



The following paper was originally published in the  
Proceedings of the Sixth Annual Tcl/Tk Workshop  
San Diego, California, September 14–18, 1998

## Visualizing Personal Web Caches with Caubview

Charles L. Brooks, *GTE Internetworking*  
Murray S. Mazer, *Curl Corporation*  
Frederick J. Hirsch, *The Open Group Research Institute*

For more information about USENIX Association contact:

1. Phone: 510 528-8649
2. FAX: 510 548-5738
3. Email: [office@usenix.org](mailto:office@usenix.org)
4. WWW URL: <http://www.usenix.org/>

# Visualizing Personal Web Caches with Caubview

Charles L. Brooks  
*GTE Internetworking*  
*clbrooks@bbn.com*

Murray S. Mazer  
*Curl Corporation*  
*mazer@curl.com*

Frederick J. Hirsch  
*The Open Group Research Institute*  
*f.hirsch@opengroup.org*

## Abstract

Caubview is a companion visualizer for the Caubweb system: Caubweb enables a user to create a local collection of Web documents to read and update when disconnected. Caubview allows the visual selection of alternative views of the cache as well as reorganizing and restructuring these views. This paper describes our ongoing work on the Caubweb system, focusing on its visualization component, Caubview. We describe the relationship between these two systems, how we developed Caubview by re-using code from the HistoryGraph application, our results, and our plans for further development. We further describe our experiences using Tcl/Tk as the development language and "programming culture" for these applications, and indicate how ongoing developments in Tcl/Tk have influenced our work. We conclude with some observations concerning our future use of Tcl/Tk, and recommendations for ongoing efforts for the Tcl/Tk community.

## Introduction

We have been working with Tcl/Tk for over two years, building systems to simplify and improve a user's experience of the World Wide Web. We started with the HistoryGraph visualizer, a tool designed to automatically capture and display a user's browsing history in a tree, and allow manipulation and use of the representation. We then modified and reused this code to provide a visualizer for a more ambitious Tcl/Tk project, Caubweb. The Caubweb system is designed to provide disconnected Web access through the use of cached Web resources. The Caubweb system includes Caubview, the visualizer, and Cobweb, a library of reusable Tcl/Tk components.

Tcl/Tk initially gave us a rapid development environment as well as platform portability and extensibility. As the system became larger and more complex over time, Tcl/Tk also introduced some difficulties. A major issue has been the lack of compatibility from one Tcl/Tk implementation to the next (as an example, the TkNT4.0 supported both a `send` and a DDE command under Windows/NT: both these capabilities disappeared when we moved to Tcl7.6/Tk4.2). Initially we found the need to build custom Wish's for different extensions cumbersome, and we are pleased to now be able to use dynamic loading of libraries. As the system became larger, naming became an issue. Although the namespace facility in Tcl 8.0 solves this problem, we have a large body of older code to re-write. Writing object-oriented code has also been an issue. We used the *obTcl* extension because it was a pure Tcl implementation, but now [*incr Tcl*] seems to be the object system of choice (and *obTcl* is no longer actively supported). Converting to Tcl/Tk 8.0 was relatively painless, and the improvements are noticeable. Native look and feel helps meet our portability goals, and several other problems have been solved. At this time, it is difficult to decide in retrospect if we should have used Perl or some other choice instead of Tcl/Tk. The answer really will depend on what happens with the next release of Tcl/Tk, as well as ongoing work in the Java space with Jacl and Tcl Blend.

This rest of this paper discusses the development of the Caubview visualizer, how Tcl/Tk helped and where it raised issues, and how we addressed these issues by writing specialized Tcl/Tk code or using extensions. We conclude by summarizing some general issues with the Tcl/Tk "culture" (such as missing features, the release process, and a CPAN (Comprehensive Perl Archive Network

[<http://www.perl.com/CPAN/>] equivalent), and discuss alternatives to Tcl/Tk, such as Jacl and the Java Foundation Classes (Swing).

