GROUPKIT INFRASTRUCTURE MANUAL

A Guide to its Architecture, Interprocess Communications, and Programs

Mark Roseman, Salaam Yitbarek, and Saul Greenberg

Department of Computer Science, University of Calgary
Calgary, Alberta, Canada T2N 1N4
Phone: +1 403 220-6087
E-mail: saul@cpsc.ucalgary.ca

(Note: This manual is very slightly out of date)

April, 1993

INTRODUCTION

GroupKit Overview

GroupKit is a toolkit for developing real-time groupware applications. Based on Berkeley's public domain Tcl/Tk language, it provides the basis for building groupware applications. Tcl is an interpreted shell-type programming language, and Tk is an interface toolkit for the X11 window system. GroupKit also relies on Tcl-DP, a Tcl front-end to standard Unix sockets, for its communication needs. GroupKit programs therefore run on Unix machines.

About this Manual

This manual discusses GroupKit's run-time architecture, its interprocess communication, and the flow of control through the program modules that form the groupware toolkit. It is more of a guide to understanding the GroupKit infrastructure, rather than a guide for learning how to use GroupKit.

Those who want only to make GroupKit applications and not bother about the internals need only read the General GroupKit Organization, Application Level Conference Communication, and GroupKit Widgets sections. Those wishing to change parts of the GroupKit package, or make new GroupKit modules or new registrar clients should read the whole document, concentrating on the "Registration System Flow of Control" section.

Previous Knowledge

Users of this manual are assumed to be capable in Tcl (including the Tcl-DP extensions) and Tk. A basic understanding of groupware systems and building GroupKit applications is also required. It is recommended that the reader finish the GroupKit Tutorial before using this manual.

Remainder of this Document

This reference manual consists of the sections listed below:

- General GroupKit organization
- The Registrar
- Generic Registrar Client and Registrar/Registrar Client Protocol
- Registrar Clients
- Conferences Process Communication
- Registration System Flow of Control
- Application Level Conference Communication
- GroupKit Widgets and Errors Module

For Further Information

The "README" file in the GroupKit distribution provides details for installing GroupKit on your system.

The "GroupKit Tutorial" contains a step-by-step description on running existing GroupKit applications and how to build new GroupKit applications.

The paper "GroupKit: A Groupware Toolkit for Building Real-Time Conferencing Applications" (ACM Proc. CSCW '92, Toronto, Ontario) describes an earlier version of GroupKit designed using C++ and InterViews, a C++ interface toolkit. It may help illustrate some of the important concepts in GroupKit.

"Building Flexible Groupware Through Open Protocols" (ACM Proc COOCS '93), describes the *open protocols* method of groupware design upon which GroupKit relies.

The paper "Tcl/Tk as a Basis for Groupware" (Proc. Tcl '93, Berkeley, CA), explains the benefits of Tcl/Tk for groupware development.

An Introduction to Tcl/Tk (to be published by Addison-Wesley approx. Feb. '94), by John K. Ousterhout is a useful guide to Tcl/Tk.

Documentation on the Tcl-DP extension can be found in its on-line manual pages which should have been installed with Tcl-DP.

GROUPKIT RUN-TIME ARCHITECTURE

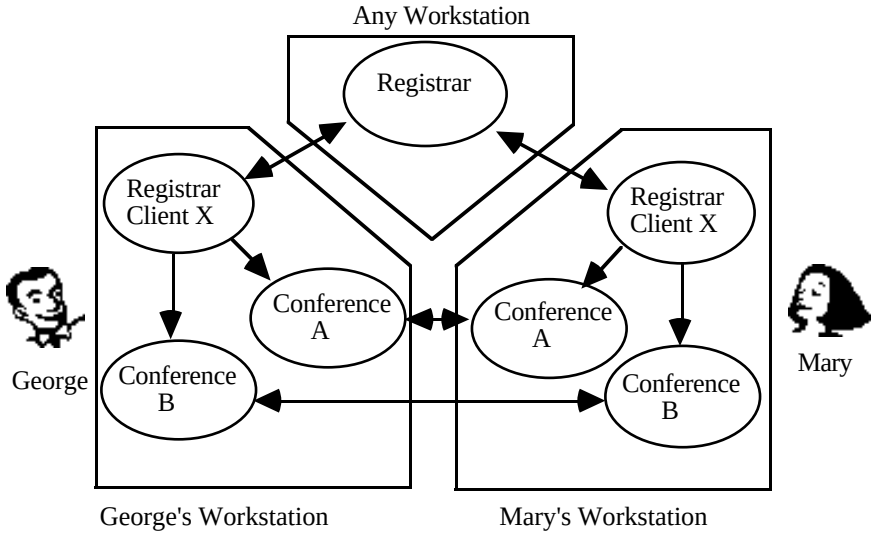
Concepts

Under the heading of group activities or group work, there are plenty of different arrangements, styles and methods to fit the goals that different groups have. Take the case of meetings: there are informal, formal, brainstorming, decision-making, private, open, etc., meetings. For groupware to accommodate these various types of group-work, it must be flexible. This is the goal of *open protocols*.[1]

GroupKit relies on the open protocols method of groupware design. The gist of open protocols is that the server for the registration system (the *registrar* in GroupKit) is a controlled object, and the clients (*registrar clients* in GroupKit) are the controlling objects. The registrar simply maintains state information and receives instructions from the registrar clients through a pre-defined messaging protocol; the clients set system access policy.

