Building Real-Time Groupware with GroupKit, A Groupware Toolkit

MARK ROSEMAN and SAUL GREENBERG
University of Calgary

This article presents an overview of GroupKit, a groupware toolkit that lets developers build applications for synchronous and distributed computer-based conferencing. GroupKit was constructed from our belief that programming groupware should be only slightly harder than building functionally similar single-user systems. We have been able to significantly reduce the implementation complexity of groupware through the key features that comprise GroupKit. A *runtime infrastructure* automatically manages the creation, interconnection, and communications of the distributed processes that comprise conference sessions. A set of *groupware programming abstractions* allows developers to control the behavior of distributed processes, to take action on state changes, and to share relevant data. *Groupware widgets* let interface features of value to conference participants to be easily added to groupware applications. *Session managers*—interfaces that let people create and manage their meetings—are decoupled from groupware applications and are built by developers to accommodate the group's working style. Example GroupKit applications in a variety of domains have been implemented with only modest effort.

Categories and Subject Descriptors: D.2.2 [**Software Engineering**]: Tools and Techniques—*user interfaces*; D.3.3 [**Programming Languages**]: Language Constructs and Features; D.4.1 [**Operating Systems**]: Organization and Design—*interactive systems*; H.5.2 [**Information Interfaces and Presentation**]: User Interfaces—*user interface management systems*; H.5.3 [**Information Interfaces and Presentation**]: Group and Organization Interfaces—*synchronous interaction*

General Terms: Human Factors

Additional Key Words and Phrases: Computer-supported cooperative work, GroupKit, groupware toolkits, synchronous groupware, user interface toolkits

1. INTRODUCTION

Over the last few years, we have been designing groupware for synchronous distributed conferencing, where two or more distance-separated people work on a shared task in real time. Our first system, called Share [Greenberg 1990], allowed participants in a distributed meeting to take

This research was supported in part by the National Sciences and Engineering Research Council of Canada and by Intel Corporation.
Authors' address: Department of Computer Science, University of Calgary, Calgary, Alberta T2N 1N4, Canada; email: {roseman; saul}@cpsc.ucalgary.ca.

Table I. Core Requirements for a Groupware Toolkit

Design Requirement	Rationale	Examples
User Centered		
Support multiuser actions and awareness of others over a visual work surface.	Many groupware applications can be seen as shared work surfaces, and studies (e.g., Tang [1991], Gutwin and Greenberg [1995], and Dourish and Bellotti [1992]) have attempted to characterize some of the properties necessary to support collaborative work. Supporting these in a toolkit can encourage developers to build more usable applications.	•Telepointers •Graphical Annotations •Multiuser Scrollbars •Gestalt Views
Provide support for structuring group processes but do so in a flexible enough fashion to accommodate the diverse needs of different groups.	Rather than adopt a single model for how groups should interact, a toolkit should provide a range of facilities to support the needs and working styles of different groups, while allowing application developers to extend these to support specific needs [Greenberg 1991]. A toolkit should support building applications relying on either social protocols or highly structured and automated process models.	*Different Policies for:* •Floor Control •Session Management •Access to Conferences •Treatment of Latecomers
Integrate groupware with conventional ways of doing work.	Groupware should not pose a barrier to "individual" ways of doing work, but instead it should be smoothly integrated [Ishii and Kobayashi 1992]. Access to both single-user applications or even noncomputer resources [Ishii 1990] is important, as is the availability of traditional communication mediums.	•Shared Terminals •Telephone Links •Video Conferencing
Programmer Centered		
Provide technical support to deal with multiple distributed processes.	Groupware systems are composed of multiple processes that communicate over a network, and toolkits can augment the operating-system-level support by simplifying creation, interconnection, and teardown of the processes. A toolkit can also provide communications models and concurrency control mechanisms that abstract away from the operating system.	•Session Management •Process Creation •Locate Other Processes •Multicast Abstractions •Session Persistence •Message Serialization and Data Locking
Provide support for shared data.	Distributed groupware should also be able to share its common data easily and should be able to detect and take action when data are changed. Concurrency control should be available to keep data consistent [Greenberg and Marwood 1994].	•Shared Environments •Data Serialization and Locking •Binding Callbacks to Environment Events
Provide support for shared data and an extensible shared-graphics model.	Since many groupware applications can be seen as shared visual work surfaces, a toolkit should support the creation and manipulation of both generic and application-specific objects on a graphical work surface. Paradigms such as Abstraction-Link-View [Patterson 1991] can be supported by the toolkit to facilitate this.	•Shared Graphics and Primitives •Object Concurrency Control •Separate View from Object Representation

