6], focusing entirely on scripting tools, applications and services, is an example of this trend.

The maturity and robustness of Tcl/Tk provides a

new opportunity to support creation and presentation of multimedia content on the WWW. Tcl-plugin [7, 8] is an example of an elegant solution for embedding Tcl/Tk applications for ready access inside the Web browser. In addition, the Tcl-plugin supports an excellent mechanism for security of client hosts using a padded cell approach [9]. The default security policy prohibits Tcl applets (*tclets*) from running other programs, accessing the file system, and creating toplevel windows (including menus), thereby giving the client hosts a high level of confidence when executing *tclets*. However, such restrictions limit the scope of the Tcl applications executed inside a Web-browser.

Our initial experience with Tcl/Tk, predating the phenomenal growth of WWW, was motivated by the need to develop a user-friendly and versatile environment to support user-reconfiguration of complex workflows that execute heterogeneous programs and data for the design of experiments in VLSI CAD. This environment, called REUBEN (for reusable and reconfigurable benchmarking environment), was implemented entirely in *Tcl/Tk* [1] and *Expect* [10]. In essence, it provides the user with the ability to create directed dependency graphs as workflows of data, program, decision, and workflow nodes. Data and programs can reside anywhere on the Internet, and execution of all nodes can be scheduled automatically, regardless of the data-dependent cycles in the graph. In its final form, the workflows in REUBEN can be multi-cast to several collaborating sites, recorded, and played-back for re-execution. An example of REUBEN environment to support a number of distributed and heterogeneous tasks in a VLSI CAD workflow is illustrated in Figure 1. More details are available in [11, 12, 13].

Migration of large applications, such as REUBEN, to the Web is not easy if highest level of confidence in terms of security is desired. This is especially true, because toplevel windows and menus are essential in such applications. One solution could be to use Jacl [14], an interpreter to run Tcl scripts in a Java environment. Unfortunately, Jacl does not

This research was supported by contracts from the Semiconductor Research Corporation (94-DJ-553), SEMATECH (94-DJ-800), and DARPA/ARO (P-3316-EL/DAAH04-94-G-2080 and DAAG55-97-1-0345).

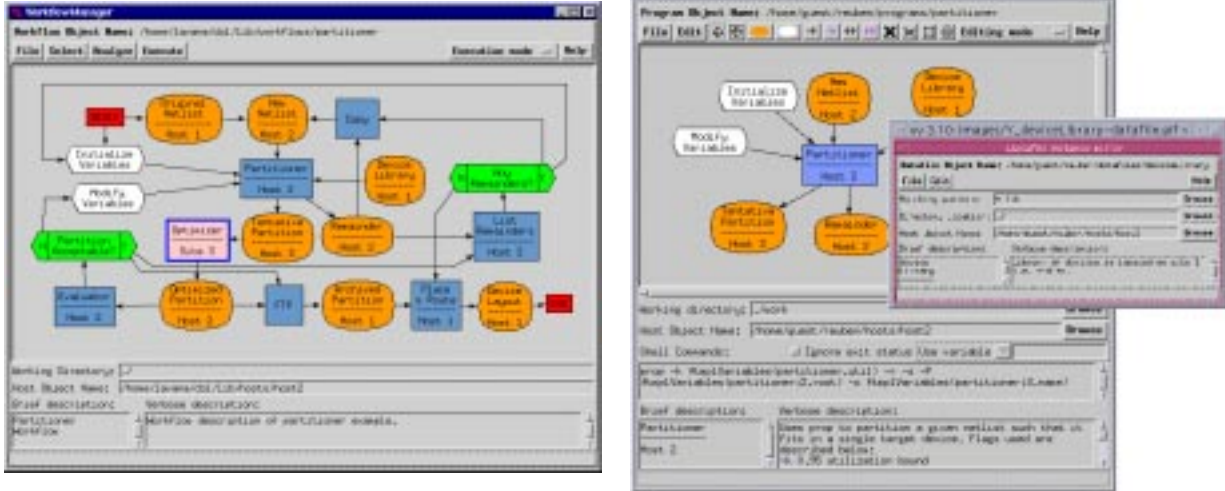


Fig. 1. REUBEN environment consisting of several windows.

yet contain the entire feature set of Tcl, including namespaces and Tk. The `WebWiseTclTk` toolkit provides an easy solution for the migration of existing Tcl applications to the Web. Minimal changes are required in the original application. Our approach uses an encapsulating script to call the main script of the original application.

This paper is organized into the following sections: (2) motivation; (3) `WebWiseTclTk` architecture; (4) implementation of `WebWiseTk`; (5) implementation of `WebWiseTcl`; (6) user's guide; (7) programmer's guide; (8) software status and availability; and (9) conclusions.

2 Motivation

A large application, written in Tcl, typically consists of a short main script and a library of support scripts. Applications start up quickly by invoking the main script. As new features are accessed, the code that implements them is loaded automatically, using the `auto_load` mechanism available in Tcl. A complex environment such as REUBEN, described earlier and illustrated in Figure 1, requires that a number of windows be created during its runtime.

The Tcl-plugin, based on Safe-Tcl, restricts running such large applications inside a Web-browser. A few of these restrictions are listed below:

- `Auto_load` scheme fails, unless the application package is installed on the client host. Another alternative is to merge all the scripts in the application into a single script which can be downloaded as a *tclet*.
- Applications are restricted to a single window since the command `topLevel` is not available in Safe-

Tk and new windows cannot be created.

- Menu widgets are also disabled in Safe-Tk.
- *Tclets* do not have access to standard input and standard output.

The Tcl-plugin supports multiple security policies so that the *tclets* can perform any of the functionality described above. However, this requires every client host to devise and customize their security policies for every application before accessing these as *tclets*.

It is desirable that the Tcl applications be easily translated into *tclets* and made readily available on the World Wide Web:

- without requiring any major changes in the application code, and
- without requiring any sophisticated security policy to run the *tclet*.