The third type of GroupKit process (after the registrar and registrar client) is the *conference*. Say two users (George and Mary) are sharing a multi-user sketchpad application developed using GroupKit. The multi-user sketchpad is called a conference; one of the users started it, and the other "joined it". In GroupKit, the term conference also applies to a *conference process*—a single instance of a running application. In the above example, George's copy of the sketchpad is, by itself, called a conference; different from Mary's copy.

The GroupKit model is based on a number of processes arranged in a *replicated* architecture as shown below (the arrows denote paths of communication). Each user on their workstation has their own registrar client and conference processes running; they usually see these as windows—one for the client, and one for the appication. The registrar client processes communicate with each other through the registrar, which usually exists only on one machine in the network. The conference processes that are part of one overall conference (the two conference A's below, for example), communicate directly with each other.



Using open protocols, conference participants, through their registrar clients, control who can initiate certain conferences, who can join them, how a conference can be joined, who is to know about the existence of certain conferences and other such issues. The registrar keeps state information that includes a list of conferences running and a list of users (actually, the users' registrar clients).

Modules

[1]See the "open protocols" paper above for more details.

The **registrar** (defined in reg.tcl) is the server for the groupware application. As open protocols specifies, it is a controlled object that only keeps lists of conferences and users. It communicates directly with the registrar clients by way of rc.tcl.

The **registrar client** module (openrc.tcl, nukerrc.tcl, and cmdrc.tcl are examples) creates, joins and deletes conferences. It usually provides an interface to the user as well from which they can undertake the above tasks. Registrar clients communicate with the registrar and with the conferences that they spawn.

The **protocol** or **generic register client** module (rc.tcl) provides the basic facilities for communication between the registrar and the clients. It contains the basic set of tools required by all registrar clients.

The **conference** module (conf.tcl) provides tools for use by groupware applications. These include procedures for conference to conference and conference to registrar client communication. Conferences only communicate with each other and registrar clients.

The **widgets** module (gkwidgets.tcl) provides widgets that can be used in groupware applications.

The **errors** module (errors.tcl) contains error-handling and exiting routines.

GroupKit Initialization

The .tclgkrc file is the GroupKit initialization file. It resides in the user's home directory and contains some information on the user and the host machine of the registrar. It also includes a list of the types of conferences that the user wishes to have access to. Below is a sample .tclgkrc file.

```
set myusername "Mark Roseman"
set internetdomain "cpsc.ucalgary.ca"
set reghostname janu
set myColour sienna
set gk_program(SimpleSketch) "exec gkwish -f $gk_library/drawing/sketch.tcl"
set gk_program(TestSketch) "exec gkwish -f $gk_library/groupsketch/groupsketch.tcl"
set gk_program(Brainstorming\ tool) "exec gkwish -f $gk_library/brainstorm/brainstorm.tcl"
set gk_program(Drawing\ Program) "exec gkwish -f $gk_library/drawing/draw.tcl"
```

For GroupKit installation information, see the "README" file in the GroupKit distribution.

GroupKit Naming Convention

All major global variables and procedures in GroupKit begin with the prefixes "_gk_" or "gk_". Those that begin with "gk_" are those meant to be used by the application developer. Variables and procedures beginning with "_gk_" are usually used only within GroupKit and in the design of registrar clients.

THE REGISTRAR

The registrar is the server for the registration system. It uniquely identifies every conference and user, and keeps lists of conferences and users in conferences. It also has functions that allow the registrar clients to manipulate those lists.

reg.tcl is the registrar module in GroupKit. When reg.tcl is run, it checks the .tclgkrc file for the host machine, checks the command line for a port number, and creates a server at that host and port. It initializes its list of conferences, "_gk_regConfsList", and "myID", the identifier for conferences and users.

Key variables in reg.tcl are:

_gk_regConfsList: A list of keyed lists, where each keyed list represents a conference. The conference keyed lists contain the following keys:

required keys:

confnum—a unique integer id for a conference. confnum is assigned by registrar.

normal keys:

confname—a "user-defined name" for a specific conference, e.g. Mark's Meeting)

conftype—an identifier for all conferences of a certain type, e.g. Brainstorming Tool; found from .tclgkrc)

originator—the conference creator as identified by the host and port of that registrar client, e.g. janu9090. This will change in the future.

optional keys:

Anything the registrar clients want to put in through the **_gk_doNewConf** procedure.

myID: A counter used to assign unique id's to conferences and users.

_gk_regUsersList: This is an array of lists of keyed lists of users per conference. Therefore, _gk_regUsersList(confnum) contains a list of keyed lists. These keyed lists represent a user in conference number confnum. The keys in the users' keyed lists are the following:

required keys:

usernum—a unique integer id for a user. usernum is assigned by the registrar.

normal keys:

userid—the login id of the user, e.g. roseman.

host—the internet hostname of that registrar client, NOT of the user's conference process. E.g. janu.cpsc.ucalgary.ca.

port—the port number on that host (again, the registrar client's, not the conference process').

username—the full name of the user. This is myusername in the .tclgkrc file. If myusername is not specified, the userid is used. E.g. Mark Roseman or roseman.

optional keys:

confnum—the unique conference number. This is used for an implementation hack in rc.tcl and will not likely exist in the future.

Anything else that the registrar clients may put in through their **_gk_doJoin** or **gk_foundNewConf** procedures. If the registrar clients want additional keys in the user keyed list, they should have their **_gk_doJoin** and **gk_foundNewConf** functions call the procedure **_gk_callJoinConfWithKeys** instead of **_gk_callJoinConference** in rc.tcl.

connections: A list of all the file descriptors of our attached registrar clients, connections is used by the registrar to "broadcast" to all the registrar clients. connections is maintained by a patch to Tcl-Dp; see rpc.tcl for details.

