WTK for APWTCL

An implementation of TK like Widgets for APWTCL.

A paper for the Nineteenth Annual Tcl/Tk Conference

Abstract

During first half year of 2012 APWTCL (the successor of itcl in javascript) has been implemented and based on the javascript version two additional versions have been implemented for better native performance: APWTCL (Java) for Android based handhelds and APWTCL (Objective-C) for iPhone. To have running something comparable to Tk for APWTCL, there was the decision to use wtk from Mark Roseman (https://github.com/roseman/wtk) as a base. This package splits up the administration and data of a widget to be handled with Tcl (snit classes and objects) from the representation (displaying on the screen), which uses native support from the environment it is running on. Original wtk was dedicated to support javascript only, my implementation inserts an environment independent message interface (using a string based protocol) in between and then uses native support for displaying and handling the widgets and events. The action handling on the events is done by passing the information back to the Tcl part, which is modifying the (Tcl side) data and calling Tcl callback scripts if necessary.

Contact information

Arnulf Wiedemann

Lechstr. 10

D-86931 Prittriching

Email: arnulf@wiedemann-pri.de

1 The Idea

Already during implementing itcl in javascript there was the decision to use wtk as the frontend for GUI building. WTK (WebTk) is a Tk like implementation of some widgets (frame, entry, button, label, checkbutton and canvas) and a rudimentary implemenation of a grid manager. It is based on the idea to separate the data and administration part of the GUI from the presentation part. The communication between the two parts can be done in different ways:

- Using a direct function call to the wtk client side functions
- Using a client/server solution which is running the client part (the presentation part in javascript) in the browser and the server part (the Tcl part) on the serving machine

The base for the above implementation is a snit class widget, which can create and administrate a widget. It is (from the comment of the implementation of Mark Roseman):

A 'generic' widget object, which handles routines common to all widgets like assigning it an id, keeping track of whether or not it has been created, etc. Purely for convenience, we also include some code here that manages widgets that use -text or -textvariable, though not every widget will do so.

The “mega”widgets like frame, button, entry etc. are built with snit classes, which delegate a lot of functionality to the widget base class. Again a comment from Mark Roseman from the implementation:

Stuff for defining different widget types here. Note that all widgets are expected to implement the "_createjs" method. This is called by the generic widget code, and should return a Javascript command that can be used to create the widget on the web side of things (i.e. calls routines in wtk.js).

Widgets that support -text and -textvariable are expected to implement the "_textchangejs" method, which is called by the text handling pieces of the generic widget code, and should return a Javascript command that will change the text of the widget on the web side to match the current internal state of the widget here.

Widgets that receive events from the Javascript side are expected to implement the "_event" method, which is passed the widget-specific type of event and any parameters.

Wtk.js is a set of functions building the base of the repesentation/displaying (client side) part.

Communication between the administrative side and the displaying side is done using two global procs:

- toclient
- fromclient

These classes and procs have been used to implement a running version in itclinjavascript using a few javascript classes to allow the direct communication with the interpreter written in javascript.

2 How the current version started

As can be seen in my presentation APWTCL at the 10th European Tcl/Tk User Meeting ths year and on the wiki page there has been a reimplementation of itclinjavascript based on JimTcl. This version was the first version wtk from Mark Roseman as an interface for handling basic widget support. Following that APWTCL (Javascript) version there was another implementation of APWTCL in Java for supporting Android smartphones and an implementation in Objective-C for support of iPhones.

When arriving at the point for building support for wtk widgets the first approach was to use the administrative part of wtk mostly as it was and to replace sending of javascript code to the displaying part by a generic message (string based) interface.

On the client side there should be a small interpreter for decoding the messages and for switching and dispatching to the appropriate functions for doing the displaying and event handling work.

This was first implemented for the iOS version, later on there was a port of that code to Java to support Android.

3 The Message Interface

For the message interface a simple string based protocol was implemented, which had the the message itself and the parts encoded as length and info parts. A Message is generally built similar to a Tcl proc call in having a command and some parameters for the command. The command is normally a class object and the first parameter is the action to be executed on a GUI element. The parameter is normally a handle for the GUI element to be worked on and the other parameters are additional info for the action like option and value pairs. The first character of the message shows the type of the message normally M and there will eventually be a type E for end of a message block. That way the protocol is extensible with other types. Right now there exist no other types.

The layout of the message interface is as follows:

M<length of message>:<message>

<length of message> is the length of the following message text (not including the ":") as an integer number

and a message is composed of 1 .. n parts with the following layout:

<length of part>:<part>

<length of part> is the length of the following message text (not including the ":") as an integer number

<part> is a sequence of printable ASCII characters.

These messages are interpreted on the client side (or the part which acts as a client for example some class methods).

Some examples:

M53:9:wtkclient11:createLabel4:obj120:label: Hello Chicago

M33:9:wtkclient7:newGrid4:obj05:grid0

M38:9:wtkclient4:grid5:grid09:insertRow1:0

M48:9:wtkclient4:grid5:grid03:row1:010:insertCell1:0

M70:9:wtkclient4:grid5:grid03:row1:04:cell1:011:appendChild7:widgets4:obj1

M47:9:wtkclient12:createButton4:obj213:Hello Chicago

M70:9:wtkclient4:grid5:grid03:row1:04:cell1:011:appendChild7:widgets4:obj2

M38:9:wtkclient12:createButton4:obj34:Quit

M70:9:wtkclient4:grid5:grid03:row1:04:cell1:011:appendChild7:widgets4:obj3

4 The Client Side

The client side is responsible for creating and displaying the GUI elements like a button or a label.

