Tools for Developing, Distributing and Using Tcl/Tk Applications over the World Wide Web

K. J. Nash and A. J. Simons
CitizenEarth Internet Ltd
1-3 College Yard, Worcester WR1 2LB, UK.
kjnash at citizenearth dot com

ABSTRACT

Tcl/Tk safe interpreters provide a secure environment for running arbitrary untrusted code. We have developed Web clients in Tcl/Tk that are optimized for running Tcl/Tk applications. We have also developed server-side tools that permit collaborative development of Tcl/Tk code in a Wiki-like environment, but optimized for code rather than text. We combine the advantages of the browser plugin and the Wiki, with added features such as an enhanced security policy, local caching, and remote storage of personal files. The clients can be compared with Adobe® Air® or Microsoft® Silverlight™, but are better oriented to sharing code because by default the client receives the full Tcl/Tk source code.

I. INTRODUCTION

Tcl/Tk provides the ideal framework for the execution and large-scale collaborative development of open-source Rich Internet Applications (RIAs), firstly because untrusted code can be executed in a safe interpreter, and secondly because by default applications are distributed in the form of source code. The Tcl browser plugin [1] is an example of the use of safe Tcl interpreters to execute untrusted code, but it is not used very widely. The Tclers' Wiki [2] is the foremost example of a Wiki dedicated to a programming language, but it is primarily designed for sharing text and example code, rather than self-contained code that is intended for immediate execution.

We have produced [3] a Web-based client/server system that integrates the safe execution environment of the browser plugin, and the collaborative development model of the Wiki (but optimized for code rather than text), while also offering an enhanced security policy, remote storage of personal data, and local caching. These features allow Tcl/Tk applications and their data to be accessed over the network, not only by developers but also by end-users.

II. WEB CLIENTS

The Web client software is written in pure Tcl/Tk 8.5, and makes heavy use of BWidget and Snit. This collection of tools proves well-suited to the modular construction of complex desktop applications. The improvements in Tcl/Tk 8.5 that were found useful include: the enhancements of the text widget; the new HTTP/1.1 capability of the http package, which is used in this project

for all HTTP access; dictionaries; and argument expansion `{*}`.

The Web client is a replacement for the conventional Web browser and the part of the Tcl plugin that is written in C (Figure 1): it reuses the part of the Tcl plugin that is written in Tcl, almost without modification. Although there are much simpler ways to launch slave interpreters from a Tcl application, this design choice has the advantage that it guarantees compatibility with the Netscape Plugin Architecture, and gives us the option for the future of providing our Web client in the form of a plugin for a conventional Web browser. Because our Web resources use different Content-Types from the existing Tcl plugin (Sec. IV), our proposed plugin and the conventional Tcl plugin will co-exist. Indeed it would be straightforward for a single plugin to provide access to both types of resource, ours and the conventional "application/x-tcl", without changing the behaviour of the plugin, including its security policy, when it handles resources of type "application/x-tcl".

The Web client in Fig. 1 (the shaded block in the top and left of the figure) can be split into two parts, a library package that interfaces to the plugin and to the Web, and a GUI application that interfaces to that library package. We have developed five such applications:

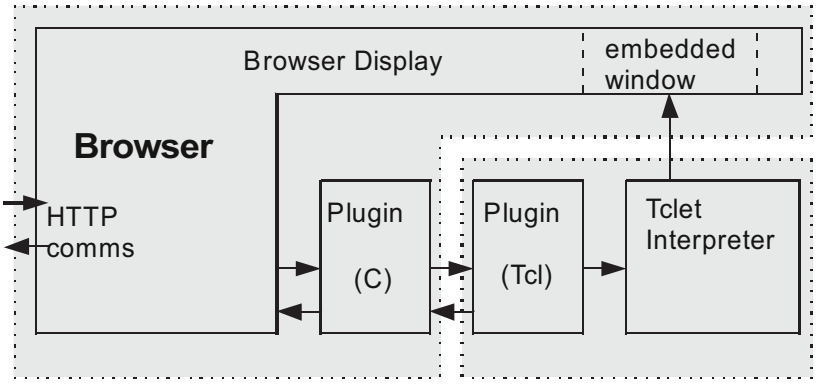

FIGURE 1. Block diagram of a Web browser with the Tcl Plugin. Blocks with solid-line boundaries represent components of the browser/plugin system. The browser interfaces with the plugin shared library (written in C) according to the Netscape Plugin API. This shared library interacts with the part of the plugin written in Tcl, which is responsible for launching Tclet code in a safe interpreter, and for embedding that interpreter's window in the container provided by the browser in its display. Shaded areas with dotted-line boundaries represent components of the present project. The part of the plugin written in Tcl is re-used (shaded block in the lower right), and a pure Tcl/Tk Web client (shaded block in the top and left) replaces the browser and the part of the plugin written in C.

