The following paper was originally published in the
Proceedings of the Fourth USENIX Tcl/Tk Workshop
Monterey, CA, July 10-13, 1996

Managing Complexity in TeamRooms, A Tcl-Based Internet Groupware Application

Mark Roseman
University of Calgary
Calgary, Alta, Canada

Managing Complexity in TeamRooms, a Tcl-Based Internet Groupware Application

Mark Roseman
Dept. of Computer Science, University of Calgary
Calgary, Alta, Canada T2N 1N4
Tel: +1-403-220-3532
E-mail: roseman@cpsc.ucalgary.ca

Abstract

This paper describes TeamRooms, a Tcl-based real time groupware application that provides "network places" for users to collaborate. TeamRooms is significantly more complex than previous groupware applications, providing not only generic tools such as shared whiteboards, but also custom groupware applets running within an OpenDoc-style embedded window. As well as describing TeamRooms itself, the paper relates the use of several Tcl programming techniques — meta-architectures, multiple interpreters, and embedded windows — that are used to manage the resulting complexity of the system.

Introduction

This paper describes a novel groupware application called TeamRooms, written using Tcl/Tk. Groupware systems provide a means for several users to work together, even though they may be separated by distance. TeamRooms approaches this problem by providing "network places" on the Internet, where users can gather to meet in real-time or can asynchronously leave information for each other. The metaphor is based on the physical team rooms used by many co-located work groups [4].

Previously, we had developed a number of applications in GroupKit, a Tcl/Tk extension or toolkit we had developed for building groupware [7]. TeamRooms was somewhat of a departure from these applications, demanding a different network architecture, more provisions for security and robustness, and needed to go cross platform. The user interface was to move from the relatively straightforward model of "one tool per window" to a model where several tools could exist in a single window, as found in compound document architectures such as OpenDoc or OLE [5].

In developing TeamRooms, we were faced with the following constraints: there was not enough time or resources to just completely rewrite everything, and it was important to keep the ease of building applications found in the original GroupKit. Even though the entire system was becoming much more complex, that added complexity had to be carefully managed and controlled.

This paper consists of two parts. The first part provides some background on real-time groupware and GroupKit, and then carries on to describe TeamRooms and its user interface. The discussion emphasizes the novel aspects of TeamRooms as a Tcl/Tk program: it is multi-user, multi-process, and an example of a highly-interactive Internet application. Its combination of several smaller Tcl programs with a compound document interface is also new.

The second part of the paper describes how TeamRooms was constructed, while still keeping our investment in existing GroupKit code and its straightforward API. The techniques used include meta-architectures, multiple interpreters, and embedded windows. Because some of these techniques may be applicable to managing the complexity in other Tcl/Tk programs, some problems that were faced along the way are also described.

About Groupware and GroupKit

Before delving into TeamRooms, some background is necessary. For those unfamiliar with the domain, this section first introduces real-time groupware applications. It then describes our GroupKit extension, and in particular the scope of applications which it has been possible to create with GroupKit.

Real-Time Groupware

Groupware is software that helps two or more people collaborate. It is a pretty general category that includes