The implementation of the GUI part started with the iPhone version, the Java version was done some time later.

Some details of the client side:

The client side is implemented as a class with methods for the GUI elements and other parts. There is one object of that class instantiated at the beginning and when starting the application the implementation of the toclient and fromclient methods is defined.

Toclient encodes the message and uses the instantiated client object as the object and calls the decode message implemented there.

A reference to the fromclient method is set by an appropriate setter call to the client object.

The client side decode method decodes and interprets the messages sent via the message interface

For example for this message:.

M53:9:wtkclient11:createLabel4:obj120:label: Hello Chicago

After decoding we get a Tcl like list with the following contents:

{wtkclient createLabel obj1 {label: Hello Chicago}}

The first two parts build the client objects method to be called (after some mangling): wtkclientCreateLabel and there are two parameters: obj1 and {label: Hello Chicago} for that message.

As iOS and Java both can call class method using a text string with reflection/selectors this is the technique used.

Method wtkclientCreateLabel is responsible for creating a GUI element label with the text: “label: Hello Chicago”

First approach for crating GUI elements was, to use the native GUI elements available on iPhone namely the UI* classes. Using that approach, there is a rather limited implementation of a button and label support. Rather limited in that respect only means there is no completely compatible environment available as for a Tk Button.

Instead of a mouse click there is the possibility to hit the button using a touch screen event. When this event fires a native method is called, which in turn can call another method (in our case come method inside the client class. This method is implemented to forward that “event” to the Tcl wtk part in calling the fromclient method with parameters. Via that way the notification for an event is reaching the Tcl part, which in turn can handle the administrative part of the event and eventually is sending back some other message to be handled, for example to change the text of a button when the button is hit.

5 Different Approach for GUI Elements

Very soon the implementation reached a point (at least on the iPhone side) where it was obvious, that there is a lot of functionality missing, when looking for Tk like widgets. That might be partially because Apple wants to look all their GUI stuff look like they want it to be dispayed.

At that time some experiments with OpenGL ES started. OpenGL ES is a cut down version of OpenGL running on iOS and on Android and as part of WebGL in javascript for browsers too.

The experiments were based on the idea to use screen buffer implementation of OpenGL ES as a base for displaying pixels on the screen and to use some primitive functions of OpenGL ES like drawing a line or a rectangle or a polygon and filling some area with colors. For displaying text the idea is to use a freetype font implementation, also available for the iPhone.

Using that approach, it would be possible to use 3-D elements to be shown and also rotation of text would be very easy using OpenGL Es functionality.

As OpenGL ES is also used as a base for some games on iOS the guess was, that it should be fast enough for the implementation of a Tk GUI.

Having some small knowledge of OpenGL from the implementation of ntk_widget the decision was made to give OpenGL ES a try, to see, what can be done using that.

6 Use of OpenGL ES

The implementation of OpenGL ES for iOS (iPhone) has a rather simple interface to work with. There is an OpenGL graphic context, which can be used to display OpenGL ES primitives like a screen buffer.

OpenGL ES also offers primitives for drawing lines and polygons and to fill areas. Areas are built using for example triangles (there is no support for rectangles, which can be built using two triangles). Lines and triangles are built using vertices. There is an advanced interface available for using arrays of vertices for building graphics elements.

There were some successful experiments in building a Tk button using two triangles for the inner rectangle part and using a combination of some lines for building borders.

It is possible to add an event handling function to be called when a user fires a touch screen event in hitting at some point on the screen. This event also contains the x and y coordinates, where the event happen, Using that information and knowing where on the screen is the area of the simulated button it is possible, to detect when the touch screen event was fired inside the button rectangle area.

When the touch event happened inside the button area it is possible to emulate the Tk like press and release events of a button in setting different border colors for the four borders built using some lines. That way it is possible to make the button look sunken or raised. Depending on the touch screen event.

Experimenting with that implementation the idea came up to eventually use the ideas behind themed Tk (tile or ttk). Looking at the implementation of ttk it seems to be feasible, to implement some modified version of themed Tk widgets using OpenGL ES as the graphic context for displaying the stuff on the screen. Going that way, it would be rather simple too to rotate elements including text. For displaying text there are still some experiments necessary as there is nreal experience yet on how freetype2 fonts are supported on the iPhone (iOS). There is a port/adaption af freetype fonts called freetype2 from David Petrie for iOS and there if a ftgl library called ftgles also from David Petrie which can be compiled, but I was not yet able to make a demo running also it can be linked and started, but the display stays black. Seems to be a problem of adapting to iOS5, as the original was designed for iOS4.

The implementation of themed Tk functionality itself seems to be straight forward, just a matter of doing the work in Objective-C respectively Java.

During testing the implementation there were some problems in building rounded corners for button corners. There have been implemented some different algorithms, but without final success. There were always problems with rendering in getting something looking nice. There is some more time needed to find something suitable, as it seems to be possible looking at iOS button with rounded corners. It has to be found out, if there is a problem with the algorithms used or with OpenGL ES or how to use the same rendering as iOS UI functions/algorithms are using.

7 Status

It seems the suggested way is doable.

It also seems, that using OpenGL ES as the base is a rather platform independent way.

Making work freetype fonts and ftgles to be done.

The complete implementation of themed Tk support is not yet started, it should be relatively easy using enough time to do the work, as the existing implementation for Tk is available.

It seems not to be possible to use native fonts with OpenGL ES.

The implementation of the widgets using OpenGL ES commands is at the beginning.

There is the need for test cases..

There is also the need for examples/demos.