Groupware Toolkits for Collaborative Mobile Groupware

Vijay Gupta

Research Scholar, Pacific university, Udaipur India

ABSTRACT:- ***Groupware toolkits provide application developers with a range of facilities for reducing the complexity of building distributed groupware. In general, toolkits support their own particular paradigm for developing groupware applications and developing groupware within his paradigm is usually convenient. However, attempting to program outside the supported paradigm is often either difficult or impossible.***

----------♦----------

INTRODUCTION

OVERVIEW

This section considers the suitability of a range of groupware toolkits for developing mobile groupware, with particular emphasis on *Group Kit*. In common with distributed systems toolkits, groupware toolkits enable developers to create distributed applications without regard to certain distributed details. However, unlike distributed systems toolkits, groupware toolkits also provide developers with a wide range of high level programming tools specifically aimed to ease the task of building groupware applications. A wide selection of groupware toolkits currently exist, ranging from relatively simple toolkits that offer developers support for creating a specific class of groupware application (e.g. the Dist Edit toolkit designed to support the development of shared text editors) to more flexible toolkits that provide developers with tools for creating highly sophisticated, graphical based, groupware. However, as one might expect, these toolkits have been developed assuming a fixed and reliable underlying communications infrastructure.

GROUP KIT: A TYPICAL MODERN GROUPWARE TOOLKIT

AN OVERVIEW OF GROUP KIT

The group Kit toolkit [Roseman,96], [Roseman,92] extends the standard Tcl/Tk toolkit to provide developers with an application infrastructure for building distributed groupware. GroupKit is currently available for UNIX, Windows and Apple based platforms and thus enables groupware to be built that is capable of operating over heterogeneous platforms.

Group Kit utilises a semi-replicated data architecture and provides developers with a variety of multi-user widgets, tools for handling session management and support for managing shared information.

GROUP KIT'S ARCHITECTURE

A typical Group Kit run-time process model is illustrated in figure 3.5. This shows a collaboration between two workstations each running two conferences: 'A' and 'B'.

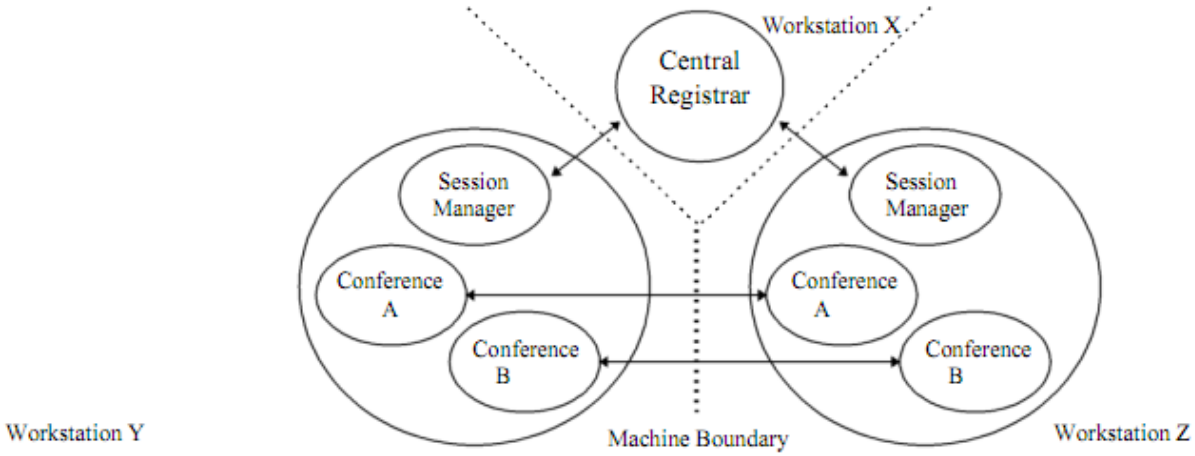

Figure 3.5, An example of Group Kit's run-time process model.