- The Kerlin™ Web Client (KWC) is comparable to a conventional Web browser and is suitable for use on a desktop computer.
- The Kerlin™ Web Desktop (KWD) has the appearance of a window manager, and is capable of full-screen operation
- The Kerlinized™ Tcl/Tk shell (kwish shell) allows execution of Kerlin™ Tclets from the command line.
- The Kerlin™ Application Store (KAS) allows selection of Tclets from a small graphical display, convenient for use in a mobile phone.
- The Kerlin™ Web Editor (KWE) writes as well as reads Web resources.

III. CONTENT FORMATS (A)

In addition to resources that consist of Tclet code, it is convenient for the client to understand two further resource types: a simple Wiki-like markup called "Notes"; and a "Worksheet" format that contains a combination of source code and supporting material.

The "Notes" markup format is a cut-down form of that used by version 2 of the Notebook project by William H. Duquette [4]. The intention of "Notes" is not to rival the capabilities of contemporary HTML, but simply to provide a convenient format for rich-text resources such as directory listings and simple documentation. Major new features are not planned for "Notes", because the emphasis of the project is on developing Tclets, not on expanding the capabilities of "Notes". When developers require rich-text facilities that are beyond the capabilities of "Notes", it is preferable that they develop libraries for use by Tclets.

A "Worksheet" is a document that is structured for loading into an outlining editor. The document contains the full Tcl/Tk source, as well as Tcl/Tk code that is not part of the source (for example, test code), and documentation in a variety of formats including "Notes". Each Worksheet has a "code abstract" facility that extracts the Tcl/Tk source to a .tcl or .ktcl file. By default a Tclet is delivered as this .*tcl file which is the Tclet source; the Worksheet is the Tclet's "metasource".

The Web server must therefore provide HTTP resources in three formats – Tclet code, "Notes" text, and "Worksheet" metasource. Rather than define the last two formats arbitrarily, it is convenient to do so as part of a general framework which we call "Dodecalogical Markup Language".

IV. DODECALOGICAL MARKUP LANGUAGE

Dodecalogical Markup Language (DML) has a number of variants; the one most relevant here is executable DML (EDML). An EDML resource is defined to be one that can be processed by execution in a Tcl interpreter, either a standard interpreter, or a special interpreter that

implements the appropriate “little language”. That interpreter will generally be a safe interpreter because an EDML resource will usually have been fetched from an untrusted source. The term “dodecalogical” refers to the twelve rules that define valid Tcl.

Three dialects of EDML are defined for the purposes of the present project:

MIME Type	Filename Extension	Purpose
application/x-ktcl+edml	.ktcl	Tcl script with Kerlin™ security policy
application/x-dkn+edml	.dkn	Text in “Notes” format
application/x-dkw+edml	.dkw or .dtw	Worksheet with metasource for a .ktcl or .tcl file

A valid EDML resource must begin with a “doctype” statement, for example

```
doctype application/x-dkn+edml 0.9.1 public http://kerlin.org/doctypes/
```

A Web server typically determines the MIME Type of a resource from its filename extension, and sends an appropriate “Content-Type” HTTP header. The Web client decides how to interpret the resource by using the “Content-Type” header and, for an EDML resource, the “doctype” statement.

IV A. Why Use DML?

The present project uses DML as a means of delivering Tclets, their metasource and their documentation in a format that is readily processed by Tcl/Tk.

Another advantage of DML is that it leads to a substantial simplification of the Web application stack. The conventional Web application stack has become rather baroque, using different toolkits in each layer:

- HTML4, HTML5, XHTML
- CSS
- Javascript/JScript/ECMAScript
- Flash
- AJAX: xmlhttp with data transfer in XML, JSON or some other format
- PHP/ASP/JSP/Ruby
- Middleware for PHP/ASP/JSP/Ruby: application framework, template system, CMS, Rails
- Relational Database
    - SQL
    - Database programming language

In contrast, the DML Web application stack is much simpler:

- Tcl
- Database with Tcl abstraction layer

Each layer of the Web application stack needs its own API, but there is not a good reason why it should also have its own programming language.

IV B. Why Use Metasource?