turns sharing unaltered single-user applications; group interaction was mediated by choosing from a variety of flexible floor control mechanisms to best match the particular style of the meeting [Greenberg 1991]. We then built several groupware equivalents of paint and structured drawing programs [Greenberg et al. 1992]. GroupSketch was a minimalist bitmapped sketchpad, where all users saw exactly the same things on their display, as well as the multiple pointers of other participants. X/GroupSketch was a second-generation version that, among its additional features,

allowed participants to view different parts of the document. GroupDraw was a prototype of a structured drawing application, where participants could jointly manipulate a variety of objects, such as lines and rectangles, on their display.

Building groupware proved a frustrating experience. Implementing even the simplest systems was a lengthy process, as much time was spent inventing similar ideas over and over again. Some of the common tasks of the programmer included the following items. The setup and management of distributed processes had to be arranged. Interprocess communication links had to be established. Actions between processes had to be coordinated and their internal states updated as people interacted with the system. Similar interface components had to be reimplemented to provide for generic group needs. Session managers had to be supplied so that people could create, monitor, and enter the conferences.

Consequently, we decided to implement a groupware toolkit to support programming of synchronous and distributed computer-based conferencing systems. Our motivation for this project was our belief that:

> A developer using a well-designed toolkit should find it only slightly harder to program usable groupware systems when compared to the effort required to program an equivalent single-user system.

We took the tasks common to almost all groupware programming (noted earlier) and transformed them into a set of core user and programmer-centered requirements for a groupware toolkit. These are summarized in Table I and are described in more detail elsewhere [Roseman and Greenberg 1992]. The table also lists the rationale behind the requirements. From these requirements, we believed that a toolkit could reduce implementation complexity by providing the following generic features:

—A *runtime infrastructure* would automatically create processes and manage their interconnections and communications.

—A simple set of *groupware programming abstractions*, built on top of a conventional language and GUI toolkit, would be available to groupware developers. Primitives would include remote procedure calls between application instances, sharing of data, and generation and tracking of conferencing events.

—A set of *groupware widgets* would let developers easily add generic interface constructs of value to conference participants, resulting in better and more usable groupware systems.

—*Session management*, the mechanism by which people create and manage meetings, would be handled separately from the groupware applications. Primitives for constructing different session managers would be available for developers wishing to create custom interfaces that suit the particular needs of a group.

The result of our efforts is GroupKit, a toolkit whose design has been evolving over several years. The first generation of GroupKit, developed in

C++ and InterViews [Linton et al. 1989], has been described in a previous paper [Roseman and Greenberg 1992]. Based on our experiences with that system, we constructed a second-generation version that has proven to be an even richer platform for developing groupware. Readers familiar with the earlier work should find that the version described here both generalizes and extends the earlier system.