The ovals in the diagram represent instances of processes running on each machine, and the directed lines joining them indicate communication paths. The figure shows three types of GroupKit process: the centralised registrar, the replicated session managers and the replicated conference applications.

The registrar maintains a list of all conferences and the users in each conference. It thus serves as an initial contact point to locate existing conference processes and their addresses. In order for new conferences to be established, the address of the registrar needs to be known to all other processes.

The session manager process is replicated per user. It provides both a user interface and a policy dictating how conferences are created or deleted, how users are permitted to enter and leave conferences, and how conference status is presented. When session manager processes are created, they connect to the registrar.

The conference application is a GroupKit program which can be invoked by the user via the session manager. Conference applications typically interact as replicated processes, with a replica running on each participant's workstation.

This run-time infrastructure is maintained entirely by GroupKit and the conference application code is not required to take explicit action regarding the creation of processes or communication establishment. Programmers are required to build both session managers and conference applications. Also, programmers need to be aware that they are building distributed applications, and must attend to issues such as concurrency control and synchronization.

GROUP KIT'S SUPPORT FOR AWARENESS

Group Kit supports awareness by providing pre-packaged multi-user widget sets. These enable programmers to build multi-user awareness functionality into groupware with relatively little development effort. Three examples of multi-user widgets provided by GroupKit are: participant status widgets, telepointer widgets and multi-user scrollbars.

- ***Participant Status Widgets***

GroupKit provides a special widget for reflecting the status of participating users. For example, as users enter and leave a conference other users can observe this activity and obtain information about conference participants. If a user wishes to obtain details about another user then, by simply selecting that user's icon, a *business card* containing further information about the user can be displayed.

- ***Telepointer Widgets***

The API provided by GroupKit enables telepointers to be added to an application using no more than a few

lines of code, such as:-

gk_initializeTelepointers
gk_specializeWidgetTreeTelepointer.canvas

Group Kit's telepointers can be used to support relaxed WYSIWIS by having the telepointer drawn relative to a specified widget, rather than the application window.

- ***Multi-user Scrollbars***

GroupKit supports multi-user scrollbars to provide users with workspace awareness, i.e. an awareness of where others are working in a large document. Group Kit's multi-user scrollbars consist of two parts: a conventional scrollbar showing the local user's viewing position in the shared document and a set of additional scrollbars showing where other users are viewing the shared document.

GROUP KIT'S SESSION MANAGEMENT

The GroupKit session manager enables a user to enter a group conference in a number of different ways: the user can explicitly join the conference, implicitly join the conference via an invitation, or create the conference. The session manager also manages the departure of group members. When the last group member elects to leave a conference, that person is asked if the conference application should persist. If the conference is required to persist then its state is saved so that it can be re-entered later with its contents intact.

GROUP KIT'S MULTICAST REMOTE PROCEDURE CALL MODEL

Group Kit's multicast model is used to communicate changes and trigger program execution across the application processes contained in a session. The multicast RPC (Remote Procedure Call) model supported by GroupKit hides all routing and communications details from the programmer and provides both blocking and non-blocking forms of RPC. Group Kit provides one blocking RPC primitive, i.e. *gk_serialise* (described in section 3.3.2.6) and three non-blocking RPC primitives. The first, called *gk_toAll*, multicasts the procedure request to all conference processes in the session, including the local user. The second, called *gk_toOthers*, multicasts the procedure request to all other remote conference processes in the session except the local process that generated the call. The third form, *gk_toUsernum*, directs the request to a particular conference process.

GROUP KIT'S SUPPORT FOR MANAGING SHARED INFORMATION

Group Kit supports the separation of an application's data model from its associated graphical view (described in section 3.2.3.1) by providing a shared data model called an *environment*. This takes the form of a dictionary-style data structure containing keys and associated values. While instances of environments run on different processes, the run-time system makes sure that changes to one instance are propagated to other instances. Changes to an environment's state can be tracked as events that trigger Group Kit's notification mechanism. Using this event/notification mechanism, the programmer can bind call-backs to an environment, and receive notification when changes to that environment occur. Interface code can thus be written that adjusts the local graphical view when changes to the environment occur.