## Background

Caubview began its existence as a way to visualize and manipulate the resources contained in a Caubweb cache: it sprang to life as a quick re-write of the HistoryGraph [Hirsch97a] application, in combination with the Cobweb libraries. Since that time (May 1997), Caubview has become an important application in its own right, particularly in the viewing of shared caches amongst various cooperating Caubweb systems.

Caubweb and Caubview were written in Tcl/Tk from the beginning. Tracking the various Tcl/Tk releases over the life of this project has been both frustrating and rewarding. Choosing Tcl/Tk was not only a choice of a programming language, but a choice of a programming "culture", complete with local idioms, beliefs, attitudes, and styles. The story of Caubview and Caubweb can't be told without focusing on Tcl/Tk as well.

## Caubweb

The Caubweb system is extensively described in [LoVerso97]. The following paragraph is taken from the introduction.

Caubweb is a research system for investigating ways to provide adaptive, on-going read and update interaction with Web-based information, even under conditions of variable or intermittent network connectivity. Caubweb is part of ... [the] Distributed Clients project, which has the broad goal of increasing the availability and customization of Web-based information services for mobile computing users. The expected benefits include increasing the availability of information, reducing the latency of servicing requests, and adapting information to the specific user and context.

At base, Caubweb aims to provide a service more akin to a history-list mechanism than an actual cache (since, in fact, Caubweb violates HTTP/1.1 caching policy [Fielding96] in order to provide the user with a simulated experience regarding resources they have previously accessed). Architecturally, Caubweb is built as an HTTP proxy that uses a cache to meet its goal of providing access to Web resources when disconnected.

As part of our initial goals of platform portability and extensibility, the Caubweb interface design supported cache viewing via a "control panel", implemented as a set of dynamically generated HTML pages that enabled several alternate views of the cache. As part of this interface, the user could obtain a listing of HTTP servers for which resources were available, and, for each server, a list of all URLs retrieved from that server organized alphabetically, and with a additional information such as size. Locally modified documents are marked with a specific icon. Another view list all modified documents in the cache.

Our initial goal in creating the Caubview application was thus to both duplicate and improve on the original cache viewing mechanisms. We wanted to permit the visualization of various relationships among the resources in the cache and to determine the success of an automated retrieval process (that is, whether a particular weblet retrieval had fetched all documents of interest).

## HistoryGraph

The HistoryGraph visualizer is a tool that provides a graphical history of Web browsing activity by creating a tree structure [Brighton97] that reflects the user's browsing activity. Our initial experiences in developing this application with Tcl/Tk are detailed in [Hirsch97b]. At base, HistoryGraph is a desktop browsing associate [Meeks95] that "automatically tracks user browsing activities, presents a graphic visualization of this activity, and provides a mechanism for manipulation and use of that history". The visualizer generates a tree of the user's browsing activity; where each node represents a visited URL and each arc indicates that the user has visited that URL via the URL represented by its parent. The resulting view is not static: a user can graphically reorganize and prune the resulting tree in order to restructure this information, and the resulting tree can be saved across sessions and shared with others. Named sets of pages can be generated by selecting nodes from the tree and assigning them to a named collection; in turn, these sets can be saved for future use or forwarded to other applications for further processing and modification.

## Caubview design

Caubview was derived from HistoryGraph and initially reused its design of a single display window. Development time for initial re-work was a little more than a week. Instead of receiving input from the user's browser and building the tree dynamically, Caubview reads the contents of a Caubweb cache index file on

start-up, and generates an internal representation of the contents. The original goal for Caubview was thus to provide viewing of the cache when disconnected from the network. Initially, no attempt was made to synchronize access to the cache with Caubweb itself, nor were any attempts made to update the view of the cache once the initial index had been loaded.