We have developed the `WebWiseTclTk` toolkit as an enhancement to the Tcl-plugin that makes use of the *home policy* only. The *home policy* is, by default, enabled in the Tcl-plugin and hence applications using `WebWiseTclTk` do not require the host clients to modify their existing security policies.

We decided to use the Tk widget demonstrations, distributed with the core Tcl/Tk, as a test-bench for testing the `WebWiseTclTk` toolkit. We chose to translate these demos for the World Wide Web because they cover most of the commands of the core Tcl/Tk that are otherwise unavailable in Safe-Tcl/Tk. Figure 2 shows the result of posting these demos on the Web and executing them on a host client as a *tclet* using the Netscape browser. We invite users to try out this demo and send us comments on its features and performance.

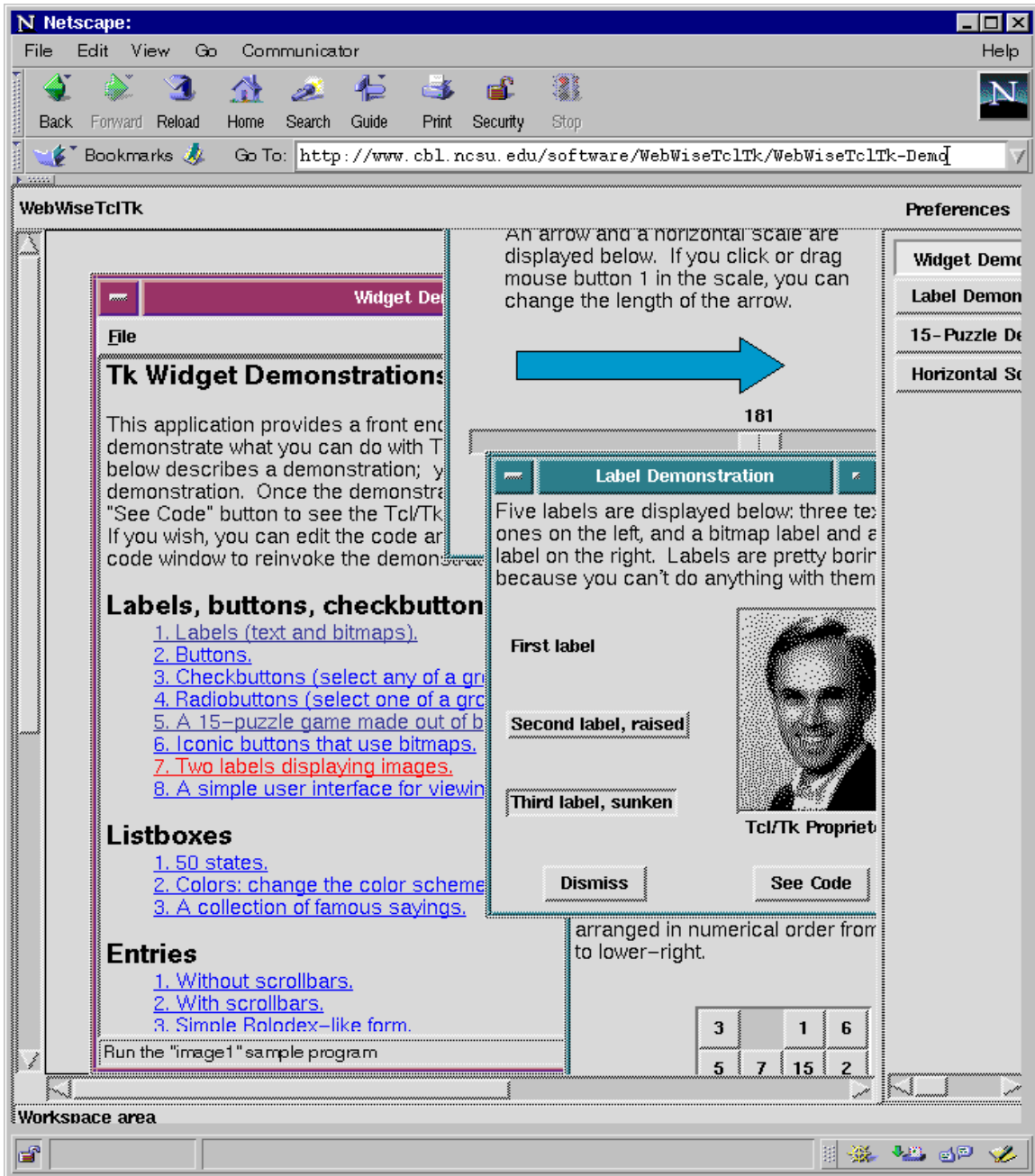


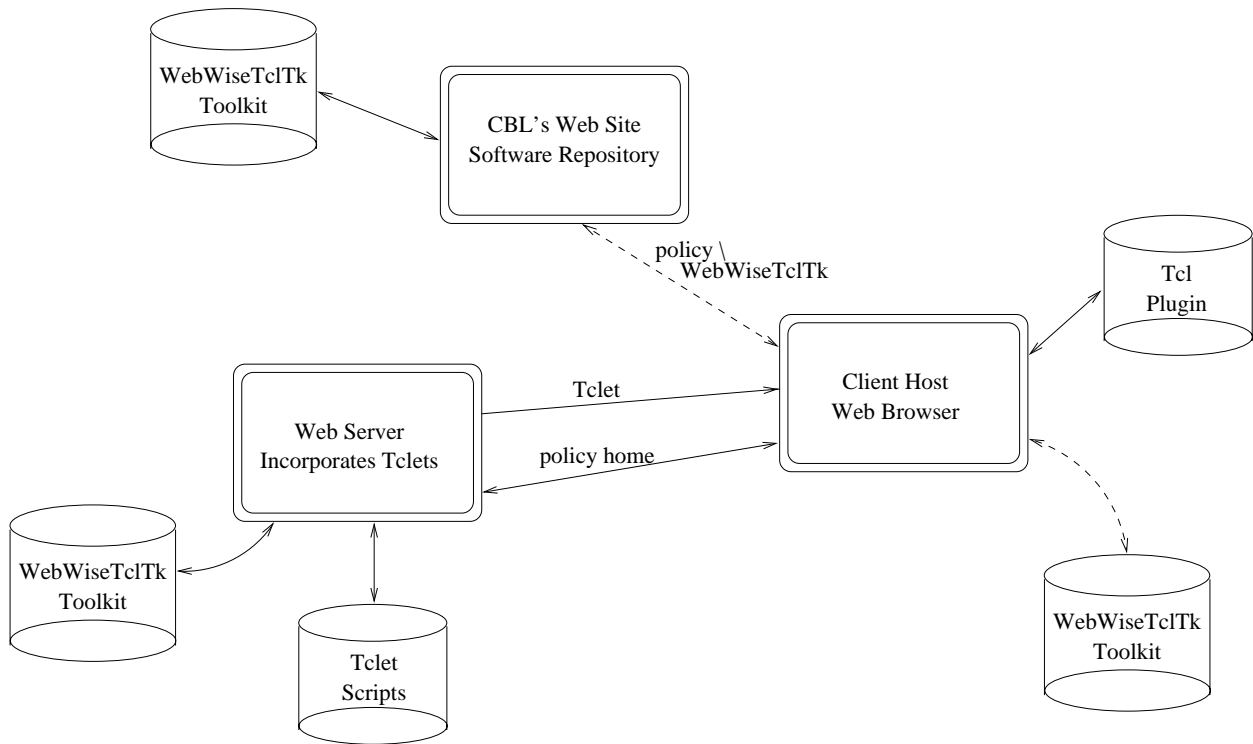
Fig. 2. Tcl/Tk widget demos on the Web.

3 Architecture

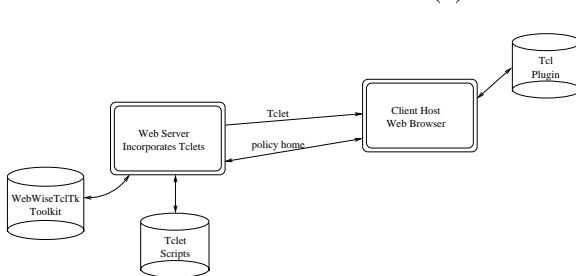
The toolkit WebWiseTclTk consists of two parts: (1) WebWiseTcl which is an enhancement for Safe-Tcl and is useful for applications that do not require display, and (2) WebWiseTk which is an enhancement for Safe-Tk for applications requiring display. The toolkit itself consists of several smaller scripts and uses the modified auto_load mechanism designed for WebWiseTclTk.

Figure 3(a) shows the general architecture that implements the auto_load mechanism. Special cases of the generalized architecture are shown in Figures 3(b), (c) and (d) and described below:

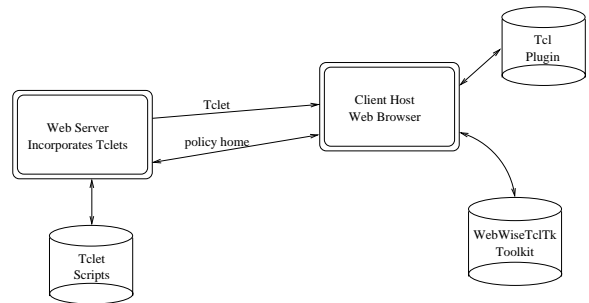
1. Typical client host, downloading a *tcllet* from a Web server, has only the Tcl-plugin installed for its Web-browser. The server site provides not only the *tcllet scripts* but also the WebWiseTclTk toolkit as shown in Figure 3(b). The client host downloads the main script for the *tcllet* which requests to use