The Kerlin™ Web Editor (KWE), and its Worksheets and metasource, are inspired by Sage [5], GNU TeXmacs [6], and Leo, the "Literate Editor and Outliner" [7]. KWE provides facilities for:

- editing source code
- outlining
- viewing and editing rich-text documentation in multiple formats
- abstracting part (but not necessarily all) the source code from the metasource to a separate "code abstract" file: the "code abstract" is the Tcl/Tk source for which the Worksheet is the metasource. The Tcl/Tk code that belongs to the Worksheet but is not abstracted may include tests, experimentation, etc.
- manual editing and execution of blocks of Tcl/Tk code, and display of the results; such code may or may not belong to the "code abstract"
- executing applications such as tutorials and demonstrations

The mixture of documentation and code is intended to help the developer and keep both kinds of information side-by-side in the same file: these aims are less ambitious than those of Literate Programming [8], and the implementation is correspondingly less elaborate.

V. CONTENT FORMATS (B)

The Tclet format ktcl ("Tclet script with Kerlin™ security policy", or "Kerlinized™ Tcl") runs in a safe interpreter with a security policy similar to that of the conventional Tcl plugin, but with additional facilities made available if this is approved by a user dialog in the main interpreter: for example, a mail client would need to break the "same origin" policy by connecting to a mail server on port 25/tcp. The attempt to do so will be detected by the KWC client, and a dialog will be posted that gives the user the option to allow or forbid such access.

The command "kPolicy" must be the first command executed after "doctype", and provides explanatory text that can be inserted into the policy dialog.

A simple ktcl file is thus:

```
#! /usr/bin/env kwish
doctype application/x-ktcl+edml 1.0.0 public http://kerlin.org/doctypes/
kPolicy {}
button .b -text "Hello, World!" -command exit
pack .b
```

Content formats such as “Notes” are much more restricted. In the safe interpreter for “Notes” there are only two valid commands, doctype and notes, and the security policy cannot be relaxed by a user dialog. In exchange for these constraints, the Tclet that displays a “Notes” document is allowed a little more freedom than the ktcl Tclet interpreter: it may open a new browser tab when the user clicks on a link; it may instruct the client to load a resource from a different server into that tab; and it may write to the client’s status bar, a facility that is useful for reporting errors, and for indicating the URL of a Web link that is beneath the mouse pointer.

A simple “Notes” file is thus:

```
doctype application/x-dkn+edml 1.0.0 public http://kerlin.org/doctypes/
notes -wrap word -title {Help Page} -content {
== Help ==
* [Login Help|login.dkn]
* Back to [FAQs|faq.dkn]
}
```

VI. SERVER FACILITIES

VI A. General

An EDML-capable Web client such as KWC may be used with any Web server that has been configured to send EDML resources with an appropriate “Content-Type” header.

A Web server may make an EDML resource available in two representations (variants): the EDML original and an HTML substitute. Content negotiation based on the “Accept” request header permits conventional Web clients to receive an HTML variant of the resource. Variant representations of a resource are permitted by RFC 2616 [9]. The HTML variant is useful primarily for indexing by search engines, and to encourage users to switch to an EDML-capable client. During content negotiation, the Web server should not believe Web clients that send an Accept header such as

```
Accept: text/html, */*;q=0.8
```

and thereby pretend that they can accept content in any format: the server should respond with EDML only if this is explicitly requested, for example by the request header

```
Accept: */*+edml
```

A Web server must always supply EDML resources with UTF-8 character set encoding, and should use end-of-line characters <CR><LF>. A Web client that accepts EDML resources must be capable of handling these features and may reject encodings other than UTF-8.

An EDML-capable Web-enabled editor such as KWE saves a resource to the server by sending a POST request to an appropriate URL.

VI B. Kerlin™ Web Server (KWS)

The Kerlin™ Web server (KWS) provides a number of additional facilities. KWS is built on Apache and PostgreSQL.

KWS provides versioning, so that no resource is ever lost by being overwritten. The version control model is inspired by Wiki, and is appropriate for managing individual files with weak (or no) dependencies. In contrast, source code management (SCM) systems such as Subversion and Git handle a strongly interdependent set of files, which typically can be combined to build a binary, and the emphasis is on transactional changes that preserve the integrity of the entire set.

KWS uses authentication (Basic Authentication over HTTPS) [10] to enforce access control, and provides four kinds of storage area:

- Private User (owner has read/write access, others have no access)
- Public User (owner has read/write access, others have read access)
- Public Group (members of a group have read/write access, others have read access)
- Wiki Style (everyone has read/write access)

A “Private User” area can be likened to the user’s own disk space. When a file is ready for public access, the user can copy it to his/her “Public User” area and, if desired, to the “Wiki Style” world-writable area.