Group Kit supports a variety of different concurrency control mechanisms. In more detail, apart from selecting to have no concurrency control the application developer can select to use either serialisation or locking. Serialisation ensures that events are received across all conference processes in the same order. In order to utilise the serialisation mechanism a *gk_serialize* multicast primitive is used as opposed to the *gk_toAll* primitive; this causes the message to be sent to a serialising process which in turn sends the message to each conference process, including the local process. Group Kit's support for locking utilises a special purpose *lock manager* which enables the use of both optimistic and pessimistic locking strategies. In addition, the lock manager enables the developer to request the identity of a lock's owner and also supports the specification of sophisticated conflict detection schemes, e.g. schemes based on a locking hierarchy.

OTHER TOOLKITS

A number of other toolkits (described below) raise interesting ideas when considered in the context of mobile groupware.

- ***The Caelum Toolkit***

The *Caelum* toolkit [Anker,97] is a general framework for constructing distributed groupware. Application developers are provided with a software development kit (SDK) for constructing groupware based around the concept of process groups. One very useful feature provided by the toolkit is the ability to associate a pre-defined time-out period with a message that allows the application to decide whether the message should be delivered or discarded should the time-out period be exceeded. This facility enables data consistency to be

determined at the application level and the level of consistency chosen could, in theory, be chosen by the user, given the requirements of their current collaboration and the state of the network.

In addition, the toolkit enables QoS requirements to be associated with group multicast. More specifically, application programmers can choose the strength of message ordering guarantee based on application requirements. So, for example, the costly virtual synchrony guarantee could be used for shared data applications with strong consistency requirements but not used for those performance oriented applications requiring soft real- time message delivery, e.g. video transmission.

- ***The Java Shared Data Toolkit***

The *Java Shared Data Toolkit* [Burridge,98] is of interest because it supports the notion of a one-to-many communications channel. These channels can be created with certain QoS properties including reliability and ordering. This functionality provides the programmer with some degree of control over the consistency/performance trade-off.

Another useful feature of this toolkit is the ability to raise and manage exceptions when certain communication difficulties arise. Although the exceptions currently supported are relatively basic (currently, a single *ConnectionException* handles all communication problems) the basic mechanism is very flexible and could enable the application to receive notification when certain *interesting* communication problems occur.

- ***The GEN Toolkit***

The prototype *GEN* toolkit [O'Grady,96] is based on the open implementation design principle (as adopted by Adapt described in section 2.4.4.2) in order to dynamically support a range of different data sharing strategies. O'Grady's key motivation for utilising open implementation techniques was to break down the common *black-box* approach [Kiczales,96] towards toolkit design, whereby a toolkit supports a given API but the actual underlying implementation is hidden from the programmer. It can be argued that this leaves the programmer with little flexibility if the API does not meet the requirements of the application currently being developed, e.g. if the consistency strategy supported by the toolkit is stronger than the application being developed requires. Furthermore, the black- box approach can prematurely fix the balance of certain system trade-offs.

The approach adopted by GEN enables developers to control the consistency/performance trade-off in order to meet the current application's requirements. In more detail, the GEN toolkit supports open implementation by supplying two distinct components: an API for utilising a set of default data sharing strategies and a *meta-interface* for enabling the programmer to create new mechanisms for data distribution and concurrency control.

- ***The Prospero Toolkit***

Prospero [Dourish,96b] is another example of a groupware toolkit that is strongly based around the open implementation technique. In common with GEN, Prospero concentrates on enabling the application programmer to revise and adapt distributed data management and concurrency control mechanisms and achieves this via the use of *meta- object* protocols [Dourish,95]. The toolkit is implemented in an enhanced version of Common Lisp which provides support for meta-objects and enables strategies to be changed through the standard object-oriented techniques of *subclassing* and *specialisation*. At the default level, Prospero supports consistency management by providing *activity streams* which can be allowed to *diverge* (or forced to re-synchronise) depending on the consistency required between streams. Conversely, at the meta-level, Prospero enables the implementation of these streams to be treated as meta-objects and therefore open to change or specialisation by the application developer.