### Views

A Caubview visualization is organized around a series of "views": a *server* view, a *site* view, and a *closure* view. The initial view that the user sees is the "server" view, that shows a tree of the top level servers, organized alphabetically by host name. Each server view has a corresponding site view that is accessed by selecting a particular server. This view generates a tree of all resources in the cache that are provided by that server, organized by URL name. This is a *syntactic* arrangement, where leaf nodes (actual cache resources) are represented graphically with file icons (📄), and intermediate path names are presented using directory icons (📁). An arc implies that the node is "lower" in the naming hierarchy than its parent, and also implies that that element is resident in the cache. Figure 1 below shows a site view of the cache.

In the *closure* view, a particular resource is selected, and new elements are generated, organized by embedded hyperlinks (we call this a *semantic* or a *link view*). In the semantic view, an arc indicates that the child node is the destination anchor for a hyperlink appearing in the parent node. Two interfaces are provided for generating closures: a "canned" series of closures, and a menu interface that provides more control over selection. A closure is defined by the

depth in the tree to search, whether to include nodes only from the current server or all servers, and whether to display in the same tree of different trees: the interface allows these elements to be set separately.

The results of a closure can be overlaid on an existing window or shown in a new window. The resulting tree can contain "ghosts": resources that are referenced by a hyperlink but are not present in the local cache. Ghosts are drawn using a different icon from a leaf node (📄). Resources from a different server than that of the selected document that are also present in the cache are also shown with a distinctive icon.

### Shared Caches

We recently have extended Caubweb to support the notion of shared caches, defined by a group of cooperating Caubweb servers. Caubweb itself was modified in the following way: if running in "cooperative" mode, and a resource is not found in the local cache, Caubweb would then query one of its defined peers (via an HTTP proxy request) to attempt to discover the resource. We initially use a simple list traversal algorithm instead of a more general flooding

algorithm to determine if the resource is held by one of the cooperating peers. Looping is prevented by the addition of a Caubweb-specific header to the request prior to sending it to a cooperating peer: this prevents the peer from attempting to retrieve this resource from the origin server.

We also modified Caubview to support viewing of the shared caches. When running as a member of a shared cache group, the server view shows an aggregate view of all WWW servers available in all caches. Two icons are used to indicate either a local server (resources are available on the local machine) or a remote server

