WubTk - Tcl/Tk Apps Anywhere

Steve Landers and Mike Doyle
Buonacorsi Foundation

steve@digitalsmarties.com
mike@iomas.com

Abstract

WubTk is a new implementation of Tk that is layered on jQuery in a browser. It allows development of web applications in Tcl using a familiar Tk-like API.

This paper describes the conversion of an existing biomedical visualization program to WubTk, enabling it to be deployed in a browser. It also describes a custom iOS App that provides a more native look and feel on devices like the iPad and iPhone.

The paper also positions WubTk with respect to Tk (including enhancements beyond the Tk API) and discusses planned new features.

Background

WubTk is a new Tk implementation that maps a subset of the Tk API onto the jQueryUI [1] widgets running in a browser. In terms of the Model-View-Controller (MVC) design pattern [2], the Model and Controller execute in the web server (in Tcl code) and the View executes in the browser (in Javascript code).

The goal of the WubTk project is to provide a Tk-like API that uses a browser as a "display device". So, rather than mapping Tk commands to X11 or Win32 or Cocoa (as does the default Tk implementations), WubTk maps onto the jQueryUI widgets (although it could use any Javascript-based widgets). Instead of using "wish myapp.tcl" the developer runs "wubtk myapp.tk" and connects to the local host from a browser.

The work is being driven by Steve Redler (steve@sr-tech.com) and his customer needs, with development work largely being done by Colin McCormack (colin@chinix.com). Steve was initially inspired by Roy Keene's tkweb project [3] but WubTk has gone much further.

The API is “Tk-like” in the sense that there is no intention of implementing all of Tk, just the

parts appropriate for web applications and which map conveniently onto jQueryUI features.

Development priorities are driven by the needs and interests of the developers and early adopters. While it would be nice if WubTk scripts could also be guaranteed to run on Tk, this isn't seen as essential. The goal is more to ensure portability of Tk skills rather than portability of scripts.

The History

The desire to build interactive web applications in Tcl/Tk isn't new.

In 1995 Eolas released a version of WebRouser, an applet-enabled web browser based on Eolas' enhanced version of NCSA Mosaic that could run Tcl/Tk scripts using Eolas' WebWish Tcl plugin. WebRouser and WebWish were presented in the cover story in the February 1996 issue of Dr Dobb's Journal [3].

In 1996 Jeff Hobbs produced a “proof of concept” Tcl plugin for Netscape following a visit to the Tcl group that was then at SunLabs. Jacob Levy (part of that group) produced the first Tcl/Tk plugin for Netscape and Laurent Demailly worked on the 2.0 implementation [5]. Version 3.0 [6] can still be installed in Firefox and Internet Explorer.

In 1998-1999 Mark Roseman and his team at TeamWave software implemented ProxyTk, a Java applet user interface toolkit for Tcl [7]. The small Java applet (50k bytes in size) runs in a user’s browser to provide the user interface, and communicates with a Tk-like API running in a Tcl web server. Unlike the two previous browser plugin examples, with ProxyTk, the application was split between the user interface running in the browser and the application Tcl code running on the server, with an efficient protocol connecting the two. ProxyTk was ahead of its time, and showed the way forward for Tcl/Tk web apps (i.e. a client / server solution), the application was split between the user interface running in the browser and the application Tcl code running on the server, with an efficient protocol connecting the two. ProxyTk was ahead of its time, and showed the way forward for Tcl/Tk web apps (i.e. a client / server solution) but unfortunately it was swallowed in a corporate takeover and never reached its full potential.

All of the above solutions have the disadvantage that they require a plugin. In many organisations, it isn't possible for the user to install plugins in the standard operating environment (SOE) and so a solution that leverages components that don't require installation is desirable.

In 2003 Roy Keene wrote TkWeb [8], an attempt at rendering Tcl/Tk scripts using HTML. And around the same time Wilfred J. Hansen published a technical note about Rendering Tcl/Tk Windows as HTML [9]. Both of these were experiments aimed at proving the practicality of retaining the Tk API while rendering Tk widgets in a browser by generating HTML.

In 2006 Tom Poindexter developed Æjaks [10] – which “combines the server-side Ajax-based

windowing system, Echo2, with the powerful simplicity of the Tcl language" . Æjaks is a thin layer over Echo2 [11] , a Java and Javascript-based platform for building interactive web-based applications. It translates Echo2 objects into Tcl objects, accessible behind a Tk-like object interface. Æjaks is a very capable system, and definitely one to consider for Tk based web development. But it does have a downside: the Tcl interpreter used is Jacl (an alternative implementation of Tcl 8.0 written in Java). Thus many recent developments in Tcl aren't available, although Jacl is currently being overhauled and this situation should improve significantly in time.

The other consideration with Æjaks is that, although Echo2 provides excellent cross-browser compatibility and uses modern Javascript techniques, it doesn't have the same amount of community acceptance or contributed widgets as other Javascript Web frameworks – in particulary jQuery and jQueryUI.

jQuery [12] is a cross-browser JavaScript library designed to simplify the client-side scripting of HTML. jQueryUI is a library that "provides abstractions for low-level interaction and animation, advanced effects and high-level, themeable widgets, built on top of the jQuery.