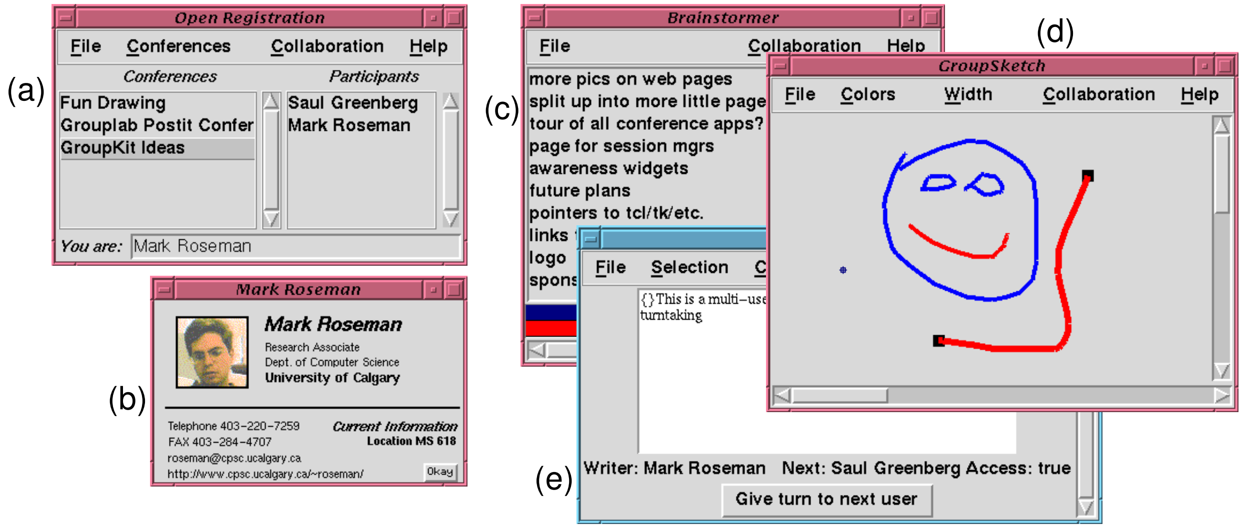

Figure 1. Some GroupKit applications, including (a) the open registration session manager, (b) information on a user, (c) a brainstorming tool, (d) a shared whiteboard, and (e) a shared text editor.

applications like e-mail or Usenet bulletin boards. Workflow and document management systems like Lotus Notes are the most commonly known groupware applications today.

Real-time groupware is groupware that lets people work together at the same time. A common example is the "talk" facility in Unix. Another example is a "shared whiteboard" program, that let people across a network draw together — any drawings marks made by one user on their computer are seen by all other users working on the shared drawing. Other examples are text editors that allow editing the same document at the same time (usually with some form of locking so users don't conflict), brainstorming or voting tools for distributed meetings, card games, and so on.

GroupKit

Groupware can be both productive and fun to use. It is not, however, much fun to write. Even ignoring the considerable technical hurdles of network infrastructures and concurrency, there are many human factors issues that have to get worked out for anyone to be willing to use it. We developed GroupKit to make it easier for programmers to build real-time groupware applications. GroupKit is a toolkit or extension that relies on lower level support from Tcl, Tk, and Tcl-DP. Some of the facilities it provides to groupware developers are message passing, shared data structures, session management, and high-level multi-user interface widgets.

Figure 1 shows some typical applications constructed with GroupKit. The session manager is used to start each tool, which runs as its own process in its own window. When several users join a groupware session (for example, a shared whiteboard tool), each user's process makes a socket connection to every other user's process, which is known as a replicated architecture. Though GroupKit supports many different tools and even different session managers, the basic run-time architecture is always the same.

Just as Tcl/Tk have made single-user applications easy to build, GroupKit has made groupware applications easy to build. The toolkit's learning curve is quick to climb, making it suitable when time is limited, such as for university class projects. It has been used at a large number of sites, and a number of substantial systems have been built using it. Its design has also made it easy to transform many existing single-user Tcl/Tk programs into groupware. The toolkit has also served well in supporting our own research interests of exploring groupware user interface issues. The combination of high-level programming constructs and ease of learning have made GroupKit one of the most popular groupware development platforms available today.

Challenges

Still, there were areas we wanted to explore where we were hindered, particularly as we started focusing more on interesting applications. Besides running on Unix, we wanted to be able to deploy applications across platforms like Macintosh and Windows. Our fully replicated network architecture worked well in a world of stable workstations and networks, but can be problematic with unreliable machines and modem connections. Finally, we wanted to explore richer, more integrated environments, where several groupware tools were closely tied together, for example embedded inside other applications, documents or web pages.

TeamRooms

TeamRooms is our most ambitious groupware application to date. Unlike most of our tools which support isolated real-time meetings, the system provides a fully persistent environment for collaboration, whether in real-time or asynchronously. TeamRooms is modelled after physical team rooms, which provide a place for teams to meet, work, leave things for other team members, add comments and changes to shared documents, and so on. Our goal is to provide an electronic equivalent for teams whose members may be distributed. TeamRooms is a "network place" that hosts a team's collaborations.

This metaphor is not new; Multi-User Dungeons (MUDs) also provide a persistent shared space, where people can meet in rooms containing various objects [3]. As with MUDs, TeamRooms uses a central server to hold information on rooms and their objects, and a separate client provides the user interface — but rather than a telnet client, TeamRooms has a full graphical interface (on Unix/X or Macintosh, with Windows under development). We wanted to move beyond the limited text-based interfaces of today's MUDs, and provide "useful" fully interactive groupware applications as tools in the room. We stopped short of full audio/video support to keep network requirements reasonable, though an external system could be added.