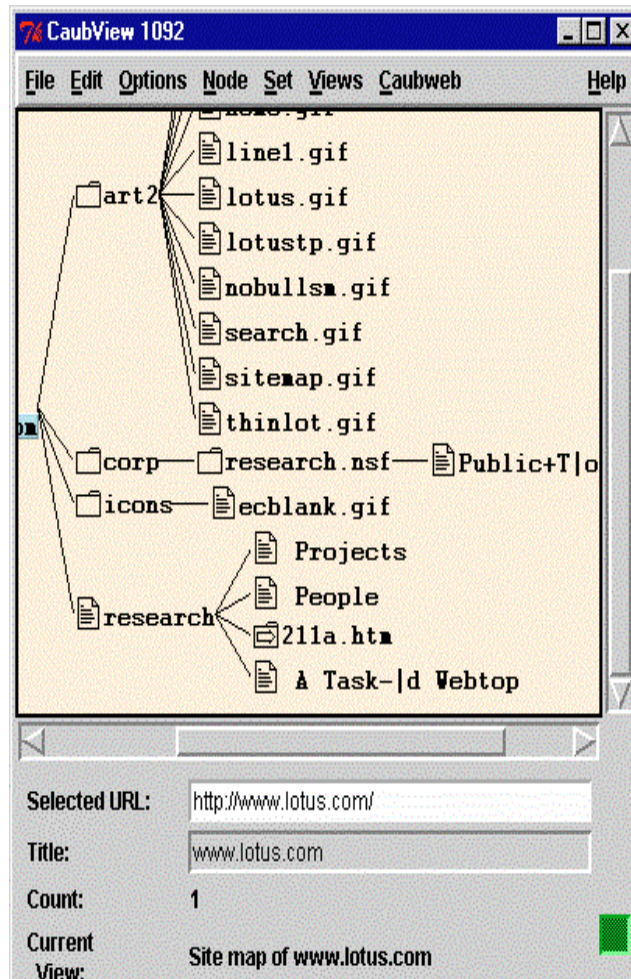



Figure 1 Caubview display of a site view

(resources are only available remotely). Servers for which resources are stored both locally and remotely are marked as local. The site views are modified accordingly to show all resources, whether local or remote: in this view, remote resources are displayed with an icon reminiscent of a pointer, and the URL is used to name the resource instead of the document title (assuming it exists). Closure views are extended such that if the search depth indicates that resources should be analyzed for embedded links, the resource is fetched from the remote system and its link set is extracted from the HTML source. Referring back to Figure 1, the node labelled with a  icon (and named 211a.htm) is a reference to a remote resource.

### Initial Results

Our initial experiences with the Caubview visualizer were mixed. We were disappointed by the initial reaction of people to whom we demonstrated the prototype. Most users responded positively: they were better able to understand the cache layout, and appreciated the restructuring capabilities of Caubview, since it retains most of the capabilities of HistoryGraph such as generating sets, manipulating the tree via drag-and-drop, and viewing resources via a specified browser (the browser is configured to proxy through Caubweb, which is running in off-line mode).

However, several users were confused between the semantic and syntactic views (especially when overlaid in the same window) as to whether or not the indicated resource was present in the cache. We believe that this was caused by differences in meaning of the arc relationship: in one case, an arc indicates a *naming* relationship (as well as existence); in another instance, it indicates a *linking* relationship (irrespective of cache residency). We are currently evaluating ways to better represent these relationships, possibly by using color or font changes to represent cache residence.

### Ongoing Work

We are currently working on two issues: how to better visualize the cache hierarchies, and how to visualize metadata associated with individual resources and aggregates (servers, document collections [a set of resources meant to be viewed as a whole]), or metadata shared across elements). Underlying these investigations are the deeper issues of *navigation* versus *exploration*. For us, navigation implies both context and ways to generate various traversals of the information space based on existing hyperlinks. Exploration, on the other hand, involves the generation of new hypertexts (relationships, or the results of

queries) by exploiting communality amongst resources.

### Cache Visualization

When thinking about cache visualization, the first question that comes to mind is: How much of the hierarchy needs to be in view at any one time? Although our existing tree widget supporting panning of the display as well as scrolling, it is constrained by the demands of screen real estate, which is a constant tradeoff between showing multiple entities whilst still enabling the identification of those entities by either the last component of the URL path (the "file name", as it were) or by the compression of the title for the resource (providing one is given). One possible extension we have considered is to enhance the existing tree widget to support collapsing of individual subtrees. We have also evaluated using the Pad widget from Pad++ [Bederson96] or a hyperbolic display widget [Lamping95][Munzner95] as a way to represent more nodes and relationships within a given space.

We are also exploring linking to other programs (hypertools) via automatic methods, either via a publish/subscribe model of sets (where Caubview would define a selection of sets and their description, and other tools could subscribe to these sets), or via a more generic model, where certain hypertools announce their availability to provide certain services, and Caubview can then avail itself of these services. We have already experimented with the latter approach in the HistoryGraph application using the WhatsNew hypertool: this tool interrogates a list of resources and determines if they have (or have not) changed since a specified date and time, and provides a list of resources that meet that criteria. The response is returned to the HistoryGraph application by invoking a callback function, that results in a new named set being generated. Selecting this set via the menu bar will cause all resources in the tree to be highlighted. The user can then navigate through these resources and view the changed pages via their chosen browser. This capability is still latent in the Caubview application, but a more generic API needs to be defined.

Finally, given alternate ways of visualizing the local cache, we must then determine how the actions of pruning, hiding, and selecting would function in this new context. For example, given an infinite display space, pruning might be achieved by "unhooking" the subtree and moving it off-screen.

