TCL/TK AS A BASIS FOR GROUPWARE

Mark Roseman

Department of Computer Science, University of Calgary
Calgary, Alberta, Canada T2N 1N4
Phone: (403) 220-7691 Fax: (403) 284-4707
Email: roseman@cpsc.ucalgary.ca

ABSTRACT

Tcl and Tk may prove an excellent environment for developing groupware applications, yet much remains to be done beyond the relatively simple TCP/IP extensions which exist today. Our groupware research has provided a number of useful models and a groupware toolkit which could be reimplemented on Tcl and Tk to provide a basis for groupware development. This position paper provides some background on groupware and our own research, and then suggests some of the advantages that Tcl and Tk might provide for groupware developers. It also addresses some of the areas of work which will need to be addressed for groupware. These include the design of a shared group object model, some extensions to both Tk's graphics model and event handling model to support overlays, and the issue of cross platform development.

INTRODUCTION

Groupware is a relatively new area of computer science research, focusing on computer systems that help groups of people work together. It is an extremely diverse area; electronic mail, network news, multi-user drawing programs, shared window systems, collaborative writing tools, group decision support systems, shared virtual realities, and sophisticated audio and video media spaces are all forms of groupware. Broadly, groupware can be categorized along two dimensions: time (people working at the same time or different times) and space (people in the same room or distributed over a number of locations).

Groupware researchers are no less diverse. Groupware falls within a larger research area called Computer Supported Cooperative Work (CSCW), which has largely evolved out of the Human Computer Interaction (HCI) community. Groupware researchers include not only computer scientists, but also psychologists, sociologists, anthropologists, business researchers and others interested in the many facets of how computers can support groups of people working together. There's much more to groupware than just getting a communications channel open between two machines!

Though groupware has become "trendy" in the last two years, the products currently on the market have a long way to catch up to the sophistication and novelty evident in much of the current research found at numerous universities and industrial research facilities. It is truly an area with much untapped potential for high-tech advancements.

CALGARY'S GROUPWARE RESEARCH

The University of Calgary has been active in groupware research, under the direction of Dr. Saul Greenberg. The following are some past and present projects.

- A shared terminal emulator called SHARE allows several users to share the same Unix shell. A flexible floor control system moderated access (who can type).
- GROUPSKETCH is a simple multi-user paint program which spawned several more sophisticated variants.
- A multi-user structured drawing program called GROUPDRAW features an underlying group object framework to handle concurrency and ownership.
- LIVEWARE is a benign virus that automatically exchanges data between users during casual contacts.
- One project is investigating the use of facilitation in groupware and developing generic facilitation tools.
- One project assists in the problem of "making contact" with potential collaborators, a difficult problem that must be overcome for successful collaboration.
- GROUPKIT is a groupware toolkit (described shortly).
- Personalizable groupware helps in the problem of user acceptance (described shortly).

GROUPKIT

GROUPKIT is a toolkit for building groupware that generalizes much of our earlier work. For example, "open protocols" (described below) are used to build a flexible registration system, the group drawing framework from GROUPDRAW was incorporated into the toolkit, and other components from the drawing programs such as gesturing — which is an important feature for real-time groupware — are provided as toolkit primitives.

GROUPKIT was designed and built on top of Stanford's INTERVIEWS C++ toolkit. While this assisted with some of the work (particularly by providing a nice graphics model for creating new widgets), a number of problems have resulted which makes the current system less maintainable and usable than was hoped. Structural changes in the widget model resulted in an incomplete set of widgets and immense difficulty in customizing existing widgets (which was an earlier strength of the system). Documentation has been non-existant leaving beginners with no assistance. Though GROUPKIT is easy to use, its reliance on INTERVIEWS has made it inaccessible to most users. Overall, INTERVIEWS has not evolved as expected, and we are looking at Tcl/Tk as an alternate platform for our work.

Personalizable Groupware

One of the problems with groupware is that the entire group must be willing and able to use the same system, otherwise it will be a failure — for the *entire* group, not just the dissenting users. Contrast this with single user software where a single user's acceptance of a package typically affects only the user.