ANALYSIS OF GROUPWARE TOOLKITS

The GroupKit toolkit, in common with other groupware toolkits, provides developers with an appropriate run-time infrastructure and API for developing groupware designed to operate over reliable network infrastructures. However, the multicast mechanism provided by the toolkit has certain problems when used for constructing mobile groupware. The fundamental difficulty is that the RPC mechanism hides too many communication details from the application programmer, i.e. it strictly enforces transparency. This is contrary to the kind of support required given the potential for communication difficulties. In fact, application programmers require flexible means for receiving notification and feedback regarding the existence of communication difficulties between any group members.

Many early toolkits, such as DistEdit, provided insufficient flexibility for managing shared data in a mobile environment. This is because they tended to utilise causal or total ordering guarantees in order to enforce a

very strict level of consistency between members views. Such ordering semantics might not be required and impose unnecessary performance overheads on a group's collaboration. The interesting contribution of the Caelum and Java Shared Data toolkits are that they both provide the application programmer with some degree of control over the strength of ordering used. In addition, the Java Shared Data toolkit supports the kind of flexible notification scheme required by developers of mobile groupware.

The work on open implementation techniques (utilised by the GEN and Prospero toolkits) tackles the need for flexibility by enabling developers to actually control the implementation of the system's data sharing strategy. This approach enables developers to tailor data distribution strategies in order to match both the consistency/performance requirements of the application and the current quality of group communications. Interestingly, as part of his motivation for utilising open implementation techniques, Dourish argues that toolkits need to support both *dynamic* and *implementational* flexibility. In more detail, dynamic flexibility refers to the requirement that the system should be able to match and adapt to changes in the group's interactions over time, and, implementational flexibility refers to the need for systems to be able to respond to a variety of implementation environments, which may change over time. This latter form of flexibility is of particular relevance for mobile groupware because the underlying techniques for supporting collaboration need to react or adapt to dynamic changes in the networking environment.

CONCLUSIONS

From this work, the following conclusions can be made:-

- ***Implications for managing shared data in a mobile environment***

In an unreliable mobile environment, the provision of appropriate support for managing shared information is difficult. If a replicated data architecture is used then groupware toolkits tend to enforce consistency between members' views by using an atomic broadcast mechanism to broadcast server updates to remote sites. However, as described in section 2.4.8.1, by using such a broadcast mechanism, if one group member suffers network difficulties (and so cannot receive the update) then group consistency is enforced by ensuring that no members receive the update. Conversely, if a centralised data architecture is chosen, then the process managing the shared data can become a performance bottleneck.

- ***The need for additional forms of awareness in a mobile environment***

The concept of awareness is of particular relevance in a mobile environment. It can be argued that new forms of awareness are required to provide group members with appropriate feedback to make them aware (or rather as aware as they wish to be) of the affect that fluctuations in the quality of group communications could have on their collaboration. This would, therefore, save them from being forced to make assumptions regarding the current state of their connectivity with the rest of the group and also give them the opportunity to adapt their behaviour appropriately.

- ***The need for suitable notification mechanisms***

In general, notification mechanisms are useful tools for supporting awareness (see [Ramduny,98] for a general discussion and exploration on the role of notification servers). In particular, appropriate notification mechanisms are required when developing mobile groupware in order to enable actions to be triggered when certain group communication problems occur. Given such notification, applications can provide the style of mobile awareness described above and also have the ability to adapt their data management strategies in response to changes in group connectivity. However, in order for such event notification mechanisms to operate effectively, low-level communication information needs to be provided by the underlying transport service.