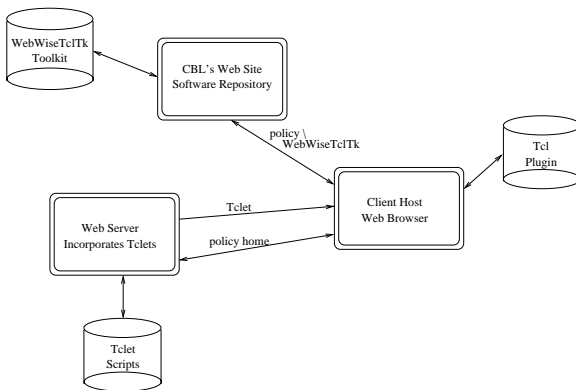
(a) Generalized architecture.



(b) WebWiseTclTk toolkit on the Web server.



(c) WebWiseTclTk toolkit on the client host.



(d) WebWiseTclTk toolkit on the CBL site.

Typical scenario of a *tcl* execution:

1. Client host visits the Web server.
2. Main *tcl* script is downloaded to client host.
3. WebWiseTclTk toolkit is loaded and initialized.
4. Main *tcl* script executes.
5. Individual *tcl* scripts are downloaded as and when required.

Fig. 3. Architecture for WebWiseTclTk toolkit.

the *home policy*. If the client host has not disabled the *home policy*, then the main script downloads the initialization script of the `WebWiseTclTk` toolkit from the server site. Once the initialization has completed, the `auto_load` mechanism is modified to dynamically download the remaining scripts of the application as and when needed during execution of the *tcllet*.

2. In the second case, shown in Figure 3(c), the client host has locally installed the `WebWiseTclTk` toolkit. The main script of the downloaded *tcllet* uses the locally available toolkit and visits the server site only to retrieve its other scripts. Thus, this results in faster execution of the *tcllet* code.

3. In the third case, shown in Figure 3(d), the `WebWiseTclTk` toolkit is neither available on the server site nor is it installed on the client host. It is available at the software repository site at CBL. This requires the client host to install a special *WebWiseTclTk policy* that allows the *tcllets* to download scripts not only from its server site but to also download the toolkit from the CBL site. This mechanism has the advantage of always using the latest version of the `WebWiseTclTk` toolkit.