Both jQuery and jQueryUI have a significant and growing community of developers and users, and present an obvious choice for developers of rich, interactive web applications. The question is how best to leverage them from Tcl? The answer is to combine the approach of TkWeb (i.e. generating html) with the power and simplicity of jQueryUI.

The result is WubTk - a Tk-like API that maps Tk commands to jQueryUI widgets.

The Design

WubTk is (as the name indicates) implemented on top of Wub, a web server written in Tcl [13]. Although it isn't tied to Wub specifically, Wub was used because of its support for rapid prototyping, extreme configurability and advanced features (including HTTP 1.1 persistent connections and support for generating jQuery from within Tcl).

Each WubTk instance is run in a separate interpreter within the Wub server. The WubTk script is interpreted by the same Tcl interpreter as is running Wub, thus the application developer has access to all the features the latest Tcl provides.

Each WubTk instance has a persistent state, in that the per-instance interpreter is retained between requests from the browser. When the Wub "human" module recognises an individual user session it will crate a unique "human" cookie for the instance. Just like online stores with shopping carts (where a user accessing a store from the same browser twice will see the same shopping cart contents) so also with a WubTk intance. The current instance can be explicitly deleted from within the WubTk script by calling the "exit" command.

The WubTk API is intended to be as close to Tk as is practical, but with no guarantee as to preserving either the syntax or semantics of the existing Tk implementations. As stated previously, the goal of WubTk is to ensure portability of Tk skills and (in particular) to make those skills relevant in the Web 2.0 and mobile Internet world.

There are a number of differences with Tk, which can be categorized as follows:

- features that have already been implemented
- features that aren't yet implemented
- features that most likely won't be implemented because there is a conceptual mismatch with the client/server web world
- enhanced features that could be implemented as a shim over Tk
- convenience features relating specifically to the browser environment
- features that leverage jQueryUI functionality

Features already implemented include the button, select, combobox, checkbutton, radiobutton, label, scale, entry, text, wm, image, grid, frame, toplevel and notebook widgets.

Note that grid is the geometry manager for WubTk. In this alone it is of enormous value in that it frees the programmer from the constraints of the HTML/CSS Box Model.

Feature that aren't yet implemented include a large part of the canvas functionality. Some preliminary work has been done by Steve Redler on implementing some canvas features on top of the Tcl to SVG package [14]. This has been used, for example, to port Arjen Markus' Plotchart package [15] to WubTk, providing temperature charts from plant sensors in a factory automation product.

Features that most likely won't be implemented, or will be implemented in a different form, include the Tk event facility. It just doesn't make sense to have a low-level event mechanism that requires a round trip from the client to the server for each event (e.g. the interaction between a listbox and a scrollbar). A better approach is to abstract the event handling into two categories:

- one for user interface events to be handled between widgets in the browser
- another for application events that would be passed back from the browser to the application running on the server (more akin to the standard Tk event mechanism)

There are a number of enhanced features already implemented including the gridding option on each widget, used instead of the grid command:

```
entry .entry -textvariable ::entryvar -grid {2 1 -sticky nsew}
```

Another enhanced feature is the ability to grid an image directly, rather than needing to create a label to reference that image, and the -url flag to specify a URL for the image to load.

```
image create photo .image -url /images/logo.gif
```

Yet another is that the frame widget can take a label, and display a border around the frame. Examples of frames can be seen in the application described later in this paper. Each of these features could be implemented as a shim for standard Tk at some stage.

There are a number convenience features related to the browser environment. These are particularly useful when prototyping new widgets, or to add new behaviour to existing widgets.

For example, the following shows how to add Javascript functionality to a Tk button using the -js option:

```
button .size -text "Video size" -grid {3 2} -command VidSize -js {
      $('grid_3_2').onclick = function() {
          // do something useful in JS when button is clicked
      }
}

proc VidSize {} {
     # do something useful in Tcl when button is clicked
}
```

The -style option can be used to inject CSS to the definition of a widget. The CSS is specified as a Tcl dictionary, as follows:

```
button .stop -text "Stop it" -grid {1 1} -command StopIt -css {
      position absolute top 50% left 50% text-align center
}
```

The other convenience features are browser related widgets, including html, cookie, and upload. The html widget is used to inject HTML when the page is rendered, typically to specify a widget in HTML and CSS. For example, to define a widget to display a video using the HTML5 video element:

```
html .movie -grid {4 0 1 3} -text {
     <video src='/path/to/movie.m4v' controls autoplay></video>
}
```

The “wm header” command can be used to inject markup into the page header. The following adds a no-cache directive to the page:

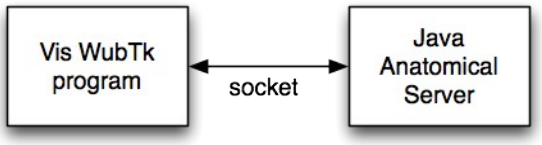
```
wm header . {
    <meta http-equiv="Pragma" content="no-cache">
}
```

And finally, there are new features that leverage jQueryUI functionality, in particular new widgets such as accordion and toolbar, but with many more to come.

