Engineering for Human-Computer Interaction, Chatty, S. and Dewan, P. Eds, February 1999, © Kluwer Academic Publishers.

External Requirements of Groupware Development Tools

Workshop Report

T.C. Nicholas Graham and John Grundy
Department of Computing and Information Science, Queen's University, Kingston, Ontario, Canada, K7L 3N6, graham@qucis.queensu.ca
Department of Computer Science, University of Waikato, Private Bag 3105, Hamilton, New Zealand, jgrundy@cs.waikato.ac.nz

Abstract: The EHCI'98 Workshop on Requirements of Groupware Development Tools examined six groupware applications in order to derive requirements for tools for developing groupware. We hope that these requirements will be useful to designers of new tools in motivating what features their tools should have.

Key words: Groupware Development Tools, Requirements

1. INTRODUCTION

Recent research in tools supporting the development of groupware applications has concentrated on two directions – making groupware easier to build by providing higher level programming abstractions, and increasing the range of applications that can be supported by groupware development tools. As examples, significant advances have been made in:

- *Support for flexible coupling*, allowing users to configure the granularity of their interaction with other users (Dewan, 1992; Grundy in this volume);
- *Support for versioning/merging,* allowing users to dynamically migrate between shared and private work (Munson, 1994; Edwards, 1997);
- *Support for group awareness,* such as the provision of awareness widgets in a toolkit (Gutwin, 1998);
- *Support for combining synchronous and asynchronous styles,* such as providing persistent rooms in which synchronous collaboration can occur, or combining web browsing with synchronous applications (Roseman, 1996; Graham, 1997);
- *Support for making existing applications into groupware,* recognizing that many existing applications cannot be rewritten as custom groupware (Begole, 1997);

- *Support for sound and video,* to allow multimedia to be used in a multiuser context (Dewan, 1992; Graham, 1997);
- *Support for workflow,* to aid in the coordination of groups (Grundy, 1998).

While significant advances have been made in the range of interaction styles supported by groupware development tools, little effort has been made to systematically relate this support to the requirements of actual groupware applications. The goal of this workshop was therefore to examine a representative set of groupware applications and to draw from them a set of requirements that groupware development tools should support. We have concentrated on the *external* requirements of groupware development tools, exploring the application functionality that the tools should support. An interesting further problem would be to consider the *internal* requirements of tools, considering how the tools support the development process.

In order to explore the external requirements of groupware development tools, we first described a set of six example groupware applications. These applications provide a wide range of interaction styles. From the applications, we drew a set of generic application features that groupware development tools should support. These features are structured using the Clover model (Calvary, 1997).

The report is organized as follows. Section 2 describes the six example groupware applications intended to motivate features of groupware. Section 3 summarizes the Clover model. Section 4 then presents the requirements of groupware development tools synthesized from the features of the six groupware applications.

2. APPLICATIONS

This section describes six applications illustrating features of modern groupware systems: a mediaspace, a visualization system, a virtual university, a metaCASE tool, a chess tutoring system and a software inspection tool. These applications cover a wide range of communication, coordination and media styles. Four of the six applications have been implemented while two are speculative, allowing features found in current groupware to be contrasted with features of future groupware systems.

2.1 CoMedi

CoMedi is a prototype mediaspace developed for exploring computer-mediated communication between the fifty members of a research laboratory (Coutaz *et al.* in this volume). The design of CoMedi is grounded on technical, functional and interaction requirements. CoMedi is implemented in Java to accommodate a wide range of hardware. The functional requirements were driven by a desire to support awareness, privacy, and scalability. The interaction requirements include that users should be able to perform frequent tasks with a minimum of explicit actions.

The functional and the interaction requirements result in a user interface based on the porthole metaphor enriched with an optional fisheye facility. To support privacy, CoMedi uses two orthogonal mechanisms: an accessibility matrix and the

published observability of private state variables. These variables express, for example, the user's level of availability and the video scene. The user can export private state variables to the members of his/her choice. If exported, private state variables can optionally be filtered. CoMedi provides filters for video scenes such as Venetian blinds, the shadow, temporal difference images, replacing one's image with a poster, and the eigen space filter. The accessibility matrix allows users to specify permissions, such as allowing a user to authorize every member of the mediaspace to contact him/her using the V-Phone or the Chat facilities. In addition, the published observability mechanism allows the user to export his/her private video scene to selected friends, possibly filtered through one of the privacy filters.