User Interface

Figure 2 illustrates the user interface of the TeamRooms client, where the user (Mark) is in a room called "Mark Roseman's Room" with two other users (Carl and Saul). Along the bottom of the screen are a text-based chat tool and different colored pens for drawing on the "walls" of the room (a shared whiteboard). User snapshots show who else is in the current room or on the server, and if a video camera is available, these pictures are periodically updated. Also shown are six different applets: a URL reference, an image, a concept map, a postit note, an outliner, and a tetrominoes game.

Applets

Each applet is embedded in its own frame, in a similar fashion as OpenDoc or OLE components. Users select new applets from the Tools menu, as well as delete, move and resize them. All changes are immediately visible to all users in the room. TeamRooms also allows users to retrieve earlier versions of applets, to compare changes over time.

Applets can be practically any groupware application, such as meeting tools (e.g. for brainstorming ideas or voting), shared document editors, drawing tools, or games. Some specific examples we built include:

PostIt. The ubiquitous yellow sticky note allows users to leave text messages in the room for other users, as reminders, or to comment on other room objects.

Outliner. A hierarchical outline tool lets users organize a set of notes or ideas. Users can add or delete ideas, drag existing ideas to rearrange them in the outline, and collapse or expand portions of the outline.

Image Tool. As a way to decorate rooms, we created the image applet, which displays a GIF image fetched from an HTTP server.

Games. The tetrominoes applet is one simple groupware game; others could include card games, chess or checkers, and so on.

URL References. To help tie in external information, this applet lets users leave pointers in the form of a URL for others. Clicking on the applet loads the requested URL into their web browser. Another applet uses Stephen Uhler's infamous HTML parsing library to display a web page inside TeamRooms for discussion.

Applets differentiate TeamRooms from most groupware tools that provide only simple facilities such as chat rooms or shared whiteboards. Applets allow the environment to be customized to suit the team's specific needs. Because we expected many users to want custom applets, we needed to make it easy to construct new ones, ideally as easy as constructing normal GroupKit applications.

Summary

TeamRooms provides a shared "network place" on the Internet where team members can collaborate, either in real-time or asynchronously. As a Tcl/Tk based Internet application, it is novel because of its multi-user, highly interactive nature, and its use of OpenDoc-style custom applets embedded inside the application.