Key procedures in reg.tcl, the routines that call them, important parameters, and some descriptions are listed below:

_gk_newConference

Parameters: **conf**—a keyed list representing the conference to be created.

Callers: _gk_callNewConference in rc.tcl.

Function: Adds the key "confnum" to conf and appends conf to _gk_regConfsList.

_gk_deleteConference

Parameters: **id**—the conference number of the conference to be deleted.

Caller: **_gk_callDeleteConference** in rc.tcl.

Function: Deletes conference with conference number confnum from _gk_regConfsList.

_gk_dispConferences

Parameters: none.

Caller: **_gk_pollConferences** in rc.tcl.

Function: Broadcasts _gk_regConfsList to all the registrar clients connected to the registrar. The registrar clients should take the list as being the "new conference state". **_gk_dispConferences** must be called explicitly; it does not automatically broadcast when there is a change to _gk_regConfsList.

_gk_addUser

Parameters: **user**—a keyed list with information on the user.

Caller: **_gk_callJoinConference** in rc.tcl.

Function: Appends usernum to the user keyed list and appends the keyed list to _gk_regUsersList.

_gk_deleteUser

Parameters: **conf**—the conference number of the conference in which the user is.
user—the user's user number.

Caller: **_gk_callLeaveConference** in rc.tcl.

Function: Deletes a user from the _gk_regUsersList.

_gk_dispUsers

Parameters: **conf**—a conference number that serves as an index to the _gk_regUsersList.

Caller: **_gk_pollUsers** in rc.tcl.

Function: Broadcasts _gk_regUsersList(conf) to all the registrar clients connected to the registrar. The registrar clients should take the list as being the "new user state".

GENERIC REGISTRAR CLIENT

rc.tcl provides services required by registrar clients in order for them to communicate with the registrar. It contains generic routines that are used by any registrar client.[2] A registrar client that uses rc.tcl must include the following routines (see the "Registrar Clients" section for details):

gk_foundNewConf: Called when a new conference is created.

gk_foundDeletedConf: Called when a conference is deleted.

gk_foundNewUser: Called when a new user is found for a conference.

gk_foundDeletedUser: Called when a user is deleted from a conference.

The above routines are called by rc.tcl's **_gk_updateList**, but are passed into **_gk_updateList** by its "front-ends", **_gk_confList** and **_gk_userList**.

Initialization in rc.tcl includes setting up the client on the host and port specified on the command line, and setting the registrar client's conference list (_gk_rcConfsList) and user list (_gk_rcUsersList) to nil. A means of connection to the registrar is also set up.

rc.tcl contains the following important global variables:

_gk_rcConfsList: The same structure as _gk_regConfsList.

_gk_rcUsersList: The same structure as _gk_regUsersList.

host: The host on which the registrar client is running.

myport: The socket listener maintained by the registrar client.

registrar: The file descriptor of the registrar to which the registrar client is connected.

pending: An array of lists of Tcl commands, where each element of the array contains a list of commands which must be sent to a particular conference (e.g. pending(1) is the list of commands to be sent to conference 1). This is used as a "buffer" so that registrar clients can send commands to conferences they spawn even before the conference has been created. When the conference connects up, all pending commands are sent.

conferences: A list of keyed lists, where each keyed list represents a conference. This keeps track of the conferences that have connected up to the registrar client but does not hold conferences that were spawned by the registrar client but who have not connected back yet. Usually, registrar client programs maintain a data structure that includes such conferences (e.g. _gk_confsJoined in openrc.tcl).

The following three procedures form the first part of rc.tcl. They deal with exchange of information about conferences and users between the registrar and the registrar clients. Again, important parameters, calling routines, and the function of the procedure is listed.

_gk_updateList

Parameters: **list**—_gk_regConfsList or _gk_regUsersList.

oldlist— _gk_rcConfsList or _gk_rcUsersList.

newproc—procedure to be called when an item exists in list that does not exist in oldlist.

oldproc—procedure to be called when an item exists in oldlist that does not exist in list.

Callers: **_gk_confList** and **_gk_userList**. Both are in rc.tcl.

---

[2]Some simple registrar clients that do not use code in rc.tcl can be created. Examples are cmdrc.tcl and nukerrc.tcl, described further in the "Registrar Clients" section.

Function: Compares oldlist with list and calculates list-oldlist (new items) and oldlist-list (deleted items). It then calls newproc with each of the new items and calls oldproc with each of the deleted items.

_gk_confList

Parameter: **_gk_regConfsList**.

Caller: **_gk_dispConference** in reg.tcl.

Function: This routine is a front-end to **_gk_updateList**. It just calls **_gk_updateList** with _gk_regConfsList, _gk_rcConfsList, and two procedures provided by the registrar client module—one for handling new conferences (newproc) and another for handling deleted conferences (oldproc). Then **_gk_updateList** does the following:

"new conferences" = _gk_regConfsList-_gk_rcConfsList

"old conferences" = _gk_rcConfsList-_gk_regConfsList

_gk_updateList then calls newproc for each item in "new conferences" and calls oldproc for each item in "old conferences". newproc and oldproc implement the registrar client's policy of handling new and deleted conferences. For the registrar client module openrc.tcl, newproc is gk_foundNewConf and oldproc is gk_foundDeletedConf.

_gk_userList

Parameters: **conf**—conference number of the conference for which users are being polled.

_gk_regUsersList.