A user may copy any file that he/she can read to his/her “Private User” area for development. Individual files are copied on request, and although the “Private User” area can be likened to the “working copy” of an SCM repository, there are substantial differences: the equivalents of the “check out” and “commit” operations can be applied only to individual files.

When a file is loaded into the KWE editor and then saved, the “Save” or “Save As” operations can write only to the “Private User” area. A separate action is needed to “publish” the file to a “Public User”, “Public Group” or “Wiki Style” area. While this arrangement cannot prevent deliberate vandalism, it does at least mean that world-writable areas will not accidentally be used as a scratchpad.

This development model is experimental – it is not yet clear how it will work in practice. A Wiki is text-based, and strives to maintain a single authoritative version of each file. KWS might develop along those lines, with such resources kept in the “Wiki Style” area. However, users

might find these world-writable resources too close to the "bleeding edge" and, except in their own particular areas of development interest, prefer to use stabilized resources maintained in a "Public User" or "Public Group" area. It is entirely up to the owners of such areas to decide how to deploy them – many users will want to maintain stable versions of a few files, a few users will want to maintain a stable "distribution" of anything useful that is posted in the "Wiki Style" area.

When a resource has dependencies (cf. the "source" and "package" commands), it is up to the author to specify from which location these should be loaded, and whether to load a particular version or the most recent version of the dependency.

VII. RELATED WORK

By default, Web content is formatted as HTML+Javascript+CSS, the specifications of which are constantly evolving, and the implementation of which is significantly different in different Web browsers. Furthermore, both the HTML+Javascript+CSS format and the Web browser itself have evolved from tools that were never intended for the distribution and deployment of GUI applications (these applications are sometimes referred to as "Rich Internet Applications" (RIA), and the framework as "Cloud Computing"). The conventional Web browser has not yet completed its evolution into a satisfactory platform for executing applications.

For these reasons, among others, several vendors have produced alternative Web content formats for RIAs, such as Mozilla's XUL+Javascript+CSS, Java™ applets, Java™ Web Start, Adobe® Air®, and Microsoft® Silverlight™; each of these formats requires either a specialized Web client ("browser") that can read the RIA content, or a plugin for a conventional Web browser, or runtime commands that integrate the RIAs into the desktop's means of launching conventional applications.

Kerlin™ shows that RIAs can be produced equally well using Tcl with suitable libraries. A Tcl-based system has a number of advantages:

- platform independence
- simplicity
- small footprint
- reference implementation is available as open source
- simple means of execution outside a browser
- content is delivered by default as source code, not as bytecodes or binaries
- Tcl/Tk is designed (among other things) to provide the whole of a graphical desktop application

Although each RIA platform mentioned above has some of these advantages, none has all of them. While the option of bytecode might be preferred for commercial content, our default delivery of source code, combined with the simplicity of Tcl/Tk, is not available to most RIA

platforms, and will facilitate code sharing.

VIII. DISCUSSION

The safe interpreter facility of Tcl/Tk, and the flexibility of its security policies, are substantial assets that are not available to other language platforms. These facilities provide an ideal framework for the deployment of Rich Internet Applications (RIAs). The delivery of Tclets as source code makes Tcl/Tk an ideal platform for code sharing. These qualities, when combined with appropriate Web-based storage, make Tcl/Tk the most suitable platform for the large-scale collaborative development of RIAs.

IX. ACKNOWLEDGEMENTS

Adobe® and Air® are registered trademarks of Adobe Systems Incorporated. Sun™ and Java™ are trademarks or registered trademarks of Sun Microsystems, Inc. Microsoft® and Silverlight™ are trademarks or registered trademarks of Microsoft Corporation. CitizenEarth® and Kerlin™ are trademarks or registered trademarks of CitizenEarth Internet Ltd.

X. REFERENCES

[1] The Tcl browser plugin, http://www.tcl.tk/software/plugin/
[2] The Tclers' Wiki, http://wiki.tcl.tk/
[3] The Kerlin™ Project, http://www.kerlin.org/
[4] Notebook, http://www.wjduquette.com/notebook/
[5] Sage, http://www.sagemath.org/
[6] GNU TeXmacs, http://www.texmacs.org/
[7] Literate Editor and Outliner, http://webpages.charter.net/edreamleo/front.html
[8] Literate Programming, http://en.wikipedia.org/wiki/Literate_programming
[9] RFC 2616, "Hypertext Transfer Protocol – HTTP/1.1", http://www.ietf.org/rfc/rfc2616.txt
[10] RFC 2617, "HTTP Authentication: Basic and Digest Access Authentication", http://www.ietf.org/rfc/rfc2617.txt