The generalized architecture allows the main *tcllet* script to dynamically use one of the above three mechanisms, based on the configuration of the client host.

We next describe the implementation details of the two packages `WebWiseTk` and `WebWiseTcl`.

4 WebWiseTk

Several Tk commands are hidden in Safe-Tk to prevent *denial of service* attacks against the host system. This, however, limits the scope of the Tcl-plugin to very simple applications consisting of a single window only.

We propose to overcome these limitations as follows:

- re-introduce several of the hidden commands in Safe-Tk,
- use existing commands that are already available in Safe-Tk, to define re-introduced commands,
- change the implications of a few commands, such as "`grab -local`" and "`grab -global`".

The following sub-sections describe the methodology used for implementation of the `WebWiseTk` toolkit.

Layout. Figure 4 shows the layout window of the `WebWiseTk` toolkit. It consists of two main widget frames:

- A canvas widget is used to display several toplevel windows that may be created during the execution of a *tcllet*. If the toplevel window is larger than the visible canvas area, then scrollbars may be

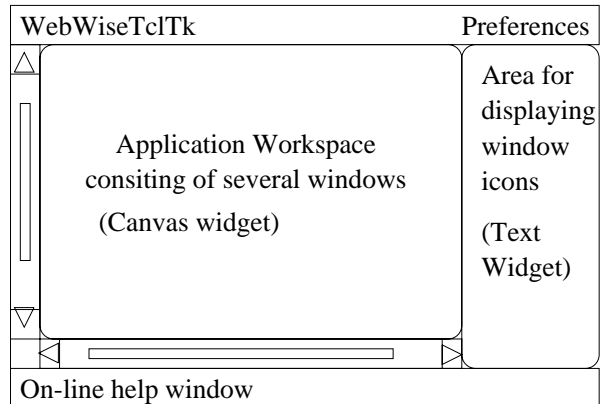


Fig. 4. Layout window of `WebWiseTk` toolkit.

used to display its hidden area. The scrollregion of the canvas is initialized to 1000×1000 pixels, but may be resized using the preferences option on the top right corner.

- A text widget is used to display button icons for all windows that have been created, including those windows that have been iconified.

Several other widgets are shown, such as the On-line help window, scrollbars for the canvas and text widgets, and preferences to configure the size of the canvas widget.

Toplevel. The ability to create a detached window, as provided by the command `toplevel`, is very useful for GUI applications of even moderate complexity. We define a procedure called `toplevel` which makes use of the command `frame` to create a detached window and display it on the canvas widget. For every toplevel window, a set of several frames is created, as shown in Figure 5.

This gives the look and feel of a real window that would have, otherwise, been created by the window manager of the local host system. The frames are laid out using the `grid` geometry manager. Each frame serves a special purpose:

- The frames on the border have a default color and an active color which is highlighted whenever the mouse cursor moves inside a window. This helps the user to identify the window that is currently active. These frames are also useful for changing the size of the window.
- The next set of frames, just below the resizing frame on the top, provide several functions related to the window, such as `kill`, `iconify`, `maximize/restore size` or display the title of the window.
- A main frame is created in the center corresponding to each toplevel window. All subsequent child windows of the toplevel are packed into this frame.

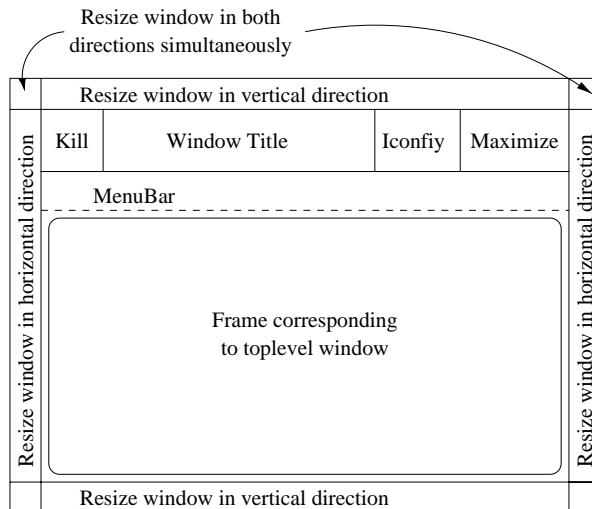


Fig. 5. Implementation of a toplevel window using frames.

- If the toplevel window has a Menu Bar associated with it, then the menu items are packed into a frame just above the main frame.

Having created these sets of frames, they are then packed onto the canvas widget in the application workspace area in Figure 4 by creating a canvas item of type *window*. This results in a restriction that the pathname of the window must either be a child of the canvas widget or a child of some ancestor of the canvas widget. Hence, the window names of every toplevel command is translated to a new window name that is a child of the canvas widget. For example, a new toplevel window called ".w" is translated to a new window name called ".c.1.w", where ".c" is the name of the canvas widget and ".c.1" is the name of a unique frame created for encapsulation of the new toplevel window. In addition, we create several bindings to manage and interact with the command `wm`, described next. A `<Destroy>` binding is also associated with every toplevel frame so that the entire set of frames is destroyed whenever the toplevel window is destroyed.

Wm. The window manager command `wm` needs to be defined as a procedure which manages the various attributes of the window created using the procedure `toplevel` described earlier. It can be used to change the title of the window, to iconify/de-iconify the window, or to return the state of the window.

Grab. An indefinite global `grab` performed by a *tclet* will result in a *denial of service* attack since all the input from the terminal would be re-directed to the *tclet* forever. But, if we re-define the implication of a global `grab` such that it affects only the windows created by the *tclet*, then it can be considered to be

safe.