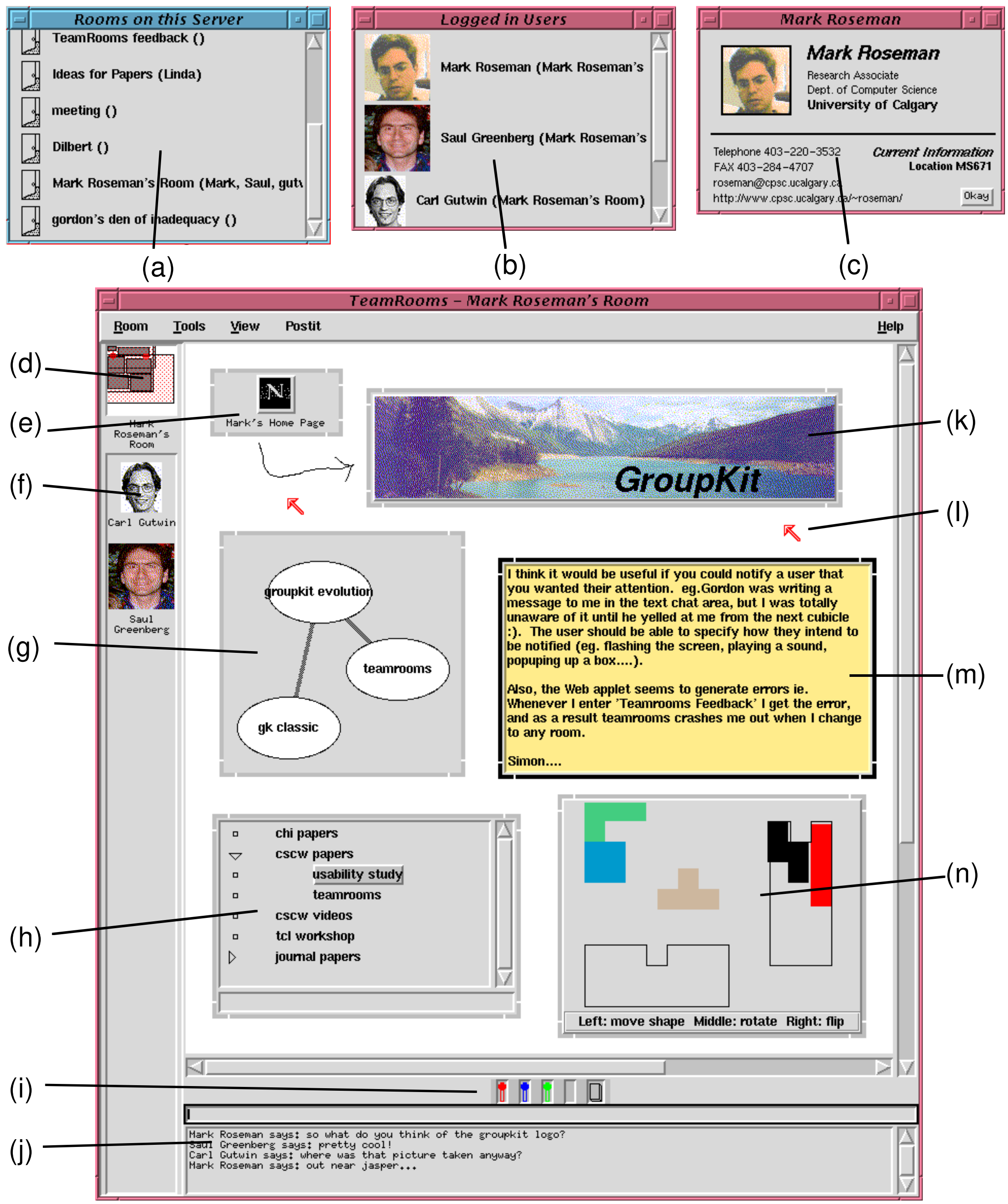

Figure 2. TeamRooms user interface, showing (a) available rooms; (b) connected users; (c) business card; (d) radar overview of room; (e) URL reference applet; (f) users in room; (g) concept map applet; (h) outliner applet; (i) whiteboard pens; (j) text chat area; (k) image applet; (l) telepointer; (m) postit applet; (n) tetrominoes applet.

Strategies for Managing Complexity

TeamRooms represents a rather significant challenge for a GroupKit application. Its architecture is centralized, not replicated; it requires user authentication; it demands a very robust, multi-versioned persistence facility; it needs to be multi-platform; and several groupware applications need to be embedded in the same toplevel window.

This section describes three techniques that were used to build TeamRooms while still leveraging the existing GroupKit code base and API where possible: meta-architectures, multiple interpreters, and embedded windows. After a description of these techniques, some of the particular issues that were encountered in building TeamRooms are addressed.

Meta-Architectures

Meta-architectures provide a way to change the underlying behavior of a software system while still retaining an existing interface or API. For example, we wanted to provide a centralized network architecture (new behavior), though still allowing developers to view the system as having direct connections to other processes for passing messages (a key component of the API).

In a meta-architecture, the user level API calls a small number of well-defined underlying primitives. The meta-architecture provides hooks to allow replacing those primitives. In GroupKit, we had primitives for opening, accepting and closing sockets, and passing messages. The existing primitives supporting a replicated architecture were replaced with new ones for a centralized architecture, and the user level routines continued to do the right thing. When it came time to add authentication (i.e. logins), we could again use the hooks to add the new behavior.

Building good meta-architectures comes down to good software design. It happens that highly dynamic languages like Tcl make them easy to implement. A more in-depth discussion on meta-architectures in Tcl is provided in [6].

Multiple Interpreters

The main problem for TeamRooms is dealing with all the different pieces: locating and navigating rooms, tools such as the shared whiteboard in the room itself, and then the numerous applets. Everything needs to be kept fairly separate and modular, while still being bundled together in the same application process.

Our first approach was to use an object system. A prototype of TeamRooms was built using [incr Tcl], where each applet was a mega-widget with groupware facilities added. While this worked, for this particular application it was not the ideal solution for two main reasons.