- ***The importance of flexible coupling in a mobile environment***

The need for groupware that supports flexible coupling is of particular importance in a mobile environment. At the interaction level, this should enable users to switch between asynchronous and synchronous styles of interaction depending on the current quality of group communications and the requirements of the current task. More specifically, when group communications is poor then an asynchronous style can be used to reduce the frequency of communication with other group members. Conversely, a more synchronous style of interaction can be used when the quality of the underlying network can support more frequent communication between group members. It can thus be argued that, in a mobile environment, where the quality of group communications is often dynamic, the classical distinction between asynchronous and synchronous

interaction [Ellis,91] should be less distinct [Cheverst,96].

- ***The need for enhanced multi-user widgets in a mobile environment***

Toolkits providing multi-user widgets need to support the issues of awareness and flexible coupling described above. For example, a multi-user telepointer widget needs to support flexible coupling such that when the quality of group communications is high the telepointer widget can behave in a tightly coupled manner, i.e. by continually tracking and propagating each member's mouse movements. However, when the quality of the network degrades, then a loosely coupled form of telepointer is required. This version could be designed in such a way that it only broadcasts a member's cursor position when that member indicates that it is necessary to do so. Regarding the issue of awareness, a multi- user status widget could provide information such as whether a group member had formally left the group, or was experiencing a period of disconnection. In addition, such status widgets could reveal certain details regarding a group member's connection, e.g. the level of cost associated with his/her network communication.

- ***Implications for session management in a mobile environment***

In a mobile environment the issue of providing suitable session management is complicated by the fact that group members can temporarily lose network connectivity. For this reason session managers should provide suitable support for bringing such group members up-to-state with the rest of the group. This could be achieved by treating reconnected group members as latecomers and either sending them the current state of shared data or buffering and then transmitting the sequence of updates which were missed by the group member, whilst he or she was disconnected.

- ***The need for flexible concurrency control***

Greenberg and Marwood [Greenberg,94] argue the need to support flexible concurrency control when developing groupware applications. Such flexibility is required because the user is an active part of the collaborative process and their concurrency control requirements can change over time depending on the current task. They also argue that some conflicting interactions are best left to users to solve by social means, implying that some feedback of conflicting actions be shown within the interface. Another advocate for flexible concurrency control is Dourish [Dourish,96b] who introduces the notion of divergence within his Prospero toolkit in order to enable a flexible balance to be maintained between the consistency of data and its availability. In a mobile environment the importance of flexibility is increased further because the implications of choosing a particular method of concurrency control can vary as the underlying quality of group communications fluctuates. For this reason, it is vital that applications have the ability to select appropriate concurrency control policies that match not only the user's current task but also the quality of group communications.

In summary, it is apparent that many of the concepts and techniques fundamental to groupware are complicated when examined in a mobile context and therefore special consideration needs to be given to the development tools aimed at supporting mobile groupware. The following chapter describes a prototype mobile collaborative application designed to highlight the kind of problems that can occur when building an application designed for operation in a mobile environment using traditional groupware techniques.

REFERENCES:-

- **[Anker,97]** Anker, T., V. Gregory, D. Dolev and I. Keidar. "The Caelum Toolkit for CSCW: The Sky is the Limit.", *Proc. Third International Workshop on Next Generation Information Technologies and Systems (NGITS 97)*, June 30 - July 3, 1997, Neve Ilan, Israel.
- **[APM,89]** APM Ltd. "The ANSA Reference Manual Release 01.00.", Architecture Projects Management Ltd., Cambridge, U.K. 1989.
- **[APM,92]** APM Ltd. "An Introduction to ANSAware 4.0.", Architecture Projects Management Ltd., Cambridge, U.K. 1992.
- **[Brinck,92]** Brinck, T. and L.M. Gomez. "A Collaborative Medium for the Support of Conversational Props.", *Proc. ACM CSCW'92 Conference on Computer Supported Cooperative Work*, pages 171-178, Toronto, Canada, October 31-November 4 1992.
- **[Burridge,98]** Burridge, R. "Java Shared Data Toolkit User Guide.", *User Guide*, Version 1.4, Sun Microsystems Inc, June 1998.
- **[Casio,99]** Casio. "Casio Announces World's Smallest Multimedia Color Palm-Size PC.", *Press Release*, Casio Inc., http://www.casio.com/corporate/pressdetail.cfm?ID=60. January 1999.