Caller: **_gk_dispUsers** in reg.tcl

Function: This is also a front-end to **_gk_updateList**—similar to **_gk_confList**, but for users. It calls **_gk_updateList** with _gk_regUsersList, _gk_rcUsersList, and two procedures provided by the registrar client module—one for handling new users (newproc) and another for handling deleted users (oldproc). Then **_gk_updateList** does the following:

"new users" = _gk_regUsersList-_gk_rcUsersList

"old users" = _gk_rcUsersList-_gk_regUsersList

_gk_updateList then calls newproc for each item in "new users" and calls oldproc for each item in "old users". newproc and oldproc implement the registrar client's policy of handling new and deleted users. For the registrar client module openrc.tcl, newproc is gk_foundNewUser and oldproc is gk_foundDeletedUser.

The next part of rc.tcl deals with interaction with conferences to which the local client is attached or attaching itself. The variable "conferences" keeps a list of the conferences to which the local client is attached. The procedures in this part are:

_gk_createConference

Parameters: **conf**—a keyed list with information about the conference process to be created.

usernum—the user number of the user who spawned conference conf.

Caller: **gk_foundNewUser** in openrc.tcl or a similar procedure in any registrar client. Note that a given registrar client may choose NOT to call **_gk_createConference**, depending on the type of policy it wishes to implement.

Function: Creates a conference process. It first fetches a command from gk_program, the array defined in the .tclgkrc file. It then executes this command, spawning a new conference process. Now, the registrar client has to somehow tell the conference process about itself so that the conference can connect back. This is now done by attaching command line arguments to the command from gk_program. _gk_checkConnection is called after ten seconds to check whether the new process has connected back. If it has not, it is removed from the registration system.

_gk_toConf

Parameters: **confnum**—conference number of conference to send messages to.

msg—monosodium glutamate! Actually, the message to be sent.

Caller: **_gk_joinTo** in rc.tcl.

Function: Sends messages, in the form of dp_RDO's, to conferences. The only messages it sends now are to its own conference processes, telling them to call their **_gk_connectTo** routines to connect to remote conference processes. If the conference to which msg is to be sent does not yet exist, then msg is put on the **pending(confnum)** list of messages to be sent to conference confnum.

_gk_remoteInfo

Parameter: **whom**—a keyed list with information about a new conference process.

Caller: **gk_initConf** in conf.tcl

Function: To connect the registrar client with the new conference process. **_gk_remoteInfo** puts whom into the registrar client's conferences array, signaling that the conference has connected back up. It then sends the conference all the messages that it had pending.

_gk_joinTo

Parameters: **user**—user of the conference process to whom the local registrar client's conference process must join.

conf—the local user; the user of the local registrar client's conference process.

Caller: **gk_foundNewUser** in openrc.tcl, or an equivalent routine in another registrar client.

Function: Initiates a series of steps to join the local conference "conf" to the remote conference used by "user". **_gk_joinTo** first fetches the "address" (actually the conference keyed list) of the remote conference from the remote conference's registrar client. Then it sends a message (via the procedure **_gk_toConf**) to it's local conference process telling it to connect to (using its **_gk_connectTo** routine) the remote conference.

_gk_userLeft

Parameters: **conf**—the keyed list of a conference process that has died (by not connecting back to its registrar client or by losing an existing connection).

usernum—a user of the dead conference.

Callers: **_gk_checkConnection** and **socket_closed**, both in rc.tcl.

Function: Tells the registrar that user usernum has left the conference. If user usernum was the only remaining participant in the conference, the registrar is told to delete the conference from its list.

socket_closed

Parameters: **filedesc**—the file descriptor of a closed socket.

Caller: **tkerror** in errors.tcl.

Function: **socket_closed** is called when a conference's socket connection is closed. It deletes that conference from the registrar client's "conferences" list and calls **_gk_userLeft** to remove it from the registrar.

The last part of rc.tcl consists of procedures used by the registrar client to send messages to the registrar.

_gk_callJoinConference

Parameter: **confnum**—the conference number of the conference that the local user wants to join.

Callers: **_gk_doJoin** and **gk_foundNewConf** in openrc.tcl, or like routines in other registrar clients.

Function: **_gk_callJoinConference** is called when i) a user wishes to join a conference that he did not create (**_gk_doJoin**) or ii) the creator of a conference process wants to join the conference (**gk_foundNewConf**). It creates a user keyed list and sends it to the registrar (by calling **_gk_addUser**), telling the registrar to add that user to _gk_regUsersList.

_gk_callLeaveConference

Parameters: **conf**—the conference being left by user.

user—the user number of the user leaving conference "conf".

Caller: **_gk_userLeft** in rc.tcl.

Function: **_gk_callLeaveConference** is called when a user exits a conference process. It tells the registrar, by calling **_gk_deleteUser**, to remove the user from the _gk_regUsersList.

_gk_callNewConference

Parameters: **conf**—a keyed list representing the new conference.

Caller: **_gk_doNewConf** in openrc.tcl, or a similar procedure in another registrar client.

Function: **_gk_callNewConference** is called when a new conference (not just a new conference process) is started by the user. It tells the registrar, by calling **_gk_newConference**, to put the new conference in _gk_regConfsList.

_gk_callDeleteConference

Parameters: **confnum**—conference number of conference to be deleted.

Callers: **_gk_userLeft** in rc.tcl, and **gk_doDelConf** in nukerrc.tcl.

Function: Tells the registrar to delete conference confnum from _gk_regConfsList. This removes the conference from the whole registration system.

_gk_pollConferences

Parameters: none.

Callers: global **openrc**, **_gk_doNewConf** in openrc.tcl, and several others.