Thus, the command "`grab -local $win`", as defined here, results in grabbing of a single window within the *tclet* code and the command "`grab -global $win`" results in a grab across all the windows within the *tclet*. This effect of grab can be implemented by associating a new class of bind called `WebWiseTclTk` with every window in the *tclet*, as follows:

```
bindtags $w [linsert [bindtags $w] 0 WebWiseTclTk]
```

Initially, the class `WebWiseTclTk` has no bind scripts associated with any of the events. Whenever a `grab` is performed on a window, a bind script is created for each event sequence that redirects the event to the grabbed window. The event `generate` command is used to process the event in the grabbed window. Figure 6 shows a script that achieves a global grab for a specific window. A grab is re-

```
foreach seq $AllEventSequences {
  bind WebWiseTclTk $seq "
    # Redirect events to grab window
    if {![string match $win* %W] &&
        \[wininfo exists $win]} {
      event generate $win $seq
      break
    }; # End of if stmt
  "
}; # End of foreach loop
```

Fig. 6. Script to achieve a global grab on a window.

leased by re-initializing the bind scripts for the class `WebWiseTclTk` to null. Variables are used to store the state of the `grab` command and return appropriate values for queries such as "`grab current`" and "`grab status`".

Menus. Menu widgets are as important as any toplevel widgets in any GUI applications, since they allow the user to invoke a list of one-line entries as and when required. The structural layout of the menu widgets created using `frame` and other Tk commands is shown in Figure 7. The command `menu` creates a toplevel frame and different types of widgets are added inside this frame: `button` widgets for command entries, `checkboxbutton` widgets for `check button` entries, `radiobutton` widgets for `radio button` entries and `menubutton` widgets for `cascaded menu` entries. Separator entries are created using `frame` widgets as shown in Figure 7.

This structure is hidden from the display until the user clicks on the menu button at the top. The implementation of the command `grab`, as described earlier, is important and allows us to post the menu widget frame whenever the user clicks on the menu button. As the user moves the cursor over differ-

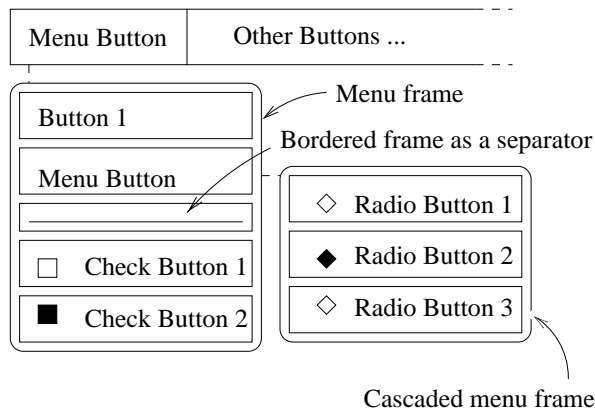


Fig. 7. Implementation of the command menu.

ent widgets in the menu frame, each widget is highlighted and the associated command invoked if necessary. Clicking on the cascaded entry results in the posting of another menu frame with its associated entries.

If the menu widget is of the type pulldown menu in a Menubar, then the menu entries are packed into the Menubar frame that was created in the `oplevel` procedure (Figure 5).

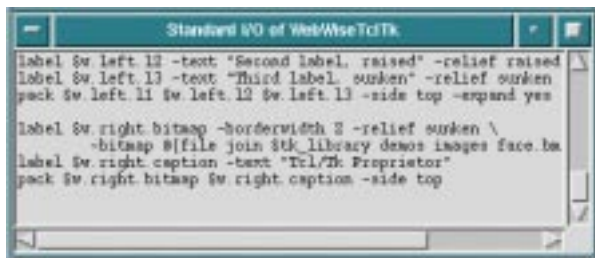


Fig. 8. Standard I/O of WebWiseTclTk toolkit.

Standard I/O and audio. We have created a special window for standard I/O in WebWiseTclTk. Any communication to the standard I/O channel by commands such as `puts` and `gets` is redirected to the special window, as shown in Figure 8. Therefore, it is possible for the `tcllet` and a user to interact through the commands `puts` and `gets`.

Audio commands, such as `bell`, are still potentially dangerous, with the risk of producing a continuous tone. Therefore, we defined a procedure `bell` which produces a visual effect by momentarily changing the background color of the canvas widget.

Safe commands. Earlier, we noticed that whenever a toplevel window, say ".w", is created, the window name is mapped to a new window name ".c.1.w", corresponding to the main frame in the set of toplevel frames. Therefore, existing safe commands such as `button` with window names ".w.b"

will fail, unless their window names are also translated to a new name ".c.1.w.b", which is in the hierarchy of the toplevel main frame's children.

We rename the existing safe commands by moving them into a namespace for WebWiseTk, and define new procedures for them. Figure 9 shows a sample code for re-defining the command `button`. The newly defined procedure does the following:

1. maps all the window names, in the arguments passed to the procedure, to the corresponding hierarchy in the toplevel frame.
2. evaluates the original command `button` with mapped arguments. This creates a new command "\$new_w" for the translated window pathname.
3. defines a new procedure for the original window pathname "\$w" that would have been created otherwise. This procedure, in turn, invokes "\$new_w" whenever it is called.
4. translates the window names in the returned values back to original window names. This is important because the returned values may be directly passed to other code for evaluation. Example: "`pack [button .b]`".
5. returns the translated value "\$new_ret".

```
# Move original command to WebWiseTk namespace
rename button ::WebWiseTk::button-Orig
# Define a new command in the global namespace
proc ::button {w args} {
# Map window names
set new_w [mapArgWindowNames $w]
set new_args [mapArgWindowNames $args]
set ret [uplevel 1 ::WebWiseTk::button-Orig \
        $new_w $new_args]
# Create a proc called $w
proc ::$w {args} {
# Script to invoke command $new_w ...
}
# Map back returned window names
set new_ret [mapRetWindowNames $ret]
return $new_ret
}
```

Fig. 9. New definition for command `button`.