As a result, there is great interest in making *personalizable* groupware, allowing different users to see the system differently according to their own needs and preferences. We've developed a technique called *open protocols* which allows new modules to be inserted into a system long after the original system is designed, allowing for new and unexpected uses of the system. Being able to easily design these modules is an important issue which will be addressed later.

ADVANTAGES TO TCL/TK FOR GROUPWARE

In many ways, Tcl and Tk show excellent promise as a platform for groupware development. As a general programming tool, it has many advantages: its rich widget set (including a canvas and sophisticated text editor), excellent documentation (online as well as the upcoming book) and importantly a growing community of users who almost daily add to the collection of tools available. More specific to groupware, a number of areas stand out, some of which are mentioned below.

Quick to Design New Interfaces

Development of interfaces with Tk is quick and painless compared to alternatives such as Motif and Xt. This is important in groupware for two main reasons.

First, because groups' needs vary, groupware systems are notoriously difficult to build correctly. An iterative prototyping approach, suggested for single-user software, is fundamental to building groupware. Groupware interfaces must be designed quickly, and redesigned often.

Second, it may be desirable for end users (or at least sophisticated representatives of the end user community) to design new interfaces. Our personalizable groupware work suggests application-specific building blocks may be provided by the developer (e.g. as Tcl scripts) so users can build their own modules, but still work must be done by end users. Tcl and Tk may make this more possible.

Tcl Commands *are* Communication Protocol

A basic part of groupware is sending messages between processes. A constant issue is the format of such messages. Typically one converts between a native data type (i.e. C integers, floats and lists of objects) into strings and then converts back again on the other side.

Yet with Tcl, no conversion is necessary, everything already is a string. As was nicely shown with the RPC extensions in Tcl-DP, commands can be sent verbatim over sockets and executed in the receiving end's interpreter. Tcl commands *are* the communication protocol!

Flexible Bindings

Groupware often requires new widgets, or at least widgets that behave differently than standard single-user widgets. The typical solution is to design entirely new widgets, or in object-oriented systems, subclass the original widgets.

Tcl and Tk present another solution: change the bindings to redefine the behavior of existing objects. Since every button click, every mouse move, and every key press can be easily trapped, arbitrary operations can be invoked to change the behavior of the widget to anything desired. This also provides a powerful mechanism to experiment with new behaviors interactively, and to do so with a minimum of code. This flexibility of behavior is of the utmost importance in developing new groupware widgets.

Easy to Separate Interface from Application

Single-user interface developers often argue that the interface should be separated from the underlying application. In groupware this becomes more desirable, as dealing with interfaces at the same time as the very complex underlying behavior of group objects is exceedingly difficult. Further — again following the personalizable approach — if different users are working simultaneously with different interfaces to the same objects, some separation of the interface is critical.

There are a number of approaches for separating interfaces from underlying objects, ranging from hard-coded special cases to constraint management systems. Tcl's "trace" function presents a very realistic solution to this problem, so that interfaces can find out about changes to their underlying object automatically.

FUTURE ISSUES

If a serious effort at groupware development were to made based on Tcl and Tk, a number of areas would need to be addressed. While some are reasonably application specific and do not interfere with current directions of Tcl and Tk, others present issues which may have a direct effect on the core systems. Some of these effects could of course benefit other application domains immensely. The following represents some of the issues.

Group Object Management

An obvious issue in groupware is that of distributed or "group" objects, objects where copies exist on several different machines.

Tcl-DP has made a good initial start at this, providing replicated objects whose fields ("slots") can be changed by any user. Changes are serialized (to keep consistent states) by sending them to the slot's "owner", who forwards changes to other networked copies of the object.

Though an elegant scheme, it does not yet deal well with issues such as latecomers, persistence, and recovery if an object's owner crashes. More important for groupware applications, it works only at the level of individual slots, and ignores application level semantics, which often includes locking of slots.

To give some examples of application semantics, the group object model we define has three access levels to an object: private (only an object's owner sees it and can modify it), public (everyone sees the object but only the owner can modify it), and shareable (everyone can see and modify it). This control can be important in some applications. To illustrate the need for locking, imagine a group object representing a line in a drawing program. Two users should not be able to modify the same endpoint of a line (the first person should obtain a lock, drag the endpoint and release the lock). Yet this locking must be finegrained, so that another user might be able to drag the other endpoint of the line.