Function: Tells the registrar, by calling **_gk_dispConference**, to broadcast _gk_regConfsList to all the registrar clients. This must be done everytime a change occurs in _gk_regConfsList. Therefore, **_gk_pollConferences** is usually called after the registrar has been told to add or delete a conference. It is also called by newly starting registrar clients so that they can update their _gk_rcConfsList.

_gk_pollUsers

Parameters: **confnum**—the conference number of the conference whose users must be polled.

Callers: **_gk_doJoin**, **gk_foundNewConf** in openrc.tcl and several others.

Function: **_gk_pollUsers** is called when a user has been added or deleted from a conference in the registrar's _gk_regUsersList. It calls **_gk_dispUsers** in the registrar to tell it to broadcast the list of users for conference confnum to all the registrar clients.

REGISTRAR CLIENTS

The first sample registrar client provided in the GroupKit package is openrc.tcl. It is a straightforward registrar client—users can create, join, and delete any conference. Openrc.tcl uses the generic registrar client routines provided in rc.tcl. Thus it contains the variables _gk_rcConfsList (list of conferences as kept by the registrar client) and _gk_rcUsersList (list of users) that are initialized in rc.tcl.

Global variables in openrc.tcl are:

_gk_confsJoined: A list of conferences joined by that registrar client.

All the globals in rc.tcl.

The first part of openrc.tcl contains the routines that implement registrar client policy on who can start, join, and delete conferences and how these tasks are to be handled. As mentioned in the section on rc.tcl, the following routines are essential for a registrar client using rc.tcl. They are all called by **_gk_updateList** (in rc.tcl).

gk_foundNewConf

Parameters: **conf**—a keyed list representing a conference new to this registrar client.

Function: This procedure must respond to a conference (conf) being in _gk_regConfsList but not in its _gk_rcConfsList. openrc's response is to put conf in its _gk_rcConfsList, and initialize its _gk_rcUsersList(conf). This means that the registrar client now has access to the new conference, and the user may join the conference. If the new conference's originator is the local registrar client, then the local registrar client joins the new conference.

gk_foundDeletedConf

Parameters: **conf**—the conference number of the deleted conference.

Function: This is called when the registrar client notices that the registrar has deleted a conference from _gk_regConfsList. openrc simply deletes the conference from its _gk_rcConfsList.

gk_foundNewUser

Parameters: **user**—a keyed list representing the new user.

Function: In openrc, when a new user in a conference is found, it is added to the _gk_rcUsersList. If the new user is the local registrar client, it spawns a conference process by calling **_gk_createConference**. It then joins to the other participants in the conference. If the new user is not local, there is no need to create a conference process.

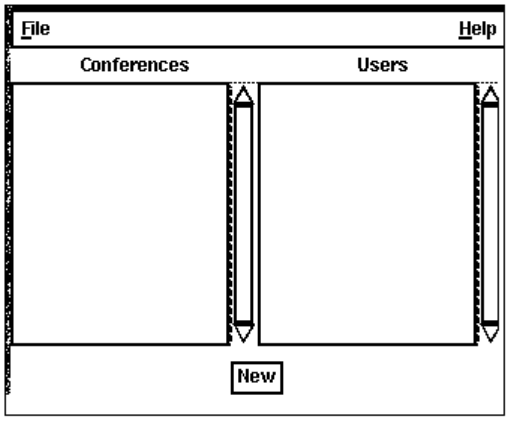
gk_foundDeletedUser

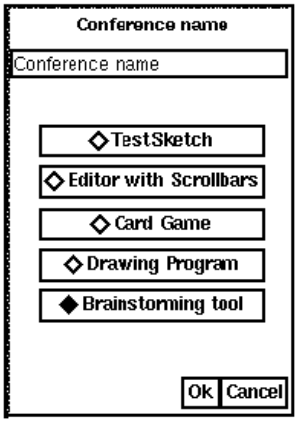
Parameters: **conf**—the conference number of the conference from which the user exited.

user—the user who left a conference.

Function: Again, openrc remains straightforward. When a user leaves a conference "confnum", the user is removed from _gk_rcUsersList(confnum). The registrar client then calls **_gk_removeUser** in the conference process to tell it to remove the user from its data structures. If the user is the local user, the conference process then removes itself.

The next part of openrc.tcl is concerned with making the interface; this is straight Tk code. The main window (below left) consists of two listboxes and a “New” button.





One listbox contains the conferences that are running. Clicking once upon a conference brings up the list of participants in that conference in the other listbox. Double-clicking on the conference signals a request to join the conference. The “New” button is clicked upon if the user wants to create a new conference. This brings up the “Conferences Dialogue Box” (above right), a menu listing all the different types of conferences available. Choosing a conference and clicking “OK” creates a new conference.

The important procedures in openrc.tcl’s interface section are listed below:

_gk_doJoin

Parameters: **whichconf**—the conference number of the conference to be joined.

Caller: Called when the user double-clicks on a conference in the conference listbox, signaling that he wants to join that conference.

Function: Creates a conf keyed list, puts the conference number in the global “_gk_confsJoined” variable, and passes confs on to **_gk_callJoinConference** in rc.tcl.

_gk_updateConfList

Parameters: none.

Caller: Called on a trace to _gk_rcConfsList.

Function: It updates the list of conferences in the openrc main window.

_gk_updateUserList

Parameters: none.

Caller: Called on a trace to _gk_rcUsersList.

Function: It updates the users listbox in the main window.

_gk_newConfDlg

Parameters: none.

Caller: Called when the user clicks on the “New” button in the main openrc window.