New Programming Model. Using an object system introduced a new programming model, where each groupware tool was an object. This added an extra level of complexity that we thought would be an obstacle to our target audience, most of whom are not experienced programmers or familiar with languages like C++. GroupKit's existing message passing paradigm was hard to resolve with objects, and imposing a particular structure on applications would impede the ability to adapt single-user applications.

Modularity Concerns. Surprisingly, modularity was also a concern. The burden was on the object's developer to ensure it did not use globals or otherwise interfere with other objects running in the application (despite interacting with its equivalent objects in other users' processes). This also had implications for security; though we were not immediately concerned with applets being downloaded over the network, the need to "trust" each object to interact nicely with the system seemed to preclude the possibility.

For the final version of TeamRooms, we abandoned objects and implemented the system with multiple interpreters, using the "stcl" extension that was added to the core in Tcl 7.5. Multiple interpreters allow us to view each piece as a completely separate groupware application that looks almost exactly like standard GroupKit code.

The TeamRooms client application consists of several GroupKit interpreters, as shown in Figure 3. The overall *application interpreter* is logically connected (via the central server) to all other users on the server, and deals with navigating between rooms, finding who is logged in to the server, and what rooms are available. When the user enters a room, a *room interpreter* is created to manage the overall room. Logically connected to all users in the room, this interpreter manages the chat tool, shared whiteboard, creates and resizes applets, etc. Finally, each applet runs in its own *applet interpreter*. Some advantages of this technique are noted below.

Standard Tcl Programs. The main advantage is that each component looks just like a standard "run-on-its-own" Tcl application (or in this case, GroupKit application). There are no extra constructs, and no special considerations to worry about. This perfectly addressed

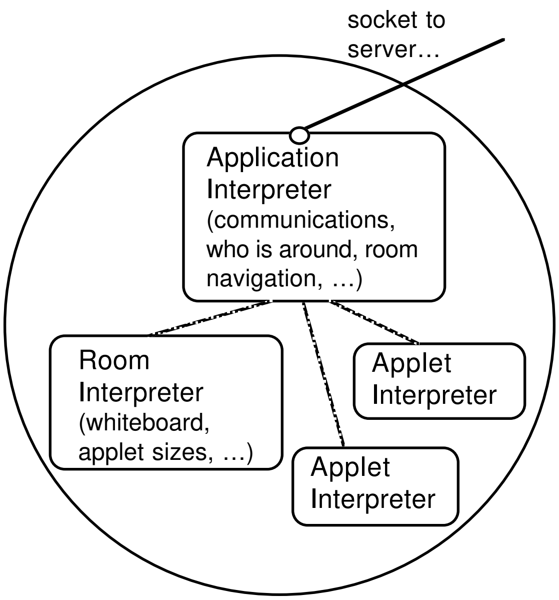

Figure 3. Structure of TeamRooms client, showing use of multiple interpreters.

our concerns with learning curve, and preserved our investment in existing applications.

Modularity and Security. Unlike with objects, programmers using multiple interpreters must explicitly go out of their way to access code out of the program's scope. This meant no accidental interference between applications. By providing a clear dividing line between pieces, it also makes it easy to replace pieces, such as the program for the room interpreter. Finally, this left open the possibility to enforce security restrictions on applets, using the model supported in Safe Tcl [2].

Shared Resources. Multiple interpreters were used to share resources across the entire application. For example, a single socket connection is shared between all interpreters. When an applet sends a message to its counterpart running in another user's client, the message is first routed to the application interpreter. It is then sent over the network to the server, which relays it to the application interpreter of the other user's client. From there, it is routed to the interpreter managing the specific applet. The actual mechanism was implemented by having the application interpreter set up an alias in the applet interpreter to intercept communications. Of course, the flexible routing scheme was specified using GroupKit's meta-architecture.

Embedded Windows

Though multiple interpreters give TeamRooms the needed lower level functionality, all those interpreters still had to be able to share the screen somehow. Luckily, Steve Ball had already done most of the work for us in his SurfIt! web browser [1], which features Tcl applets running in their own subinterpreters and having access to Tk features. The basic approach is to carve off a piece of the Tk window hierarchy for the application interpreter, alias that to "." in the applet interpreter, and use aliases to redefine all Tk commands in the applet to run in the application interpreter, with appropriate changes to window names, etc.

We made several changes to this code. First, we allowed the window hierarchy of child interpreters to be rooted at an internal frame widget rather than only at a toplevel, so that interpreters could share the same toplevel window. We removed many of the security limitations enforced by SurfIt!, since at this point we wanted full access to Tk facilities. Finally, we moved several pieces of the code from Tcl into C to improve performance in critical areas.