The command `bind` also has to be re-defined. This is because the value of "%W" in the `bind` script gets the real window name (".c.1.b") instead of the window name (".b") supplied by the `tcllet`. Thus all window names referred by "%W" in the `bind` script are mapped back appropriately, before invoking the original `bind` script.

Similarly, the command `wininfo` is also redefined, so that its queries, such as "`wininfo width`", "`wininfo children`", etc., are correctly handled.

Unsafe commands. Few commands, such as `send`, `tk_getOpenFile`, `tk_getSaveFile`, etc., do not pose

the *denial of service* attacks, but are still unsafe and very dangerous to the client host system since they present other forms of security attacks. These commands are therefore not available in `WebWiseTk`. However, it is always possible to use an appropriate security policy, other than the *home policy*, to enable these commands.

Unsafe options. A few options for safe commands are considered unsafe and hence not available in `Safe-Tk`. These include `"-bitmap @filename"`, `"-file filename"` and `"-maskfile filename"`, among others. It is possible to allow these options on the following conditions:

1. the host system supports the use of the *home policy*, and
2. the specified file exists on the server site of the *tcl* code.

In such a case, the data for the specified filename is downloaded from the server site using the command `::browser::getURL filename`. Then `"-file filename"` or `"-maskfile filename"` option is replaced by `"-data $downloadedData"` or `"-maskdata $downloadedData"`. On the other hand, for the option `"-bitmap @filename"`, a bitmap image is first created using the command `image`. Here the replaced option is `"-image [image create bitmap -data $downloadedData]"`. The command `image create image -file filename` is also replaced with `"-data $downloadedData"` option, after we download the data for the specified filename from the server site. However, the option `"-data $downloadedData"` expects the image data to be in *base64* format. Images in other formats are therefore translated to *base64* format using the *tcl-only* encoding procedures available in the Data Handling Package [15]. The encoding process is slow and hence for images of considerable size, one should save the original images in *base64* format, instead of encoding them on the fly during execution of the *tcl*.

5 WebWiseTcl

To maintain security, it is important that the unsafe Tcl commands be hidden or restricted in `Safe-Tcl`. Several different security policies offered by `Tcl-plugin 2.0` are convenient and allow the application programmer to design *tcl* codes accordingly. We intend to make use of the *home policy* to enhance the functionality of the `Safe-Tcl` for `WebWiseTcl`.

Script libraries and packages provide an excellent mechanism to structure an application code into several smaller scripts, and then dynamically load each script as needed. We modify the restricted com-

mands in `Safe-Tcl` such that it supports the packaging facility to automatically load scripts from the server site of the *tcl* code. We only need to append the location of the server site, given by `"getattr originHomeDirURL"`, to the `auto_path` variable for the `auto_load` procedure to work correctly with the modified commands described next.

Source. The filename argument for the `source` command is parsed for a URL. If the filename is a URL, then it is downloaded using the command `::browser::getURL filename` and its contents are evaluated. Otherwise, the original `source` command is invoked, as shown in Figure 10.

```
# Move original command to WebWiseTcl namespace
rename source ::WebWiseTcl::source-Orig
# Define a new command in the global namespace
proc ::source filename {
    if {[string match http:* $filename]} {
        # Evaluate script downloaded from a URL
        uplevel 1 [::browser::getURL $filename]
    } else {
        # Invoke original source command
        uplevel 1 ::WebWiseTcl::source-Orig $filename
    }
}
```

Fig. 10. New definition for command `source`.

Open and close. When a filename specified for `open` is a URL, the specified URL is downloaded and saved on the temporary disk space of the host system assigned by the *home policy*. Then, this file on the local disk is opened and its channel identifier returned. Correspondingly, when a `close` command is invoked for a URL, the file on the local disk is not only closed, but also deleted. These functions are useful for opening a file/URL in read-only mode.

File. We have re-defined the command `file` so that its options `dirname`, `join`, and `split` return correct results even when the specified filename is a URL.

Pwd, cd and glob. These commands are not available in `Safe-Tcl`. We therefore assign the URL of the server site, given by `"getattr originHomeDirURL"`, to be the default working directory returned by the command `pwd`. This value is stored in a variable defined in `WebWiseTcl` namespace. The invocation of the command `cd` then results in change of value of the current working directory stored in the variable. The command `glob` returns a list of all matching URLs found under the URL given by the current working directory.

6 Users Guide

We define *users* as those who intend to download and view `WebWiseTclTk` toolkit-based Tcl-plugin appli-

cations within their Web browsers.

Users can very easily and quickly familiarize themselves with the WebWiseTclTk environment. Figure 2 shows one such typical view of the environment within a Netscape browser. The layout of the environment is shown in Figure 4. It has two widget areas - the one on the left contains windows created by the *tcl*et, and the one on the right displays a list of buttons corresponding to each iconified window. Both the widgets have auto scrollbars. At the bottom, a single line help message is displayed, based on the location of the mouse cursor. The size of the widget containing the *tcl*et windows may be increased or reduced by the user under the Preferences option. A user may also resize the canvas and the text widget areas by merely dragging the border between the two with a mouse cursor.

Installation. It is not necessary for the users to install the WebWiseTclTk toolkit. The scripts in the toolkit are dynamically downloaded, as and when required, from the server site of the running application/*tcl*et. However, for faster access, users do have an option of installing the WebWiseTclTk toolkit in their Tcl-plugin directory. In this case, the installation procedure consists of the following:

1. Download the latest version of the WebWiseTclTk toolkit from:

```
http://www.cbl.ncsu.edu/software/#WebWiseTclTk.
```

2. Change to the installation directory of the Tcl-plugin on your local file system.

```
# For local installation,  
csh% cd ~/.netscape/tclplug/2.0
```

```
# Or, for site installation,  
csh% cd /usr/local/lib/netscape/tclplug/2.0
```

3. Gunzip and untar the toolkit.

```
csh% gzip -dc WebWiseTclTk-x.y.tar.gz | tar xf -
```

4. Verify the installation by visiting the test site under <http://www.cbl.ncsu.edu/software/#WebWiseTclTk>.

The toolkit consists of *tcl-only* scripts and hence does not require any compilation step.