Function: Pops up the “Conferences Dialogue Box” that displays a list of conference applications for the user to choose from.

_gk_doNewConf

Parameters: none.

Caller: Called when the user picks a new conference from the “Conferences Dialogue Box”.

Function: It initializes conf as the keyed list for the conference being created and then calls **_gk_callNewConference** and **_gk_pollConferences** in rc.tcl.

nukerrc.tcl is the second sample registrar client provided by GroupKit. It is almost the same as openrc, except that double-clicking on a conference “nukes” (deletes) the conference from the registrar instead of joining you to the registrar. This is done through the gk_doDelConf procedure, a replacement of _gk_doJoin in openrc. See the program listing for more.

cmdrc.tcl is the third sample registrar client provided by GroupKit. It is a command line registrar client.. Entering “cmdrc conference” on the command line joins the user to the conference or starts up a new one. The client remains running until the conference is removed. Therefore, the user can only join or create one conference at a time and is not told who the rest of the participants are.

cmdrc.tcl is an example of a registrar client that uses only some parts of rc.tcl. It ignores some of the functions provided by rc.tcl, and substitutes some of its own functions for others. See the program listing for cmdrc.tcl for details.

CONFERENCE PROCESS COMMUNICATION

There are two important keyed lists maintained in conf.tcl. The first, "_gk_remoteUsers", holds the following information about all the remote users:

usernum: The unique user number for that user.

userid: The login id. of the user.

username: If available, the user's proper name, otherwise the userid.

filedesc: The file descriptor used by Tcl-DP to send messages to users.

The second keyed list, "_gk_localUser", includes the above data and the following about the local registrar client:

confname: The name of the conference (as entered by the user in the conferences dialogue box.

conftype: The type of conference (the application name).

originator: The host and port of the registrar client who began the conference.

confnum: The unique id. of the conference.

reghost, regport: The host and port of the conference's registrar client.

The other major global variables in conf.tcl are "gk_newUser" and "gk_deletedUser". gk_newUser is set whenever a conference has a new user. For example, if registrar client A spawns a conference and registrar client B joins that conference, gk_newUser will be set in registrar client A's conference process to the user of registrar client B. The conference process spawned by registrar client B will also have its gk_newUser variable set, NOT to the local user, but to the user of registrar client A. gk_deletedUser is set when a user leaves a conference.

The first part of conf.tcl handles internal connections amongst conferences and conference initialization. It contains the following procedures:

gk_initConf

Parameters: The host number, port number and other such data about the registrar client that is spawning the conference.

Caller: A GroupKit conference application, e.g.. simplebrain.tcl. Note that simplebrain does not pass gk_initConf the above parameters. The parameters are passed on the command line by **_gk_createConference** in rc.tcl

Function: Initializes the conference process and connects back to the registrar client by calling **_gk_remoteInfo**.

_gk_connectTo

Parameters: **whom**—the conference process with whom to connect.

Caller: **_gk_joinTo** in rc.tcl.

Function: Connects to conference process whom by calling its **_gk_remoteInfo** procedure.

_gk_remoteInfo

Parameters: **whom**—the conference process with whom to connect.

echo—a hack, really. A boolean variable used to make sure that _gk_remoteInfo is executed exactly twice—once in each of the conference processes attempting to connect.

Caller: **_gk_connectTo** in conf.tcl

Function: Connects to a remote conference process by putting the remote user in _gk_remoteUsers. It also sets the gk_newUser flag.

_gk_removeUser

Parameters: **whom**—the keyed list of the user to be removed.

Caller: gk_foundDeletedUser in openrc.tcl, or a similar routine in another registrar client.

Function: Removes a user from the _gk_remoteUsers or _gk_localUser list, depending on whether the user is the local user. If the user is the local user, the conference process is also killed. _gk_removeUser sets the gk_deletedUser flag.

_gk_deleteConf

This procedure just exits this conference process.

_gk_doUpdateEntrant

Parameters: none.

Caller: called on a trace to gk_newUser.

Function: It finds the conference participant with the smallest user number (the oldest participant). If the local conference process is that conference participant, it sets the variable “gk_entrantUserNum” to the user number of the new user. Applications put a trace on this variable in order to update conference entrants.

REGISTRATION SYSTEM FLOW OF CONTROL

The best way to understand the flow of control in GroupKit is through a trace. We begin with a very high level description of the interaction between modules. Then we shall go through a more detailed procedure-by-procedure trace.

High Level Description

Let's use openrc.tcl as the registrar client. When a user starts a new conference, the registrar client informs the registrar. The registrar puts the new conference in _gk_regConfsList, its list of conferences. The registrar client (Rc) tells the registrar to broadcast _gk_regConfsList to all the connected Rc's. All the Rc's then compare this list with their own list of conferences, _gk_rcConfsList. They will find a discrepancy—there is a new conference in the _gk_regConfsList. According to the protocol in openrc, all the registrar clients do is add the new conference to their lists.

The registrar client who started the new conference (call it Rc A) does a little bit more—it tells the registrar that it is joining the conference. (The others need not do this since they only care about the conference's existence until their users ask to join it). The registrar adds the new user to its _gk_regUsersList. Rc A then tells the registrar to broadcast its new list of users to all the Rc's. All the Rc's then see that there is a new user in a conference, and following their openrc protocol, add the new user to their _gk_rcUsersList's.

Again, Rc A does a little bit more. Note that the conference process has NOT yet been spawned. Only a record of the conference has been made by the registrar and registrar clients. Rc A notices that he is the new user, and so spawns a conference process. When the conference process begins, it connects back to it's registrar client (Rc A). Now, we have a conference running.