## Metadata

A second focus in the next phase of our work is in the representation and visualization of metadata stored with Web resources. This metadata can include information provided by a shared descriptive framework (such as that defined by the Dublin Core metadata framework [Dublin98]), or can be made available by specialized servers that provide organization-specific characterization of resources, such as relevancy to a particular task or role. One possibility is to use extra gestures (e.g. right click brings up metadata window, with information organized as a property list) as a means of directly viewing associated metadata. We are also evaluating the use of the Pad++ techniques of portals and "magic lenses" as alternatives. Using these techniques, a user would choose a selection criteria, and resources matching that criteria would be either highlighted in the display or transformed in some lens-specific fashion. All this activity is basically an application of the Visualization Mantra: *Overview, zoom and filter, then details on demand.* [Shneiderman97]

## Successes

Once again, direct manipulation of a view (in our case, the cache hierarchy) has proven to be an effective and powerful UI technique. Unlike many Model/View based systems, manipulation of the *view* of the cache actually doesn't modify the actual contents of the cache: rather, what is modified are the *relationships* amongst the various cached resources. Thus, re-drawing of the tree via drag-and-drop becomes a way to provide "visual commentary" - creating (pseudo) hyperlinks (and hence personal associations) where none previously existed. The ability to save the modified tree for further manipulation or to define sets of resources that can be handed off to other applications is another powerful feature that supports re-use of existing hypertools, as well as supporting such search related tasks as history and progressive refinement [Shneiderman97].

While we agree on the need for better visualization tools to describe hypertext based information systems, we are still debating the issue of what works best in what situation. Are 3-dimensional representations more effective? 2.5 dimensional? Collapsible Trees (hierarchical) displays? We see our on-going challenge as answering the latter question in the context of our overall goal: namely, to provide our users with better information regarding the precision and relevance of their cached resources to their immediate tasks when

disconnected from their network information servers. We believe that providing this context will consist of

- visualization (and direct manipulation) of hyperlinked information spaces;
- making metadata available as part of visualization, and
- the display of self-organizing (via metadata specifications) collections versus user-defined collections (based on some arbitrary criteria, whether that be content, metadata, or ad-hoc characterizations).

## Tcl/Tk: Programming and Culture

Caubweb and Caubview have been based in Tcl/Tk since the beginning of the project. As such, we have gone from TkNT4.0 and Tcl7.5/Tk4.1 through the current Tcl/Tk 8.0 release. As is the case with several other programming languages (Perl and Python come to mind), Tcl/Tk is both a programming language and a programming culture, that provides its own sets of idioms, beliefs, and patterns of development. Our initial decision to move forward with Tcl/Tk was based not only on our desire for a rapid development environment, but also on our desire for platform portability and extensibility: our project requirements targeted Unix and Win32 systems (Windows/95 and Windows/NT) from the beginning.

## Code Reuse

We were gratified by the amount of code reuse between the HistoryGraph and the Caubview application (more than 75%), as well as the ease of developing our initial implementation. We have had to re-write some of the code in porting from TkNT4.0 to Tk4.1 and thence to Tk8.0, specifically in the area of sockets and communication with other processes under Windows/NT (specifically Netscape Navigator and Internet Explorer). We have successfully used several other Tcl extensions, including Alan Brighton's Tree widget for display, and Tcl-DP [Perham97] for UDP and IP multicast sockets. In the former, we rebuilt the widget for Tcl8.0 ourselves; in the latter, we imported a binary distribution for Tcl8.0, and have since had occasion to rebuild the system on both Unix (Linux) and Windows/NT, largely without incident.

Both of the above extensions make good use of the package facility, although neither as yet utilizes the namespace facility as a means of preventing namespace collisions. We advocate strongly that any extension writer begin to use the namespace facility, especially when writing internal support routines. We

have had to track down a couple of problems with duplicate definitions of `lremove` or `ldelete`, and while such bugs are not difficult to fix, they do take time from other more productive tasks.