Although perceptual bandwidth may be unimportant for loosely coupled activities, it becomes vital for real time communication such as V-phone connections and tele-explorations. CoMedi proposes Fovea and a face tracker as two interaction techniques to alleviate visual discontinuity. The video image presented to the distant user is a composition of a high resolution fovea and a low resolution periphery. The fovea is provided by a high resolution steerable camera, while the periphery is given by a low resolution fixed camera. The fovea can be zoomed to provide the required level of detail.

2.2 The Manicoral Cooperative Visualization Tool

Manicoral is a prototype distributed cooperative visualization (DCV) tool. It allows geoscientists to work together on visualizations without having to share the same machine. The following scenario (Duce, 1998) illustrates use of the tool:

A geoscientist (A) is examining a dataset of sea surface height readings from a satellite-borne altimeter. He is interested in a specific part of the Mediterranean. The original track data has now been gridded and this is what A is visualizing. The visualization mapping method could be quite straightforward - a set of solid contours with some parameters controlling the range of data values with an active colour mapping. He has experience of other more widely known data and something does not look right.

He phones or emails a colleague (B) elsewhere in Europe and proposes that they engage in a computer mediated cooperative session. A informs B at the start of the session how to find the data and the visualization processing network. A sets the scene for B showing some pregenerated visualizations on the shared whiteboard. Now that each participant has the visualization system and all the software necessary for cooperation, A guides B through his reasoning with the help of shared control of colour maps via the DCV. B then begins to have some ideas about the cause of the problem and guides A through these in turn, reversing the roles.

B then explains to A that the problem is the choice of dataset and explains where an alternative can be found. At this stage both need to see both datasets in order to compare them. Ideally, the shared control parameters would govern the visualization of both datasets. This does not require additional shared control capability per se, but does require flexibility in how each shared parameter is linked to the visualization system - a flexibility which a dataflow system is able to provide.

B guides A through his reasoning and demonstrates that the new dataset solves the problem. As a result of the discussion a better understanding of the data has been obtained and an opportunity to publish a joint paper has been generated!

In the scenario a shared whiteboard is used to present prepared material, in this case snapshots of visualizations. The shared whiteboard can be used both as a presentation tool, where the presenter controls what "images" are shown, and as a workspace where non-standardized material can be shared. Interaction devices like telepointers and annotations are also available.

Geoscientists can share data at any point in the visualization pipeline, ranging from the raw data, the pre-processed data, the analyzed data, and the visualization itself, including the attributes controlling the visualization. The DCV tool, similarly to the whiteboard, allows the use of telepointers and annotations.

The work with the visualization tool needs to be supplemented by some kind of communication device (in the scenario phone/e-mail, but in the prototype this was achieved by using an existing audio and video communication tool).

Working with the visualization it is important to allow the sharing of existing data and the use of existing software. It is also important that the tools support flexible transition between local and collaborative work. In the DCV prototype, values may be local or shared in the session. Users may switch between local and shared values, and introduce new values or data sets that might later be shared.

Each researcher runs a copy of the visualization tool on his/her own machine. This can lead to problems if a researcher with a powerful machine experiments heavily with a shared value, causing slower machines to spend all available resources on keeping up with redrawing.

2.3 Virtual University

The Virtual University is a proposal to support remote real-time lectures (Cockton, 1998). Students attend class by opening a set of windows allowing them to view a lecture in progress. A video window may show the lecturer or other physical materials the lecturer chooses to show. Normally, video is broadcast in real time. However, students may enter a lecture up to ten minutes late, and review the video they missed in condensed form.

An application window contains prepared materials such as lecture slides. Students may view these materials in slaved mode, automatically following the lecturer, or may skip forwards or backwards privately. The application is pluggable, meaning that any existing application can be used.

Students may communicate with the lecturer by posing questions. Questions are typed off-line and sent to the lecturer, who may pause and respond at any time.

A set of *gestalt* views gives students an overview of the virtual lecture room. A question queue shows how many people wish to pose a question to the lecturer. Students may indicate their current level of comprehension. A mood view synthesizes the general level of comprehension in the room into a single image. A position view shows what point the lecturer has reached in his/her materials, allowing students to retain context when they are privately reviewing the materials.

The virtual university is flexible with respect to the students' hardware. As networking or machine performance degrades, the presentation of the lecture also degrades. For example, video images may be replaced with portholes, and eventually with still images. Live sound may be replaced with chat windows.