- **[Chang,84]** Chang, J. and N. Maxemchuk. "Reliable Broadcast Protocols.", *ACM Transactions on Computer Systems*, Vol. 2, No. 3, pages 251-275, August 1984.
- **[Dourish,96b]** Dourish, P. "Consistency guarantees: Exploiting application semantics for consistency management in a collaboration toolkit.", *Proc. ACM CSCW'96 Conference on Computer Supported Cooperative Work*, pages 268-277, Boston, November 1996.
- **[Edwards,96]** Edwards, K., "Policies and Roles in Collaborative Applications.", *Proc. ACM CSCW'96 Conference on Computer Supported Cooperative Work*, Boston, November 16-20 1996.
- **[Ege,87]** Ege, A. and C. A. Ellis. "Design and Implementation of GORDION, an Object Base Management System.", *Proc. 3rd International Conference on Data Engineering*, pages 36-45, February 1987.
- **[Greenberg,91]** Greenberg, S. and R. Bohnet. "GroupSketch: A multi-user sketchpad for geographically-distributed small groups.", *Proc. Graphics Interface '91*, Calgary, Alberta, Canada. 1991.
- **[Greenberg,94]** Greenberg, S. and D. Marwood, "Real time groupware as a distributed system: Concurrency control and its effect on the interface.", *Proc. ACM CSCW'94 Conference on Computer Supported Cooperative Work*, pages 207-217, Chapel Hill, North Carolina, October 22-26 1994.
- **[Greenberg,96]** Greenberg, S. and M. Roseman. "Groupware Toolkits for Synchronous Work." *Research Report 96/589/09*, Department of Computer Science, University of Calgary, Calgary, Canada, November 1996.
- **[HP,99a]** Hewelet Packard, "HP Jornada handheld PCs.", http://www.hp.com/jornada/. 1999.
- **[HP,99b]** Hewelet Packard, "HP Omnibook 4100 Notebook PC - Data Sheet.", http://www.hp.com/omnibook/products/4100/datasheet.html. 1999.
- **[IEEE,97]** Institute of Electrical and Electronics Engineers. "Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications.", *IEEE Standards document,* 802.11-1997. ISBN 1-55937-935-9. June 1997.
- **[IrDA,99]** Infrared Data Association. "Technical Summary of IrDA DATA and IrDA CONTROL.", http://www.irda.org/standards/standards.asp. 1999.
- **[ISO,92]** International Standards Organisation. "Draft Recommendation X.901: Basic Reference Model of Open Distributed Processing - Part1: Overview and Guide to Use.", *Draft Report*, International Standards Organisation WG7 Commitee. November 1992.
- **[Lauwers,90]** Lauwers, J.C. and K.A. Lantz. "Collaboration awareness in support of collaboration transparency.", *Proc. ACM SIGCHI'90 Conference on Human Factors in Computing Systems*, pages 303-211, Seattle, Washington, April 1-5 1990.
- **[Leopold,91]** Leopold, R.J. "Low-earth orbit global cellular communications network.", *Proc. IEEE International Conference on Communications - ICC '91*, pages. 1108-1111. 1991.
- **[Microsoft,98]** Microsoft. "Distributed Component Object Model Protocol.", *Internet Draft Specification*,http://www.microsoft.com/oledev/olecom/draft-brown-dcom-v1-spec-02.txt. January 1998.
- **[Microsoft,99]** Microsoft, "Microsoft Windows CE", http://www.microsoft.com/windowsce/. 1999.