Another important aspect of code reuse is the ability to integrate facilities from different applications in order to create new ones. We were able to easily replace home-grown, ad-hoc code in the HistoryGraph with code from the Cobweb library (Cobweb is the shared library portion of Caubweb), specifically code to parse URLs and HTML source. Problems arose, however, when we came to do the supporting shared caches. As long as Caubview had functioned as an *off-line* visualizer, the issue of cache modification was moot: the cache would effectively never change (modulo the modification of existing resources, which would have created a separate cache entry representing the original resource). By using Caubview as a visualizer when Caubweb itself is active, we require that Caubview be kept up to date regarding new entries in the cache. Testing cache index modification times, and re-reading and re-generating the cache is not an alternative, since Caubweb's internal index is updated each time a new resource is fetched from the network, and only written to disk when necessary. We anticipate adding a Observable/Observer implementation to Caubweb such that Caubview can register as an observer of the cache, and have Caubweb notify it when a new entry is made in the cache. This will require code changes such that Caubview instantiates a cache object if stand-alone mode, otherwise, it must register with the running Caubweb application. We have added a command line parameter to Caubweb to allow Caubview to be started along with Caubweb: Caubview in turn can be started with a switch that indicates it was started from Caubweb (once the `send` capability is available under Win32, we can then determine if there is an interpreter named "Caubweb" present in our environment).

### Porting to Tcl/Tk 8.0

In January of 1998, we began a port of the existing Caubweb and Caubview applications to Tcl/Tk 8.0 (hereafter Tk8.0). Both had previously been using Tcl7.6/Tk4.2. The port to Tk 8.0 was relatively trouble-free. We re-wrote our *obTcl* library (a small, pure Tcl object-oriented extension) to remove the `::` separators used to indicate class methods and instead used a more Java-like syntax using `.` as a separator. We also needed to modify the use of `upvar`-ed arrays: this was accomplished by generating a similar name in a separately defined *obTcl* namespace. The resulting code was highly portable between Tcl7.6 and Tcl8.0

(although the 7.6 implementation has not been exhaustively tested). Both the Caubweb and Caubview application seems faster, although we have not conducted extensive measurements.

We have welcomed the addition of several new features in Tcl8.0, including the support for binary data, and the newly legitimized `fcopy` (*nee* unsupported0) command for Caubweb. Caubview has seen the most improvement, however. The native look and feel support in Tk8.0 has made the user interface more consistent and palatable, and the new font specification mechanisms has made font handling more consistent and the results less "surprising".

### Tcl/Tk futures

In the time that we have been using Tcl/Tk for the Distributed Clients project, we have observed several changes in Tcl/Tk that has made the language more suitable for use in a larger projects. Chief amongst these changes has been the increased and better use of the `package` and `namespace` features added to the language in Tcl7.6 and Tk 8.0. Prior to these features, one had to resort to a "surfeit of Wishes" via a series of statically linked extensions. Today, most extension writers are using dynamically loadable packages, and increased use of the `package provide` and the `namespace` facilities to provide better separation of function as well as better distribution of executable code (Don Libe's CGI package is an example). We are modifying Caubview and Caubweb to use these features more extensively as we re-work various parts of the system.

While Tk 8.0 has been an major improvement for us, we are still waiting for functionality that was available almost two years ago in TkNT. We eagerly await the release of Tk8.1 with support for the `send` command under Win32, as well as DDE support. Another feature we would dearly welcome is a working version of the capability to embed another applications window inside of Tcl/Tk under Win32, since this would provide us with another mechanism with which to integrate Caubview with Caubweb.