2.4 JComposer

The JComposer Object-Oriented Analysis and Design metaCASE tool (Grundy 1998 in this volume) supports flexible collaborative editing for OOA/D diagram views. JComposer has been built and used in an industrial setting. JComposer supports a wide range of collaborative editing styles. A collaboration menu allows users to specify which other users can collaboratively edit their views and the coupling level of these other users, ranging from asynchronous to fully synchronous editing. Asynchronous view editing allows users to independently modify private versions of a view, then exchange and incrementally merge sets of view edits. Presentation-level view editing distributes view edits to collaborators as they are made, and presents them as human-readable descriptions. Users can choose to incrementally merge selected changes into their version of the view. Synchronous view editing broadcasts edits as they are made and automatically merges them into the views of collaborators.

JComposer provides a range of awareness, coordination and communication facilities. Colouring of OOA/D iconic components indicates who last modified parts of a view. Human-readable change descriptions are annotated to indicate who made each view edit. Audio and/or chat facilities help to coordinate editing, especially when using the synchronous editing level. Email helps coordinate asynchronous editing. A workflow tool, Serendipity-II (Grundy et al 1998b), can be used with JComposer to coordinate or automate editing level usage, to automate notification of view edits, and to annotate change descriptions with workflow stage information.

2.5 Chess Collaborative Teaching Application

The Group for Interactive Tools and Applications (GHIA) at the Universidad Autónonoma de Madrid has specified a Chess tutoring application to motivate the requirements of teaching applications. The Chess application has not yet been built.

A chess tutoring application should allow interactive collaborative learning, reviewing, and analysis of chess rules and strategy. The application should allow one person to create a chess game or to review an existing one. When reviewing a game, a user should be able to explore alternative scenarios, by adding alternative moves and possible continuations from these moves. The resulting analysis would include attached notes and graphical comments, such as indications of weak and strong points and threatening aspects for the players. Both textual and graphical information could be created by authorized users and by an agent that has knowledge about relevant aspects of chess positions. Analysis would be available for other players to consult or further analyze, either individually or synchronously with others. Attached notes could become discussion threads on game issues. Users could show attached notes and graphical comments to other users while participating in a discussion.

In addition to discussions, the application should permit two players to play a game on distant screens or to continue an existing game where they left off. At any time they should be able to go back and analyze the game, either individually or collaboratively. The application should permit strong and weak synchronization.

Finally, a chess teacher should be able to access games as they are played or following their completion, analyze their history, and discuss them with any of the players. Agents may be used to advise the teacher of appropriate times to enter a game.

Users should also be able to filter chess analysis. Filtered analysis could be generated either by direct selection of the relevant snapshots or by automatic checking of positions and moves by agents that locate hot spots. Filtered analysis could be the basis for individual and synchronized review of chess games, and for interactive teaching.

2.6 Collaborative Software Inspector (CSI)

John Riedl's group at Minnesota (Mashayekhi, 1993) has developed several versions of a tool for supporting collaborative inspection. The first version supported Humphrey's inspection process, where a group of *reviewers* asynchronously inspects a document, preparing a list of faults. These are handed to a *producer*, who correlates the faults into an integrated list. A *moderator* then guides a synchronous meeting in discussing the integrated fault list. A *recorder* takes minutes of the meeting. Thus, the meeting consists of an asynchronous fault detection phase and a synchronous fault discussion phase.

The first version supported both the asynchronous and synchronous phases of the meeting. During the asynchronous phase, users made annotations to the lines in which they found faults, incrementally publicising their annotations. During the synchronous phase, they used the same tool, but this time coupled the scrollbars, mouse positions and all other aspects of their user interface so that they could have a shared discussion thread. Consensus was reached through a talk window and audio conferencing.

Their experiments showed that users wished to work asynchronously during the fault discussion phase. In particular, they wished to privately review the faults made by them or others, and to enter new faults. The tool supported private reviewing by allowing users to dynamically uncouple their windows from the moderator's window. To support asynchronous addition of new faults, the tool was extended to permit asynchronous voting and to provide discussion threads allowing parallel groups to discuss unrelated faults (Stein, 1997).

3. DEFINITIONS

The requirements presented in the next section are structured using the Clover model (Calvary, 1997). This model partitions the features of groupware applications into functions supporting *production*, *coordination* and *communication*.

The *production space* denotes shared artifacts that are collaboratively manipulated to perform some task. Example objects in the production space might include shared documents or drawings. Functions related to the production space include the viewing and manipulation of these shared artifacts.

The *coordination space* codifies the protocols governing how tasks are carried out by groups of people. Such protocols may be purely social (e.g., social rules

specify that only one person should talk at a time) or may be formally specified through a workflow system.

The *communication space* supports person-to-person communication. Email and mediaspaces are examples of systems designed for supporting computer-mediated communication, either asynchronously or synchronously.