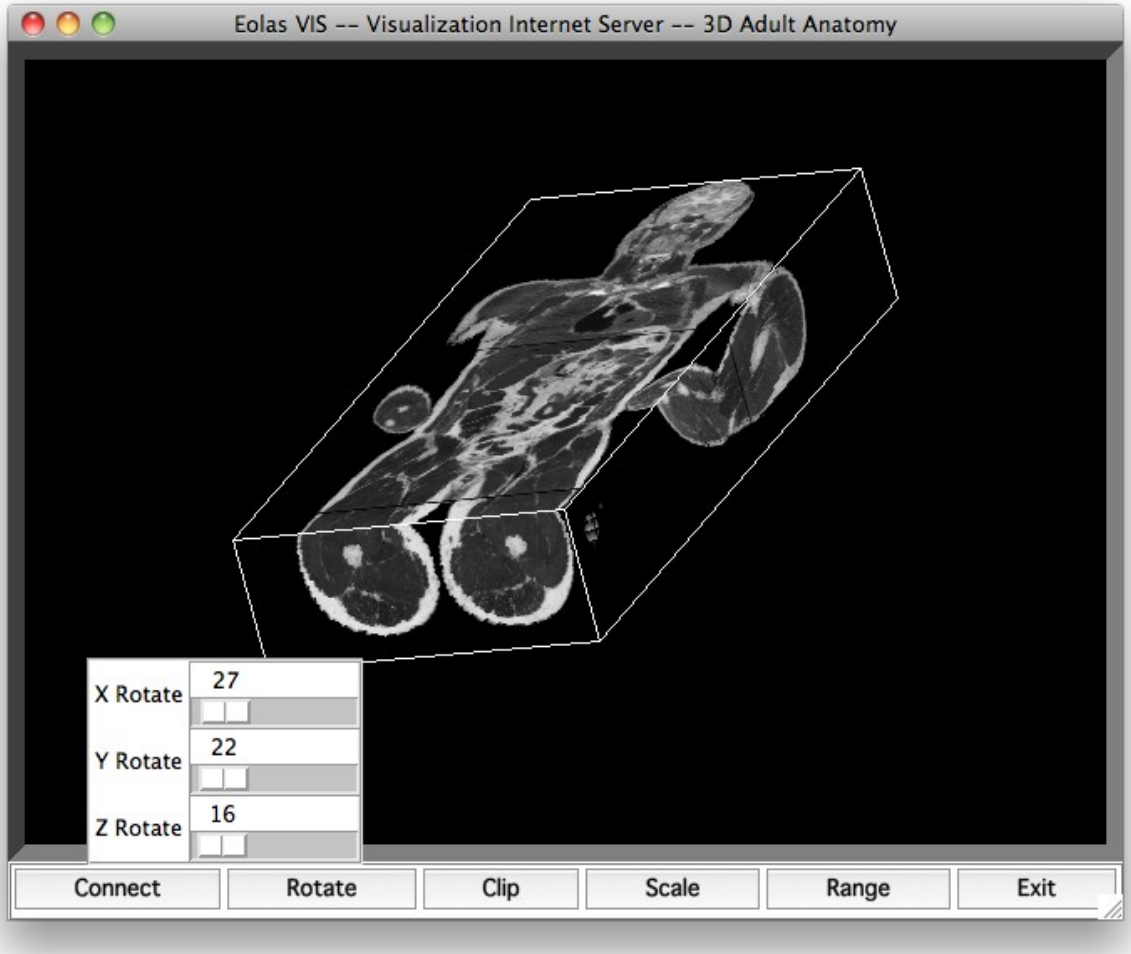
An Application - Biomedical Visualization

As an exercise in familiarising the authors with the WubTk features, an existing Tcl/Tk application was adapted to run under WubTk.

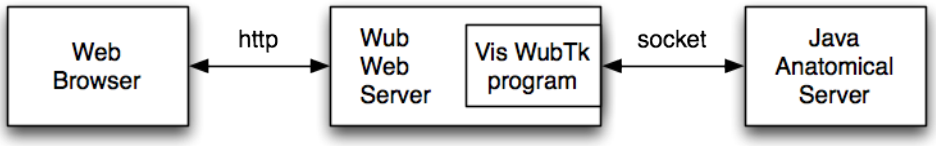
Vis is a biomedical visualization application developed by a team led by Mike Doyle in 1996. It comprises a Tcl/Tk front-end application communicating with a back-end Java Anatomical Server (MultiVis) using a socket connection and a custom textual protocol.