7 Programmers Guide

We define *programmers* as those who: (1) intend to write Tcl-plugin applications based on the WebWiseTclTk toolkit, or (2) wish to translate their existing Tcl applications into *tcl*ets for execution over the Web.

Programmers, who intend to use the WebWiseTclTk toolkit for their *tcl*ets, should follow the guidelines listed below:

1. Download the latest version of the WebWiseTclTk toolkit from:

```
http://www.cbl.ncsu.edu/software/#WebWiseTclTk.
```

2. Change to a directory on your system that is

accessible on your Web server.

```
csh% cd /home/user/public_html/tclets
```

For example, let the URL corresponding to this directory be <http://www.your.web.site/~user/tclets>.

3. Gunzip and untar the toolkit

```
csh% gzip -dc WebWiseTclTk-x.y.tar.gz | tar xf -
```

4. Verify the installation by clicking visiting the examples distributed with the toolkit under

```
http://www.your.web.site/~user/tclets/
```

```
WebWiseTclTk-x.y/examples
```

Figure 11 shows an example to encapsulate the Tk widget demos and execute them on the Web. The TkWidgetDemos.tcl script, the demos directory and the WebWiseTclTk toolkit directory all exist in the same directory location on the Web as shown below:

```
/home/user/public_html/tclets/WebWiseTclTk-x.y/examples  
├── TkWidgetDemos.html  
├── TkWidgetDemos.tcl  
├── WebWiseTclTk  
│   ├── DownloadToolkit.tcl  
│   └── toplevel.tcl  
│   └── ...  
├── demos  
│   ├── widget  
│   ├── arrow.tcl  
│   └── button.tcl  
│   └── ...
```

When a user downloads the TkWidgetDemos.tcl script, the script first tries to load the WebWiseTclTk toolkit from the user's host system. If it succeeds, then the *home policy* is requested since the Tk widget demos consist of several different scripts. On the other hand, if the WebWiseTclTk toolkit cannot be loaded from the user's host system, then it is downloaded from the *tcl*ets's server site.

The variable `$tk_library` is set to point to the *tcl*ets's server site so that it knows from where to `auto_load` the demo script. Finally, the widget script is sourced to execute the demos.

If the *tcl*et does not require the use of display, possible by setting `"tk=0"` in the html embed statement, then it is also possible to load only the WebWiseTcl toolkit.

Debugging. When writing new *tcl*ets, programmers can avoid using *Tcl-commands* which are either *not available* or *not yet implemented* in WebWiseTclTk toolkit. However, when converting existing applications, it is very difficult to isolate and remove these commands in the code. Therefore, we provide a mechanism whereby a dialog box is popped up whenever any *unavailable* or *unimplemented* command is used in the code the first time, as shown in Figure 12.

The programmer, who is testing the *tcl*et as a user,

```

# TkWidgetDemos.tcl --
#

set WebWisePKG WebWiseTk ; # Necessary, if the tclet requires display
#set WebWisePKG WebWiseTcl ; # Use this if display is NOT required
#
# Get the URL for tclet's server site
if {[catch {set originHomeDirURL [string trimright [getattr originHomeDirURL] /]}]} {
# Software is invoked from the local file system (NOT in a Web browser)
set originHomeDirURL /home/your_path/tclets
set auto_path [linsert $auto_path 0 $originHomeDirURL/WebWiseTclTk]
package require $WebWisePKG
} else {
# Software is invoked from a Web browser
# Check whether the toolkit is available locally on the user's host system.
if {[catch {package require $WebWisePKG}]} {
# Not available - so download it from the tclet's server site.
policy home ; # Need this policy to fetch URL from the server site.
eval [::browser::getURL $originHomeDirURL/WebWiseTclTk/DownloadToolkit.tcl]
# Now setup the package auto_load
package require $WebWisePKG
} else {
# Use the home policy, if the tclet code consists of several scripts
policy home
}
}
# Initiallize the WebWiseTk layout
WebWiseTk .webdesk

# Set tk_library, so that it can access other files in the demos directory.
set tk_library $originHomeDirURL
# Now invoke the Tk widget demos
source $originHomeDirURL/demos/widget

# Alternatively, if the tclet code is small, it can follow here.
# your tcl script...

```

Fig. 11. Main script for encapsulation of the Tk widget demos shown in Figure 2.

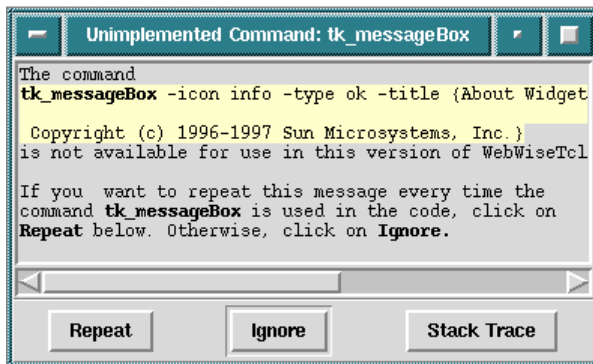


Fig. 12. Dialog box for unimplemented commands.

has a choice to either ignore the generation of this dialog box, the next time the same command is used, or to repeat it. A "Stack Trace" button is also available to locate the generation of the unimplemented command in the code.

Extensibility and reconfigurability. The WebWiseTclTk toolkit consists of several smaller scripts, specifically one file for each command that is either newly defined or re-defined. Therefore, pro-

grammers can (1) define and add their own commands which may be unavailable, such as `send`, in a separate file, or (2) re-define the existing commands by modifying the corresponding file to implement their own version of the command.

For example, the commands `toplevel` and `menu` may be re-implemented with a different layout to give a native look and feel on different platforms.