GroupKit and its applications still run on Unix workstations under an X11 environment. However, the system now uses the interpreted Tcl language and Tk interface toolkit [Ousterhout 1994] and the Tcl-DP socket extensions [Smith et al. 1993]. GroupKit developers build their applications using Tcl/Tk as well as the extensions provided by our toolkit. GroupKit and the underlying systems are all freely available via anonymous ftp; details are provided at the end of this article.[1]

The article begins with an overview of GroupKit. It shows both how an end-user sees systems constructed with the toolkit and what a GroupKit program looks like. Subsequent sections detail GroupKit's features—its runtime infrastructure, its groupware programming abstractions, the set of groupware widgets, and its session management. These sections show how some of these features work in practice by including both screen snapshots and code fragments from existing applications. The examples also illustrate the wide variety of systems that can be built in GroupKit. The article then evaluates GroupKit by examining the effort required to build groupware applications in it. It closes by comparing GroupKit to other groupware toolkits. A video is also available [Greenberg and Roseman 1994] that captures the dynamics of many of the screen snapshots described in this article.

2. OVERVIEW

Before delving into technical details, it is worth getting an overall feel for GroupKit. To set the scene, this section begins by showing what end-users of programs built in GroupKit may see. It then takes the developer's view, by tracing through a simple GroupKit program.

The scenarios presume that users' computers are running X Windows within a Unix environment, that computers are interconnected using TCP/IP protocol over the Internet, and that the GroupKit software has been installed. Users may be located anywhere on the Internet as long as their network connection does not suffer excessive latency (which would compromise interactive performance of some applications). Because GroupKit can run in most Unix systems, the actual machine type does not matter.[2]

[1] All the systems that comprise GroupKit are under active development, with new versions of software appearing periodically. This article is mostly based upon GroupKit 3.1, Tcl 7.4, Tk 4.0, and Tcl-DP 3.2.

[2] For example, GroupKit has been successfully installed on Sun, HP, RS6000, and Silicon Graphics workstations as well as PCs running Linux.

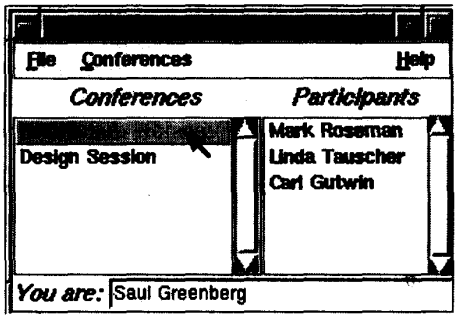

Fig. 1. The Open Registration session manager: two conference sessions are shown, with three participants present in the PostIt conference.

2.1 An Example of an End-User's View of GroupKit

This section walks through an example GroupKit session, where we will see a user monitoring conferences in progress, joining two existing conferences, and then creating a new conference. Although the scenario illustrates actual systems provided in the GroupKit release, it is important to remember that these are just examples of systems that programmers could build with the toolkit.

First, the user (Saul) invokes a *session manager*, in this case the "Open Registration" manager (Figure 1). In the "Conferences" pane, Saul sees that two conferences are in progress: "PostIt" and "Design Session." By selecting one of them, he can then see who is in a particular conference (the list in the "Participants" pane).

Next, Saul joins the "PostIt" conference by double clicking its name, which adds him to the list of participants. The PostIt Editor then appears on his display (Figure 2, left window). With this simple GroupKit application, he can type a short message and send it to one or more participants. The selected participants will see the message appear in a pop-up window. In Saul's work community, everyone uses PostIt to see who is available and to invite people into conferences. Shortly after joining, Saul receives a PostIt message from Linda inviting him to join the "Design Session" conference (Figure 2, right window).

Saul joins the "Design Session" conference via the session manager, and GroupSketch, a multiuser sketchpad that allows simultaneous drawing, then appears on his display (Figure 3). It contains the group drawing being worked on, as well as the telepointers of the other participants. The voice connection is made through a telephone conference call. After discussing the figure, Saul leaves the conference by selecting the "Quit" option from the "File" menu.

A bit later on, Saul receives a phone call from his colleague Judy, who wishes to discuss a document. Saul decides to create a new conference that runs the FileViewer. This application provides a relaxed what-you-see-is-

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.