All of this could be accomplished using a strategy similar to that employed by Tcl-DP, again using only Tcl scripts. Our group object model could be directly implemented in this manner. The use of Tcl scripts could also provide the flexibility to experiment with different object models transparently.

Bindings and Graphics Model for Overlays

In GROUPKIT, we relied extensively on *overlays* to provide several group specific toolkit components. Gesturing, important in groupware programs, is often implemented with *multiple cursors*, bitmap cursors which appear on other users' windows to allow pointing to specific items. These bitmaps follow the mouse cursor and are displayed as an overlay on top of any other running application. As another example, freehand annotation (e.g. like proofreading marks) could be overlayed on top of other applications.

Overlays have two requirements: drawing and input handling. For drawing, typically objects underneath an overlay will draw themselves, and then the objects in the overlay draw themselves, resulting in the first appearing, perhaps partially obscured, below the second. It is important that this be very efficient, as both the underlying application's graphics and the overlay's graphics may be changing rapidly. It is unclear whether Tk currently has the ability to support overlays in this fashion. The canvas widget, in which arbitrary widgets can be embedded, does provide a starting point for a general overlay mechanism.

Input events must also be handled carefully. The strength of the overlay approach is that the underlying application should be completely unaware that the overlay is present. This requires all input events to be received by *both* the overlay and the underlying application. For example, a mouse move event must be seen by a gesturing overlay (in order to move the bitmap cursor) and also the underlying application which may have use for the event. This may be quite difficult to implement with the current event binding scheme in Tk.

Cross Platform Applications

People work in heterogeneous environments. If a groupware system is designed for such an environment, it must work on all the target platforms, or risk failure when some users refuse to use it. While Tcl is reasonably portable, Tk is not. This is an issue confronting many Tcl developers, not just for groupware.

There are two approaches to this problem in terms of groupware: make Tk run on other systems or not. The issues here are many and probably well-known, not the least of which are all the X specific concepts included in the language itself (subwindows, X font names, event types, window manager calls, etc.). Though Xlib emulation exists for non-Unix machines, this may not be a desirable alternative, particularly since the result will not be in the "native" look and feel of the target machine.

The other approach is not to make Tk portable, but just to build different interfaces on different machines. Build a different toolkit on every machine, probably Tcl "wrappers" around a native toolkit. For example, we did a quick bit of work to provide a Tk style of programming on the Macintosh, yet using the Mac toolkit for buttons, labels, etc. This approach means reimplementing the entire interface on every new platform. However, if the "separate application from interface" approach advocated above is followed, it will reduce this effort substantially.

PROLOGUE

We have recently implemented most of our groupware toolkit in Tcl and Tk using the Tcl-DP extensions. Prototype applications to date include: a brainstorming tool, a voting tool, a post-it note application, a shared whiteboard, simple shared structured graphics and group hypertext programs, a primitive group text editor, a network telephone interface, and several registration systems including a start on our making contact system.

The resulting Tcl/Tk toolkit is not only significantly shorter than the INTERVIEWS version, but also significantly easier to build applications with. While the canvas widget has sufficed to provide some overlay capabilities, it is still not sufficient in the general case. Our work in the near future will focus on larger and more robust applications, further developing our group object model, and integrating distributed information sources using lightweight Tcl servers.

SELECTED PAPERS

Greenberg, S. (1991) "Personalizable groupware: Accommodating individual roles and group differences." In *Proc. of the 2nd European Conference on Computer Supported Cooperative Work (EC-CSCW '91).*

Greenberg, S., Roseman, M., Webster, D. and Bohnet, R. (1992) "Human and Technical Factors of Distributed Group Drawing Tools." *Interacting with Computers*, **4**(3), pp. 364-392.

Roseman, M. and Greenberg, S. (1992) "GroupKit: A Groupware Toolkit for Building Real-Time Conferencing Applications." In *Proc. of the Conference on Computer-Supported Cooperative Work (CSCW '92).*

Roseman, M., and Greenberg, S. (1993). Building Flexible Groupware Through Open Protocols. *In submission.*