### Jacl and TclBlend

As part of our ongoing visualization work with Caubview, we looked at hyperbolic graph widgets, and found at least two written in Java. Our problem was how to integrate Java widgets into our Tcl/Tk application, and with the 1.0 release of Jacl and TclBlend in early 1998, an answer seemed imminent. TclBlend supports the inclusion of Java classes in a Tcl program [Johnson97], while Jacl is a Tcl

interpreter written in Java [Lam97]. However, Jacl 1.0 doesn't support Tk at all, and portions of Tcl (primarily dealing with asynchronous file I/O) are missing as well. In addition (as of April, 1998), the TclBlend extension is advertised not to work with the upcoming Tk 8.1 release (which we need for the `send` command and DDE support for Win32), and also requires a native thread implementation of Java, that constrains our choice of target systems. This means that if we are to continue to leverage our investment in the Cobweb library, as well as gain the new functionality we want in the 8.1 release, we are forced to abandon Java integration with Tcl Blend; if we choose Jacl, we will have to re-implement certain portions of the library that requires Tcl functionality not present in the 1.0 release.

The recent 1.0 release of the Java Foundation Classes (*aka* Swing) further compounds our dilemma, since Swing provides a number of useful Java widgets for building applications. Our current plan is thus to reimplement Caubview using Jacl, using Tcl Blend as an intermediate step. We will re-write the UI portions to use JFC, and work on integrating as much of the Tcl functionality as possible. In this way, we hope to (yet again) leverage our investment in generating platform portable and extensible code.

### **Back to the Future**

We do not regret our decision to use Tcl/Tk, but we are not sure we would make the same decision again, especially with the advance of Java in the last two years. The problems we have encountered with Tcl/Tk over the years can be summed up as follows:

- Ongoing problems with integration of new extensions (installation scripts, namespace collisions). In addition, interfaces to various components are all just a little bit different (a PAD widget is almost a Tcl canvas widget, but not quite). There are also performance differences (sometimes vast) between extensions under Win32 and Unix.
- Each release of Tcl breaks some amount of working code
- Feature lag between platforms (e.g. `send` under Win32), and timely availability of older extensions.
- No coherent O-O strategy (*incr Tcl* seems to be the O-O package of choice, but, again, it lags the field).

Finally, it is still much too hard to determine what extensions are available, or to ascertain the quality of

these extensions. It is our fervent wish that the Tcl community can create an organizational equivalent to CPAN (perhaps ETEN, the Exhaustive Tcl Extension Network)).

For us, cross-platform functionality *and* equivalent performance are critical for the ongoing success of Tcl/Tk, and we urge developers of Tcl extensions to provide support for both Unix and Win32 platforms. There is increasing evidence that Windows/NT will continue to gain importance even within the research community. Our own work environment combines Win32 desktop machines linked to Unix file and compute servers, and we expect this situation to become even more common.

Given the above, how should Tcl/Tk development proceed in the immediate term? We believe that in order to be successful, Tcl needs to return to its roots: a small, easy-to-use embedded interpreted language, acting as a "glue language" for scripting other components. Just as Tcl/Tk provided this capability for the combination for C and the X graphical widgets, we believe that Jacl can provide this capability for Java and the AWT/JFC widget set. The combination of Tcl scripting and Java components thus holds great promise for stemming the tide of Wintel hegemony, and providing a powerful cross-platform solution for application developers.

### **Acknowledgments**

This research was supported in part by the Defense Advanced Research Projects Agency (DARPA) under the contract number F19628-95-C-0042. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, or the Defense Advanced Research Projects Agency or the U.S. Government. The research efforts upon which these findings are based was performed at the Open Group Research Institute.

Special thanks are due to John LoVerso and Scott Meeks.

Caubweb is a trademark of The Open Group Research Institute. Other trademarks are the property of their respective companies.

### **Availability**

The Distributed Clients project has completed. The last release of Caubweb is available at [http://www.camb.opengroup.org/RI/secweb/dis\\_clients/oasis.html](http://www.camb.opengroup.org/RI/secweb/dis_clients/oasis.html). Caubview remains a moving target: those



interested in more information should contact the authors.