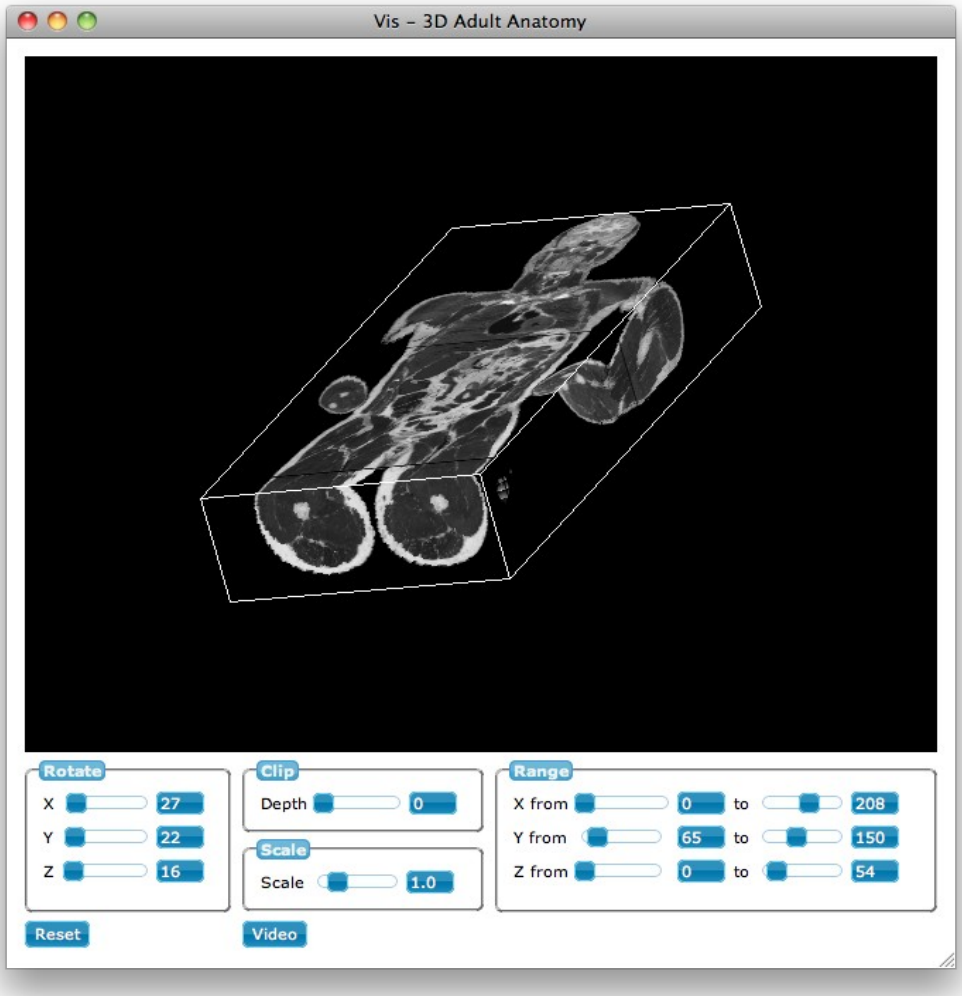
The Vis user interface comprises an image display, with a series of popup menus underneath it to control the view orientation parameters sent to the anatomical server. On receiving the parameters, the Anatomical Server constructs a GIF image and returns it (base64 encoded) to the Vis program, which displays in a label widget.



In the above screenshot, the Rotate menu (containing scale widgets for X, Y and Z rotation) is displayed after the user has clicked on the “Rotate” button. Likewise there are scale widgets for Clip, Scale, and the X, Y and Z ranges.
To gain experience in WubTk, a new Vis program was implemented that communicates with the Java Anatomical server using the existing protocol.



The layout of the new user interface was straightforward and resembles the old Vis Tcl/Tk program, except that popup menus aren't used, since they don't fit well with the browser user interface model.



The image is displayed centered in a label, since the image size is smaller than the width of the

page . Note the frames with text labels (i.e. Rotate, Clip, Scale and Range) similar to Ttk (a.k.a. Tile) labelframes.

The markup for this page is remarkably similar to a Tk program. For example, the following code was used.to define the Rotate frame

```
frame .rotate -text Rotate -grid {1 0 2 1 -sticky ns} \
              -style {height 9em}
label .rotate.xl -text "X" -grid {0 0} -style {padding-right 5px}
scale .rotate.xs -variable ::rotateX -from 0 -to 360 \
              -grid {0 1 -sticky nsew} -width 5em
entry .rotate.xe -textvariable ::rotateX -width 3 -grid {0 2}

label .rotate.yl -text "Y" -grid {1 0} -style {padding-right 5px}
scale .rotate.ys -variable ::rotateY -from 0 -to 360 \
              -grid {1 1 -sticky nsew} -width 5em
entry .rotate.ye -textvariable ::rotateY -width 3 -grid {1 2}

label .rotate.zl -text "Z" -grid {2 0} -style {padding-right 5px}
scale .rotate.zs -variable ::rotateZ -from 0 -to 360 \
              -grid {2 1 -sticky nsew} -width 5em
entry .rotate.ze -textvariable ::rotateZ -width 3 -grid {2 2}
```

Note the text (“Rotate”) in the frame command options, the -style option to set the height of the frame and the use of -grid. Another point to note is the specification of width in ems on the scale commands. But otherwise this markup should be very familiar to a Tk programmer.