The frame surrounding each applet is constructed as a standard Tk mega-widget (itself containing 20 small frames for the different pieces of the border), whose inside frame is the root of the applet's window hierarchy. We followed the practice found in the OpenDoc compound document framework [5] that the parent determines the layout of the child, so all resize decisions etc. are managed by the parent.

Issues

The previous section describes some of the techniques that we used in building TeamRooms. Because these may be applied to managing complexity in other Tcl/Tk applications, we now look at some of the obstacles that were faced in applying these techniques in TeamRooms, as well as the solutions we found.

Startup Time

The first difficulty had to do with startup time. Because each interpreter acts like its own application, starting up several different interpreters is like starting up several applications. While a two second initial application startup time may be reasonable, if it takes two seconds for every single applet to be created, the time it takes to enter a room in TeamRooms holding five or ten applets can seem like an eternity.

It took a lot of profiling (mostly using Tcl's "time" command) and subsequent performance tuning to get the time it took to create an applet interpreter and its frame down from about 2.5 seconds to a more reasonable .2 seconds. Some of the changes we made are described below. Note that most are common sense optimizations that were just never an issue before, and that the typical "if its slow, recode it in C" would only address a small number of the problems in this case.

Do the minimum amount of work. Our subinterpreters took a lot of time initializing code they didn't need. For example, we'd originally initialized Tcl-DP even though the applets used the application interpreter's socket facilities (removing this saved about .15 seconds). We used to read one large Tcl configuration file, which included much information used only by other parts of TeamRooms; this was moved into a different file (saving about .2 seconds). Obviously, minimizing work is especially important if the work is done at the slower Tcl level, rather than C.

Avoid autoloading. While autoloading is a very convenient way to load Tcl source code, it is extremely slow! We explicitly sourced all scripts rather than relying on unknown handlers and auto-loading (total saving around .5 seconds, depending on the applet).

Identify special cases. One data structure we use is created and maintained mostly through Tcl code. When creating a new instance, the programmer may specify a number of different options, which requires a lot of slow Tcl code to parse. We identified a frequently-used special case and handled that separately. These types of optimizations saved about .2 seconds.

Use smarter Tcl constructs. We found many improvements here. Our best example is a construct like "lsearch [info commands] foo" rather than "info commands foo" which runs about forty times faster. While Tcl is great to "glue" primitives together, its worth checking the manual pages to see if your favourite Tcl command will do the work for you itself.

Embedded Window Issues

Most of the embedded window issues we faced were performance issues, not surprising given that the code to do the embedding was written in Tcl. In this case, profiling identified some special cases which were rewritten (e.g. there is no need to search through a command using an expensive regular expression search to find window pathnames if the character "." doesn't appear anywhere in the command), or some general routines which were used everywhere where it was worth it to rewrite just those routines in C.

Using mega-widgets was another issue. Both the mega-widget framework we used and the mega-widgets themselves were written in Tcl. Given the overhead of the window embedding code, both creating and using mega-widgets that run in the child interpreter was very slow. Moving them into the parent interpreter (and making them available in the child interpreter with an alias, as is done with the built-in Tk widgets) improved that situation considerably (creating the mega-widget for the applet's object frame took .5 seconds when run in the child interpreter, and just under .1 seconds when run in the parent interpreter).

There were a few other difficulties, such as not being able to access the "-variable" associated with some widgets in a subinterpreter (which was resolved by a set of variable traces). Deciding how images were shared between interpreters is also an issue (we let child interpreters have full access to the parent's images, though this decision may have to be revised if we allow untrusted applets). These will need to be resolved as the "safe Tk" code is redone and integrated into the Tk core.

Interactions Between Interpreters

Interpreters need to communicate with each other to share facilities, such as sockets, information on users, and so on. The multiple interpreter package in Tcl uses a "parent/child" paradigm for interpreters, which we followed closely. Shared facilities were always supplied by the parent (the application interpreter) to the child (the room or applet interpreters), using interpreter aliases. This resulted in the application interpreter program needing extra code, while the code used in the room and applet stayed quite simple, which worked well for our need of simplifying applets.