## References

[Bederson96] Bederson, B., Hollon, J. D., et. al. *Pad++: A Zoomable Graphical Sketchpad For Exploring Alternate Interface Physics*.  
<http://www.cs.unm.edu/pad++/papers/jvlc-96-pad/index.html>

[Bederson97] Bederson, B., Hallon, J., "A Zooming Web Browser", *Proceedings of SPIE Multimedia Computing and Networking* 1996, Volume 2667, pp 260-271  
[ftp://ftp.cs.unm.edu/pub/pad++/spie96\\_html.ps.gz](ftp://ftp.cs.unm.edu/pub/pad++/spie96_html.ps.gz)

[Brighton97] Brighton, A., *Tree-4.0.1 - A Tree Widget for Tk4.0 based on C++ and [incr Tcl]*,  
<http://www.neosoft.com/tcl/ftparhive/sorted/devel/tree-4.2.README>

[Dublin98] *Dublin Core Metadata*,  
[http://purl.oclc.org/metadata/dublin\\_core/](http://purl.oclc.org/metadata/dublin_core/), valid as of 4/6/98.

[Fielding96] Fielding, R., et.al, "Hypertext Transfer Protocol: HTTP/1.1", *World Wide Web Journal*, 1(4), Autumn, 1996, O'Reilly and Associates, pp. 89-186.

[Hirsch97a] Hirsch, F.J., Meeks, W.S., & Brooks, C., "Creating Custom Graphical Web Views Based on User Browsing History", *Poster Proceedings, 6th International World Wide Web Conference*,  
<http://www.opengroup.org/www/waiba/papers/www6/hg.html>

[Hirsch97b] Hirsch, F.J., "Building a Graphical Web History using Tcl/Tk", *Proceedings of the 5th Tcl/Tk Workshop*, USENIX Association, pp. 159-160.

[Johnson97] Johnson, R., *Tcl and Java Integration*,  
<http://www.sunscript.com/java/tcljava.ps> (as of 3/30/98).

[Lam97] Lam, Ioi K., Smith, Brian C., "Jacl: A Tcl Implementation in Java", *Proceedings of the 5th Tcl/Tk Workshop*, USENIX Association, pp. 31-36.

[LoVerso97] LoVerso, J., & Mazer, M.S., "Caubweb: Detaching the Web with Tcl", *Proceedings of the 5th Tcl/Tk Workshop*, USENIX Association, pp. 19-29.

[Lamping95] Lamping, J., Rao, R., Pirolli, P. (1995) "A Focus+Context Technique Based on Hyperbolic Geometry for Visualizing Large Hierarchies", *Proceedings Chi '95*, ACM SIGCHI, New York.

[Meeks95] Meeks, W.S., Brooks, C.L., Mazer, M.S. "An Architecture for Supporting Quasi-agent Entities in the WWW", *Intelligent Agents Workshop Proceedings*, ACM Conference on Information and Knowledge Management, December 1995,  
<http://www.opengroup.org/ww/waiba/papers/CIKM/CIKM.htm>

[Munzner95] Munzner, T., Burchard, P., (1995) "Visualizing the Structure of the World Wide Web in 3D Hyperbolic Space", *Proceedings of the First Annual Symposium on the VRML Modeling Language*, ACM SIGGRAPH, New York, pp. 33-38.

[Perham97] Perham, Mike, et. al., "Redesigning TCL-DP," *Proceedings of the 5th Tcl/Tk Workshop*, USENIX Association, pp. 49-54.

[Ousterhout93] Ousterhout, J. and Rowe, L.A, "Hypertools: A GUI Revolution," *The X Journal*, 2(4), March-April 1993, pp. 74-81.

[Shneiderman97] Shneiderman, B., *Designing The User Interface*, 3rd edition, Addison-Wesley, 1997, p. 523.