The declaration of the label and associated image widgets is a little more complicated.

```
image create photo .image -url /images/logo.gif

label .display -background black -image .image -borderwidth 0 \
          -grid {0 0 1 3} \
          -style {
               display         table-cell
               vertical-align  middle
               text-align      center
               width           650px
               height          512px
          }
```

The image command uses the -url option to set an initial value. This is later updated when the first anatomical image is returned from the server.

The label command includes some (arguably) tricky CSS to both set the size, and ensure the image widget is centered within it. Surprisingly, centering an image in a label (or rather, a HTML div) is a Frequently Asked Question in the HTML/CSS world.

Rather than force the WubTk programmer to remember this, it would make sense to add a flag to the label widget to indicate vertically centered contents, as a partner to the existing -justify

flag. This is an example where WubTk might reasonably diverge from Tk, although it would be quite feasible to provide a WubTk package that could hide or support the additional features when using Tk.
Once the user interface was specified, the next step was to start adding actions in response events in the user interface (such as moving a scale widget or typing in an entry widget). Like with standard Tk, this is done by adding traces on global variables that are set by the scale. So, in the following example, the .rotate.xs scale sets the global variable rotateX, as does the .rotate.xe entry widget.

```
label .rotate.xl -text "X" -grid {0 0} -style {padding-right 5px}
scale .rotate.xs -variable ::rotateX -from 0 -to 360 \
                -grid {0 1 -sticky nsew} -width 5em
entry .rotate.xe -textvariable ::rotateX -width 3 -grid {0 2}
```

Then a trace is added to the rotateX variable to trigger the execution of a Tcl procedure when the variable's value changes:

```
trace add variable ::rotateX write do_rotateX
```

The do_rotateX procedure sends the value of rotateX to the Anatomical Server using the exiting protocol and awaits a response. It is important to remember that this code executes in a real Tcl per-instance interpreter running in the web server.

Here we encounter the first of the conceptual design issues that differs between a web application and a traditional Tcl/Tk application. Because the web application may be serving multiple users, it isn't reasonable for the Vis application to wait for the response from the back-end server. So a more appropriate approach is to suspend the Http request from the browser and resume when the data arrives from the back-end server. In Wub this is quite straightforward:

```
[Httpd Suspend $r]
```

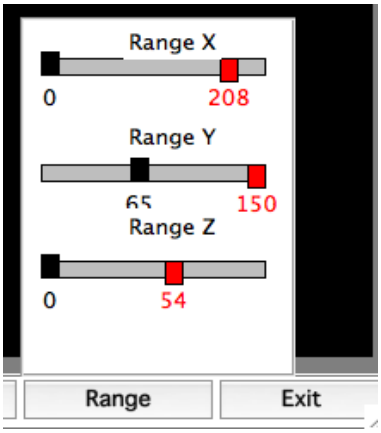
and

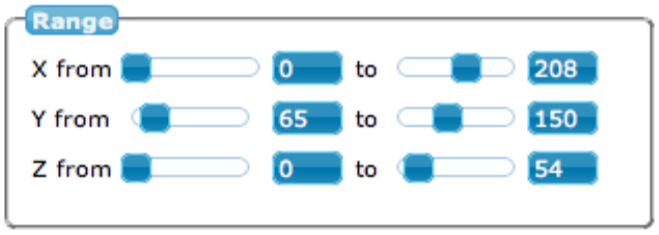
```
[Httpd Resume $r]
```

where $r is the dictionary containing the request from the browser (a standard Wub data structure).

The fact that changing a widget in the browser can trigger Tcl functionality in the server means that complex data integrity rules can be implemented in Tcl.

For example, in the original Vis application the Range menu allowed the user to specify the minimum and maximum values for each range, and to constrain these so that the minimum is less than or equal to the maximum (and vice versa). This was achieved by implementing a megawidget using customized scale widgets on a canvas, as shown in the left screenshot following:





In the WubTk Vis application (shown on the right), this is achieved using the standard scale widgets in a frame and by adding checks to the trace command on the associated variables:

```
trace add variable ::rangeXmin write do_rangeX
trace add variable ::rangeXmax write do_rangeX

proc do_rangeX args {
    global rangeXmin rangeXmax
    if {$rangeXmin > $rangeXmax} { set rangeXmin $rangeXmax }
    if {$rangeXmax < $rangeXmin} { set rangeXmax $rangeXmin }
    sendToBackendServer
}
```

The experience of reimplementing Vis using WubTk has proven that the approach combining Tcl and a Tk-like API with a Javascript-based user interface running in a browser is a practical and productive way of building web applications.