Imagine that a user on another registrar client (call it Rc B) wants to join that conference. Rc B tells this to the registrar who adds the new user to its _gk_regUsersList. Rc B then tells the registrar to broadcast this list to all the registrar clients. The Rc's notice a new user (because of the difference between _gk_regUsersList and their _gk_rcUsersList's) and, according to the openrc protocol, add the new user to their lists.

Again, the joining registrar client (Rc B) does a little bit more—it spawns a conference process. However, it cannot stop there. This conference process must connect to the conference process belonging to Rc A. So it asks Rc B for the address of Rc A's conference process and connects itself to it. It also tells Rc A's conference process to connect back.

Now we have a conference with two participants, each running one process in the replicated architecture that makes up a conference.

Procedure Trace

We now go through the same sequence as above procedure by procedure.

Notation is as follows: Reg is the registrar module, Rc is a generic registrar client module, openrc is openrc.tcl, and Conf is a conference process. All procedures calls have the format:

module::procedure name{important parameters}
{comments}.

Again, assume two openrc registrar clients running already, and a user wants to start a conference. We trace through the activities of the first registrar client (Rc1) as a new conference is being created through it. We then trace the second registrar client joining the new conference.

openrc::_gk_doNewConf
- the keys confname, conftype, and originator are set for the new conference.
- call to Rc::_gk_callNewConference{conf}

Rc::_gk_callNewConference
- call to Reg::_gk_newConference{conf}

Reg::_gk_newConference
- puts conf in _gk_regConfsList.

Back in openrc::_gk_doNewConf
- call to Rc::_gk_pollConferences

Rc::_gk_pollConferences
- call to Reg::_gk_dispConference

Reg::_gk_dispConference
- sends _gk_regConfsList to all connected registrar clients.
- call to Rc::_gk_confList {_gk_regConfsList}

Rc::_gk_confList
- supposed to sort out differences between _gk_regConfsList and _gk_rcConfsList.
- calls Rc::_gk_updateList to find the differences and generate lists of new and deleted conferences.
- tells _gk_updateList to call procedures in openrc that HANDLE new and deleted conferences

Rc::_gk_udpateList
- finds differences between _gk_regConfsList and _gk_rcConfsList.
- calls openrc::_gk_foundNewConf and openrc::_gk_foundDeletedConfs with new and deleted conferences respectively.

openrc::_gk_foundNewConf
- puts new conf in _gk_rcConfsList.
- if the conference originator is the local openrc (true in this case), it joins the conference by calling Rc::_gk_callJoinConference{confnum} and Rc::_gk_pollUsers{confnum}.

Rc::_gk_callJoinConference
- initializes a "user" keyed list with keys confnum, userid, host and port of the registrar client, and username.
- calls Reg::_gk_addUser{user}

Reg::_gk_addUser
- adds user into _gk_regUsersList.

Back in openrc::_gk_foundNewConf
- calls Rc::_gk_pollUsers

Rc::_gk_pollUsers
- call to Reg::_gk_dispUsers

Reg::_gk_dispUsers
- sends _gk_regUsersList to all connected registrar clients.
- call to Rc::_gk_userList{_gk_regUsersList}

Rc::_gk_userList
- supposed to sort out differences between _gk_regUsersList and _gk_rcUsersList.
- calls Rc::_gk_updateList to find the differences and generate lists of new and deleted users.
- tells _gk_updateList to call procedures in openrc that HANDLE new and deleted users.

Rc::_gk_udpateList
- calls openrc::_gk_foundNewUser and _gk_foundDeletedUser with new and deleted users respectively.

openrc::_gk_foundNewUser
- puts new user in _gk_rcUsersList.
- if the user is local to this registrar client (true in this case), a conference process must be created. So call Rc::_gk_createConference{confnum usernum}.

Rc::_gk_createConference
- creates a new conference process.
- we now have an application running. Applications normally call Conf::_gk_initConf.

Conf::_gk_initConf

- does initialization and calls Rc::_gk_remoteInfo with a keyed list containing confnum, the conference host and port number, and the usernum.

Rc::_gk_remoteInfo

- associates a file descriptor with the new conference process and puts it in it's local list "conferences. The local registrar client now knows of the new conference process.

We now have a conference process (CP1) running on one of the registrar clients as, essentially, a single-user application. The second registrar client (Rc2) knows of this conference through the broadcasts by the registrar. Now, say the user of this registrar client wants to join the new conference. He double-clicks on the conference and:

openrc::_gk_doJoin

- if this conference has not already been joined, call Rc::_gk_callJoinConference.

Rc::_gk_callJoinConference

- initialize "user" keyed list and call Reg::_gk_addUser.

Reg::_gk_addUser

- puts user in _gk_regUsersList.

Back in openrc::_gk_doJoin

- call Rc::_gk_pollUsers{confnum}

Rc::_gk_pollUsers

- a sequence similar to the one above, with calls to Reg::_gk_dispUsers, Rc::_gk_userList, Rc::_gk_updateList and openrc::_gk_foundNewUser takes place.

openrc::_gk_foundNewUser

- Rc::_gk_createConference is called following the same sequence as above.
- Rc2 now has a new conference process (call it CP2).
- Rc2::_gk_joinTo{"old user" "new user"} is now called since **there is an existing user in the conference**.

Rc2::_gk_joinTo

- calls Rc1::_gk_address{confnum} to get the address (host and port number) of Rc1's conference process so that it can tell it's conference process join Rc1's.
- calls Rc2::_gk_toConf{confnum "run Conf::_gk_connectTo with parameter "address"}

Rc2::_gk_toConf

- calls Conf::_gk_connectTo.