8 Experiences and Future Scope

The *Tcl-only* implementation, mapping window names onto a canvas object, real-time conversion of gif images to base64 format, etc results in degradation of performance. We have tried to minimize these effects by modularizing the code in such a fashion that related tcl procedures are clustered into single script files. Thus, procedures required for implementation of the `menu` command are loaded only if the command is used by the *tclet* code. Another area of optimization is possible by improving the efficiency of the procedure to map window names back and forth onto canvas objects, since this is one of the

most frequently called procedure.

Installing a local copy of the toolkit with the Tcl-plugin on the client host will improve the performance when the distance between the client host and the web-server is large. We also need to improve the reliability of the toolkit by adding sufficient hooks to handle cases when 'getURL' is likely to timeout or fail under high network traffic conditions.

9 Software Availability and Status

The WebWiseTclTk toolkit described is available at <http://www.cbl.ncsu.edu/software/#WebWiseTclTk>. The current version of the toolkit is *beta 1.02*.

We have successfully tested this version of the toolkit on a Sun Sparc workstation with *Solaris 2.5.1* using Netscape 4.0 and 3.0.

On a *Windows 95/NT* machine, we had to install the toolkit locally before we could access *tclets* based on WebWiseTclTk. Also, we had to use a special policy that allows downloading scripts from the server site using the command `::http::geturl` from the http package. This is because, (1) the blocking version of the command `::browser::getURL` is not supported in Netscape 4.0, and (2) the command `::browser::getURL` is not available under Internet Explorer 4.0 for the Tcl-plugin. The implementation of menu widgets in WebWiseTclTk is specific to Unix and hence do not function properly on a *Windows 95/NT* machine.

We have also tested the toolkit successfully on a *Mac* running under *MacOS 8.0*. Again, the current implementation of menu widgets do not work correctly on a *Mac*.

Some of the features of the WebWiseTclTk toolkit described in this paper are not yet implemented. For example, in menu widgets, advanced features that are not implemented include: the *accelerator* option for any of its entry is ignored, creation of clones of menu frames using the tear-off entry is not possible, etc. For details of such items and current updates, please consult

<http://www.cbl.ncsu.edu/software/#WebWiseTclTk>.

10 Conclusions

We have demonstrated the capabilities of the WebWiseTk toolkit by readily posting an existing *Tcl application*, namely the Tk Widget Demos, as an executable *Tclet* on the Web. Since these demos cover most of the commands available in the main Tcl/Tk interpreter, they signify the potential usefulness of the toolkit.

Introduction of the WebWiseTcl toolkit, which uses the *home policy*, enables *programmers* to structure

their *tclets* into several smaller scripts. Such scripts are easier to manage and dynamically loaded during the execution of the *tclet*.

While most of the commands related to *denial of service* attacks may be eventually restored in the Tcl-Plugin, WebWiseTclTk toolkit still offers the ability to confine the *tclet* windows to a single display within the Web browser.

Our first major application of WebWiseTk has been the introduction of Web-based user-configurable and executable workflows that support an environment functionally similar to one in REUBEN [11, 12, 13]. First demos of this capability has been shown in the University Booth during the 1998 Design Automation Conference [16]. On-line demos are accessible from <http://www.cbl.ncsu.edu/demos>.

References

- [1] The Tcl/Tk Home Page. Published under URL <http://sunscript.sun.com/>, 1997.
- [2] J. K. Ousterhout. *Tcl and the Tk Toolkit*. Addison-Wesley, 1994.
- [3] B. B. Welch. *Practical Programming in Tcl and Tk*. Prentice Hall, 1997.
- [4] J. K. Ousterhout. Scripting: Higher Level Programming for the 21st Century. Published under URL <http://scriptics.com/people/john.ousterhout/-scripting.html>, March 1998.
- [5] Scriptics Corporation. Published under URL <http://www.scriptics.com/>, 1998.
- [6] The Tcl/Tk Consortium. Published under URL <http://www.tclconsortium.org/>, 1998.
- [7] The Tcl Plugin Home Page. Published under URL <http://sunscript.sun.com/plugin>, 1997.
- [8] J. Y. Levy. A Tcl/Tk Netscape Plugin. Published under URL <http://sunscript.sun.com/plugin/paper.html>, May 1996.
- [9] J. K. Ousterhout, J. Y. Levy, and B. B. Welch. The Safe-Tcl Security Model. Published under URL <http://scriptics.com/people/john.ousterhout/-safeTcl.ps>, March 1997. Draft.
- [10] D. Libes. *Exploring Expect*. O'Reilly and Associates, 1995.
- [11] H. Lavana, A. Khetawat, F. Brglez, and K. Kozminski. Executable Workflows: A Paradigm for Collaborative Design on the Internet. In *Proceedings of the 34th Design Automation Conference*, pages 553-558, June 1997. Also available at <http://www.cbl.ncsu.edu/publications/#1997-DAC-Lavana>.
- [12] Amit Khetawat. Collaborative Computing on the Internet. Master's thesis, Electrical and Computer Engineering, North Carolina State University, Raleigh, N.C., May 1997. Also available at <http://www.cbl.ncsu.edu/publications/#1997-Thesis-MS-Khetawat>.
- [13] H. Lavana, A. Khetawat, and F. Brglez. Internet-based Workflows: A Paradigm for Dynamically Reconfigurable Desktop Environments. In *ACM Proceedings of the International Conference on Supporting Group Work*, Nov 1997. Also available at <http://www.cbl.ncsu.edu/publications/#1997-GROUP-Lavana>.
- [14] I. K. Lam and B. Smith. Jacl: A Tcl Implementation in Java. In *Proceedings of the Fifth Annual Tcl/Tk Workshop*, July 1997.
- [15] Document Handling Package. Published under URL <http://tcltk.anu.edu.au/DHP/>, 1997.
- [16] Design Automation Conference. Published under URL <http://www.dac.com/>, 1998.