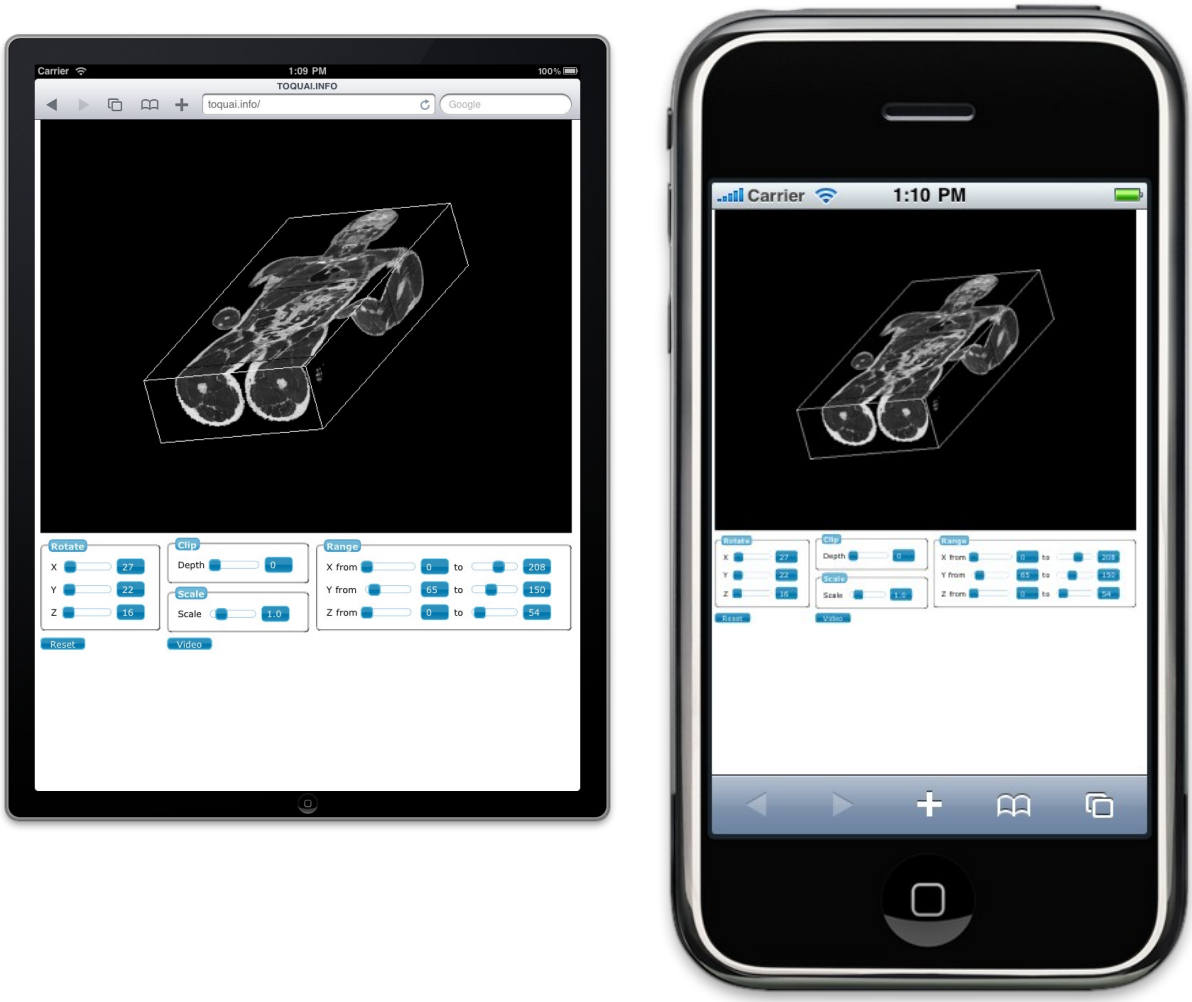
Getting Mobile

Mobile computing is the fastest growing area of IT. The market penetration of Internet-connected devices like SmartPhones and Tablets is significant and growing. Having a web-based Tk implementation makes it feasible to use these and other portable devices as “displays” connected to a server running Tcl application code. For networked applications this is, in many ways, a better solution than installing and maintaining an application on each device.

For some platforms, in particular iOS (iPhone/iPad) this is currently the only way to develop applications in Tcl/Tk for these devices. And increasingly, these devices are optimized to support Javascript, CSS and HTML (in particular, through WebKit-based browsers such as Safari and Chrome). So the performance of web applications is often indistinguishable from native applications.

The Vis application has been tested from the iPhone and iPad, and works just fine. Thus it is perhaps the first Tcl/Tk application for iOS.

The following shows Vis running on the iPad and iPhone simulators:



WubTk apps on such devices can be improved by leveraging mobile and/or Apple-specific CSS markup.

For example, the viewport tag can be used to properly fit the page content to the device screen:

```
wm header . {
     <meta name="viewport" content="width=device-width,
                    initial-scale=1.0, maximum-scale=2.0,
                    user-scalable=yes"/>
}
```

There are several Apple-specific tags that can be used to make a web application's behaviour more like a native application:

- apple-touch-icon - specifies the icon used when the user adds the app to the home screen
- apple-touch-startup-image – the application splash screen

- apple-mobile-web-app-capable – enables full screen mode so the browser location bar and controls aren't visible
- apple-mobile-web-app-status-bar-style to set the appearance of the status bar in full screen mode

These are only an intermediate solution to getting full native application look and feel. To achieve that requires two further steps – a generic embedded browser app for WubTk and a mobile device theme.