Conf::_gk_connectTo

- calls Conf:: _gk_remoteInfo in CP1, telling CP1 about CP2.

Conf::_gk_remoteInfo in CP1

- stores information about CP2.
- calls Conf::_gk_remoteInfo in CP2.

Conf::_gk_remoteInfo in CP2

- stores information about CP1.
- now CP1 is connected to CP2 and vice-versa.

Now we have a conference running with two users.

APPLICATION LEVEL CONFERENCE COMMUNICATION

conf.tcl contains some procedures to be used by conference applications for communication amongst conference participants. These are listed and described below:

gk_findUser (idnum)
Returns the keyed list of the user with usernum=idnum.

gk_toAll args
Sends the Tcl command formed by "args" to every user of a conference, including the local user.

gk_toOthers args
Sends the Tcl command formed by "args" to every remote user of a conference.

gk_getUserAttrib (UserNum AttribName}
Gets attribute AttribName of user UserNum from the _gk_remoteUsers keyed list.

gk_getLocalAttrib {AttribName}
Gets attribute AttribName from the _gk_localUser keyed list.

gk_setUserAttrib (UserNum AttribName NewValue}
Sets attribute AttribName of user UserNum to NewValue in the _gk_remoteUsers keyed list.

gk_setLocalAttrib {AttribName NewValue}
Sets attribute AttribName to NewValue in the _gk_localUser keyed list.

gk_toUserNum {UserNum args}
Sends the message in "args" to user UserNum.

gk_amOriginator {}
Returns 1 if the local user is the conference originator, 0 if he is not.

GROUPKIT WIDGETS AND ERRORS MODULE

Widgets

The file gkwidgets.tcl contains some handy GroupKit widgets and routines that manipulate them. **gk_defaultMenu** is called with one argument in which it returns a generic pulldown menu bar with the following menu buttons:

File: A "Quit" option is the only one under File.

Help: "About GroupKit" is the only option under Help.

Collaboration: This has the "Show Users" option which displays a list of the participants in a conference. The list is generated from _gk_localUser and _gk_remoteUsers.

The callback functions for the menu choices listed above are as follows:

_gk_removeUserFromWindow is called whenever there is a change to the "gk_deletedUser" variable in conf.tcl by way of a trace on the variable. The procedure updates the list of users displayed when "Show Users" clicked on.

_gk_addUserToWindow is called when there is a change to the "gk_newUser" variable in conf.tcl. The procedure updates the list of users displayed when "Show Users" clicked on.

_gk_showUsers is called when "Show Users" is clicked upon. It displays the list of participants in a conference.

_gk_about is called when the "About GroupKit" option is chosen from the pulldown menu. It proudly displays the GroupKit logo.

The second widget is a simpler generic pulldown menu bar with only the "File" and "Help" menu buttons. It is obtained by calling **gk_defaultRCMenu** with a parameter in which the menu is to go. This menu is meant for registrar clients.

The third widget is a vertical remote scrollbar, a device that displays the position of a remote user's scrollbar. The remote scrollbar widget provided in gkwidgets.tcl is contains both a remote scrollbar and the local user's real scrollbar. The following procedures implement the remote scrollbar.

gk_groupScroll is called by the application to initialize a remote scrollbar widget. The first parameter is the frame in which the remote (or remote) and real scrollbar will be placed. The second parameter is a command to be used in the initialization of the real scrollbar. This command tells the real scrollbar which object to scroll and in which direction. The following is an example of how **gk_groupScroll** is called:

```
gk_groupScroll yourScrollFrame -command "yourScrolledObject \
   yview"
```

gk_scroller is the callback function that should be called by the application when a change occurs in the object being scrolled. It should be put in the declaration of yourScrolledObject. For example,

```
.yourScrolledObject -yscrollcommand "gk_scroller _yourScrollFrame"
```

_gk_adjustRemote adjusts the remote scrollbars based on information on the movement of a real scrollbar.

_gk_newScroll is called when a change occurs to the variable gk_newUser. It adds a new bar (representing the new user) to the remote scrollbar.

_gk_removeScroll is called when the global gk_deletedUser is changed. It removes the bar of the deleted user from the local remote scrollbar.

_gk_receiveCoords is called from **gk_newScroll** to update a new user on where it should put its representation of your scrollbar. It is passed the coordinates of your scrollbar and your user number.

_gk_chooseColour asks a remote user to announce his colour so that the local remote scollbar can use that colour.

_gk_setColour is used by the remote user to announce his colour.

_gk_receiveColour receives the remote user's colour.

The last widget provided in gkwidgets is a remote cursor widget. The cursors are actually labels with the name of the user that the cursor is representing. The following procedures implement remote cursors:

gk_makeRemoteCursors is called by the application with the canvas that is to contain the remote cursors as a parameter. It initializes the globals "cursors".

gk_callMoveCursor must be called by an application with the canvas and the location of the real cursor as parameters. It gives this information to the remote users.

_gk_moveCursor tells the remote users to move their remote cursors.

_gk_removeCursor is a callback from a trace on the variable "gk_deletedUser". It removes the remote cursor of the deleted user.

gkwidgets.tcl also provides **gk_winConfigure**, a generic procedure used for configuring all the children of a certain window. For example,

```
gk_winConfigure window -bg HotPink -fg green
```

Error Handling

The module errors.tcl contains two functions. The first, **tkerror**, is the usual Tcl error handling call-back. The second, **gk_properQuit**, is called when a user quits a conference application.