Though it is possible to use hierarchical interpreters, after some brief experimentation we rejected them. With the applet interpreters being a child of the room interpreter (rather than the application interpreter), and even at one point with applets as children of other applets, things got out of control quickly. Speed was an issue (mainly in the interface code), and responsibilities were spread over many pieces. When possible, a shallow hierarchy of interpreters seems to be more effective.

Another decision we had to make was about menu sharing, so that applets could have access to the main menubar. We chose to add a single menu to the menubar for each applet (available via an alias), and the application interpreter packed and unpacked the menu as the focus changes. An alternative would be to clone the entire menubar for each applet.

Packaging

Because our audience is not only developers but also people who just want to use the system, we needed to package a binary that would not require users to compile their own Tcl, Tk, GroupKit, etc. Existing solutions need some changes to work for applications using multiple interpreters. Typically these systems "compile" Tcl code into arrays of C strings, and load them via Tcl_Eval() at the start of the program. But multiple interpreters are not always created at the program's start, and interpreters may use different files.

The solution we used in TeamRooms was to use an existing package (Joe Touch's "tcl2array" package) to generate C arrays of the Tcl code. We then created a hash table containing pointers to these arrays, indexed by their original Tcl filename. We replaced Tcl's standard "source" command with a new version that first checks if the requested filename is in the table. If so, the code is read from the array, otherwise the file is read from disk.

Cross Platform Issues

While TeamRooms now runs on several flavours of Unix, Macintosh, and Windows, at the time of writing we have little to report in terms of cross platform issues that were difficult to resolve. Most difficulties have to do with missing native functionality (e.g. proper menus and dialog boxes), differences with fonts (which are important if we want identical views of the room across platforms), and so on. Other common cross-platform issues such as layout, naming conventions and so on have not been significant issues with TeamRooms. This is likely because the system relies on a very customized, direct-manipulation interface built using Tk's canvas widget, rather than using a more conventional forms based interface.

Conclusions

This paper has described TeamRooms, a Tcl/Tk groupware application built with our GroupKit toolkit. TeamRooms provides "network places" on the Internet for collaborators, who can interact with generic tools like shared whiteboards. They can also customize their electronic rooms by using applets, which are full groupware applications that run embedded in the room's window, OpenDoc style. TeamRooms is a good illustration of a highly interactive Tcl-based Internet environment.

To accomplish this while still keeping the application's complexity reasonable, TeamRooms relies heavily on a number of techniques. Meta-architectures provide the flexibility to support new run-time architectures. Multiple interpreters allow us to structure the system so that each component acts as its own self-contained application, without requiring extra knowledge about the overall environment. Finally, embedded windows extend the power of multiple interpreters to Tk. Our experiences with these techniques should prove useful as other Tcl/Tk applications begin to use these newer features.

Acknowledgements

Thanks go to early users, in particular Saul Greenberg, Carl Gutwin, Gordon Paynter and Simon Gianoutsos, who braved early versions of the system, and offered many useful suggestions and improvements. Various bits of code have been borrowed from Steve Ball (applet embedding), Stephen Uhler (HTML library), and Shannon Jaeger (megawidget framework). The financial support provided by Intel Corporation and NSERC is gratefully appreciated.

More information about TeamRooms, including software availability, related projects, and publications, can be obtained on the World Wide Web at:

http://www.cpsc.ucalgary.ca/projects/grouplab/teamrooms/

References

1. Ball, S. *SurfIt! A WWW Browser*. In *Proc. of Tcl/Tk Workshop.* 1996..
2. Borenstein, N. EMail with a Mind of its Own: The Safe-Tcl Language for Enabled Mail. In *Proc. of ULPAA*. 1994.
3. Curtis, P. and Nichols, D. MUDs Grow Up: Social Virtual Reality in the Real World. In *Proc. of the Third International Conference on Cyberspace*. May 1993.
4. Johansen, R., Sibbet, D., Benson, S., Martin, A., Mittman, R. & Saffo, P. *Leading Business Teams*. Addison-Wesley. 1991.
5. Orfali, R., Harkey, D. and Edwards, J. *The Essential Distributed Objects Survival Guide.* John Wiley and Sons. 1996.
6. Roseman, M. When is an object not an object? In *Proc. of Tcl/Tk Workshop*. 1995.
7. Roseman, M. and Greenberg, S. Building Real Time Groupware with GroupKit, a Groupware Toolkit. *ACM TOCHI* . March 1996.