An app has been produced for iOS and ones for Android and even traditional desktops like Linux, OSX and Windows will follow: one for each platform that WubTk applications are to be deployed on.

The iOS WubTk app (called Toquai, pronounced tokay) is a native iOS Objective-C application that wraps a full-screen UIWebView (i.e. a Webkit browser without the location bar, controls and status bar).

The screenshot to the right shows Vis running via Toquai on an iPhone.

The user can add an application by dragging a URL over the Toquai icon on the iPhone/iPad home screen. Toquai then adds a new application-specific icon to the home screen. When the application icon is clicked it invokes an instance of Toquai connected to the application URL.

Toquai registers as a custom URL handler with iOS, so that a URL prefixed by “tok:” causes Toquai to be invoked and passed that URL. When Toquai is creating a home page icon for a WubTk application, it is just creating a link, such as “tok://vis.bionetlab.org” - the image and splash screen for the app come from the WubTk application itself.

There are several advantages to this approach:

Firstly, Toquai is generic, doesn't include application code and so doesn't need to be updated when an application is updated. Once Toquai is installed on a device it can present a choice of available applications simply by visiting a web site. Clicking on a link to an app will cause the app to be installed, as described above.

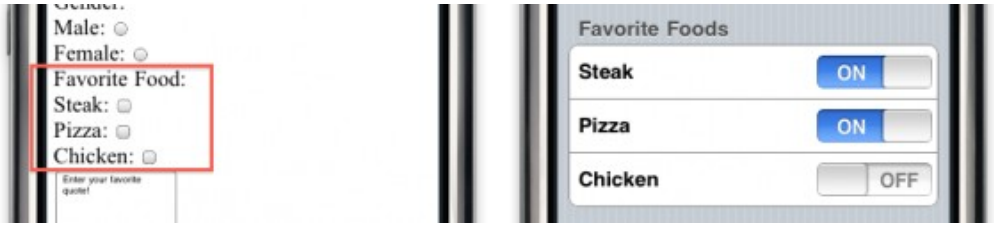
Secondly, platform-specific content can be served out of Toquai's local storage rather than

retrieved each time an application starts. In effect this will be a separate cache from the browser cache, so that libraries like jQuery and jQueryUI, and platform specific CSS will be less likely to be retrieved from the web (since it won't be flushed from the browser cache). This will potentially decrease the start-up time on slower devices or connections. Of course, Toquai can still update these locally cached copies if the versions in its cache don't match those on the web.

Thirdly, and perhaps most importantly, Toquai allows calls from the Javascript of an application into functionality provided in Objective-C (and vice-versa). This functionality will include encryption and authentication code, which is inherently insecure if implemented in the Javascript layer. By moving these functions to external code, they are kept safely out of reach from malicious code executing in the browser's address space.

The last piece of the puzzle is mobile-device specific theming.

The initial approach investigated was to use a platform specific package such as iWebkit [16] to create a more native look and feel. iWebKit substitutes more appropriate CSS when a web page renders on a mobile device. For example, checkbuttons are transformed to iPhone switches in the following before and after example:



But iWebKit also requires different markup (i.e. it isn't totally CSS based) and so a fully themed approach (i.e. CSS only) was considered a more appropriate cross-platform solution.

As mentioned earlier, jQueryUI is themed, and includes a facility called “ThemeRoller” for creating new themes [17]. One approach considered was to use to produce specific themes for each mobile platform supported.

However before this could be implemented the jQuery Mobile was announced [18]. jQuery Mobile is a “Touch-Optimized Web Framework for Smartphones & Tablets”. It provides a single user interface system across all popular mobile platforms, including iOS, Android, BlackBerry, Windows Phone, Symbian and Palm WebOS. It includes t;ouch-optimized layouts and UI widgets; and ThemeRoller-based designs.

Since jQueryMobile availability was announce for late 2010, and jQueryMobile appears to provide exactly the features needed by WubTk on mobile devices, the decision was made to wait and integrate jQueryMobile as soon as it is available.

This further vindicates the decision to base WubTk on a popular platform like jQueryUI.

Getting Funky

HTML5 [19] and CSS3 [20] are becoming mainstream, and are going to significantly change the way web applications are developed. WubTk is well positioned to take advantage of this, allowing a Tk-like API to be easily combined with advanced HTML5 / CSS3 markup.

It was previously mentioned that the WubTk html command can be used to inject HTML when a page is rendered, typically to specify a widget in HTML and CSS, and an example of including an HTML5 video element was shown. This could be expanded to use CSS3 transforms to animate the display of the video element

```
html .movie -grid {4 0 1 3} -text {
     <video src='/path/to/movie.m4v' controls autoplay></video>
     <style type='text/css'>
          video#movie {
               -webkit-transition: all 0.5s;
               -moz-transition: all 0.5s;
               -o-transition: all 0.5s;
          }
     </style>
}
```

Transformations can also be specified dynamically. For example, the “Video Size” button in the Vis application causes the video to be display in reduced size, over the top of the anatomical image. The video display can be skewed using the following jQuery command:

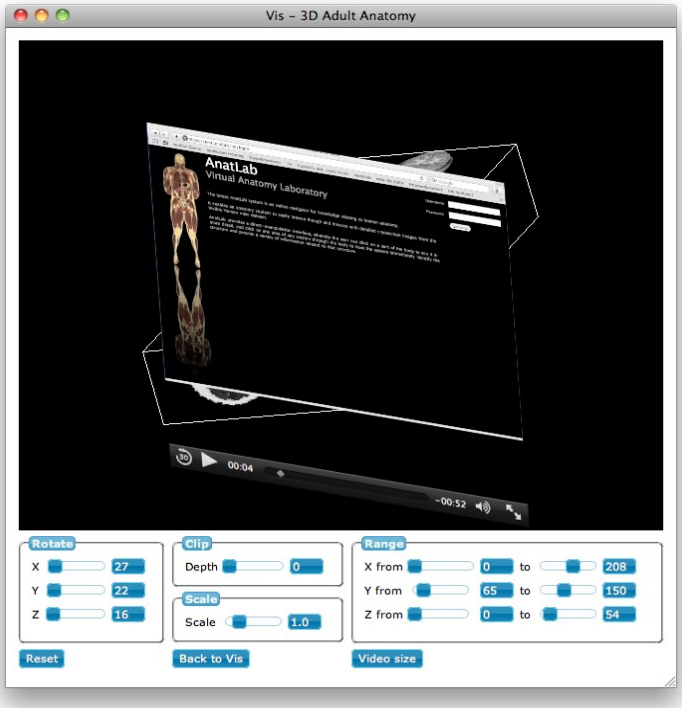
```
$('movie').style.webkitTransform = "skew(4deg, 10deg)";
```

Other transforms are available for alternative browsers such as Firefox.

The screenshot on the right shows the video in a skewed sub-window overlaying the image.

This only begins to scratch the surface of what can be done by combining Tk, jQuery, HTML5 and CSS3, but hopefully demonstrates the power of the facilities now accessible to the Tcl/Tk programmer through WubTk.

The HTML5/CSS3 train is coming, and developers can either get on board or get run over – but it can't be ignored.



Summary and Conclusion

WubTk validates the design direction of using a Tk-like API for building web apps. The combination of Tk, jQuery, HTML5 and CSS has allowed modern web features to be utilised by Tcl/Tk developers without losing the productivity that Tk is well known for.

WubTk allows Tcl/Tk programmers to apply their skills to the web and mobile domains, providing the first practical implementation of Tcl/Tk for deploying applications on iOS.

And further, it ensures the many advancements in web technology such as jQuery, HTML5 and CSS3 can be leveraged by Tcl/Tk developers.

While the current implementation is a proof of concept that will be optimised (or even replaced) over the next year, nevertheless it has already proven practical in the development of a number of web applications, including the example shown in this paper.

The technology described in this paper is covered by at least U.S. Patent 7,599,985, and other pending patents. Interested parties should contact Eolas Technologies Inc. for licensing of these patents.

References

[1] jQueryUI – http://jqueryui.com/
[2] Model-View-Controller – http://en.wikipedia.org/wiki/Model–View–Controller
[3] TkWeb - http://www.rkeene.org/projects/tkweb/
[4] WebRouser – Dr Dobb's Journal, Issue #244, February 1996
[5] Levy, J. A *Tk Netscape Plugin*. Proceedings of the Fourth Annual Tcl/Tk Workshop. July, 1996. http://www.usenix.org/publications/library/proceedings/tcl96/full_papers/levy/index.html
[6] Tcl/Tk Plugin Version 3 – http://www.tcl.tk/software/plugin/
[7] Roseman, Mark *Proxy Tk: A Java applet user interface toolkit for Tcl.* Proceedings of the Seventh Annual Tcl/Tk Conference, February 2000 http://www.usenix.org/events/tcl2k/full_papers/roseman/roseman_html/
[8] TkWeb – http://www.rkeene.org/projects/tkweb
[9] Hansen, Wilfred J. *Rendering Tcl/Tk Windows as HTML*. Proceedings of the Tenth Annual Tcl/Tk Conference, July 2003 http://www.tcl.tk/community/tcl2004/Tcl2003papers/rendering.doc
[10] Æjaks – http://aejaks.sourceforge.net/Aejaks_Home
[11] Echo2 – http://echo.nextapp.com/site/echo2
[12] jQuery – http://jquery.com
[13] Wub – http://wiki.tcl.tk/wub
[14] Tcl SVG – http://wiki.tcl.tk/TclSVG, http://wiki.tcl.tk/4534 and http://wiki.tcl.tk/4940
[15] PlotChart – http://wiki.tcl.tk/plotchart
[16] iWebKit – http://iwebkit.net
[17] jQueryUI ThemeRoller – http://jqueryui.com/themeroller/
[18] jQueryMobile – http://jquerymobile.com
[19] HTML5 – http://en.wikipedia.org/wiki/HTML5
[20] CSS3 – http://en.wikipedia.org/wiki/CSS3

All web references as at September 2010.