4. EXTERNAL REQUIREMENTS

The example CSCW applications in section 2 have a range of "external", or end user requirements, in terms of the general groupware facilities these environments need to support. Thus any 4th generation groupware architecture should provide appropriate abstractions for building applications with these kinds of groupware features. In this section we summarise the wide variety of external requirements a 4th Generation groupware architecture should satisfy, using the Clover taxonomy. In addition, some extra technological requirements of groupware were determined relating to adaptability, integration and hardware issues.

4.1 Production Space Requirements

All groupware applications have some notion of data or information that has to be shared, exchanged and/or modified, as illustrated in Figure 1. The applications outlined in section 2 have a variety of requirements in how information is shared, viewed and edited, including flexible coupling of participants' views, flexible configuration of views and privacy settings, and support for production history.

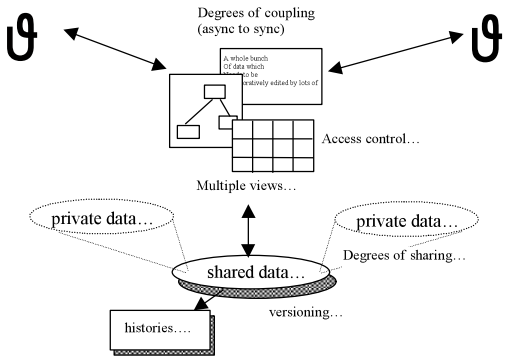

Figure 1. Production space continuum to be supported for groupware applications

4.1.1 Support for Flexible Coupling

The applications show that people work together at the same time or at different times, and that people smoothly move between these forms of work. For example, in the Virtual University, students may decouple from the lecture presentation, review material that was presented earlier in the lecture, and then later rejoin the

presentation. In CSI, JComposer and Manicoral, users may at any time make a version of shared data for private use, work alone on the private data, and later merge their results with the shared data. In the Chess application, players smoothly move between reviewing and playing games A groupware development tool is required to support the creation of applications where users can seamlessly move between synchronous and asynchronous work, and where users have control over the degree of coupling of views. More specifically, a groupware development tool should provide support for:

- *Seamless transition from shared to private use of data:* CoMedi, Manicoral, JComposer and CSI, require that information be shared to various degrees. For example, in Manicoral some physics data is local while other is shared, while some shared data is updateable, and some read-only. In CoMedi and Manicoral, users need to be able to dynamically control the degree of sharing.
- *Versioning and merging:* As asynchronous editing must be supported by some views, and some view data may be copied, edited independently and then merged with old data, versioning and merging of views and data must be supported by groupware development tools. For example, JComposer allows users to control versioning and merging of views. Manicoral also requires such dynamic versioning and merging, but for both views and viewed data. The chess learning program requires versioning of game play histories.
- *Support for conflict resolution:* JComposer and Manicoral require conflict detection and resolution during version merging. This should be integrated with both the merging system supported by a groupware development tool and the general syntactic/semantic constraint system used by the application.

4.1.2 Support for User Configurability

The applications show that it is important for users to be able to configure the behaviour of groupware. The last section discussed the importance of user control over coupling. In addition, users should be able to customize the appearance of their views of shared artifacts, and control how much private information they reveal to others. Groupware development tools should support:

- *Customizable views*. JComposer and Manicoral support multiple views of shared (and local) information. In Manicoral, viewing filters and display mechanisms can be defined by users rather than being hard-coded into the environment; in JComposer, users control what information is placed in views and how the information is laid out. View filtering, rendering, composition and layout information should itself be sharable, as well as the data actually being viewed.
- *Customizable access control*. Some users may have the ability to edit some data and/or views, while others may only view information or not have access to it at all. CoMedi, Manicoral, and the Virtual University have such requirements, and CoMedi and Manicoral require users to have some control over access control rights. Access control rights may be applied to kinds of data, subsets of data, views and/or parts of views. All should be supported by groupware development tools, with appropriate end user control mechanisms.

4.1.3 Support for History

Many groupware applications require a history of work and/or discussions to be maintained, including JComposer (history of view edits), CSI (history of discussion), and Chess (history of game play). History items should be treated in much the same way as other sharable data, as the histories in all of these environments may be edited and revised, annotated, undone/redone, filtered and versioned.

4.2 Coordination Space Requirements

There is a strong need in groupware systems to coordinate work, so that tasks done by different people are done in the correct order, peoples' work doesn't conflict, and negotiation and agreement is achieved in appropriate ways. The applications from section 2 support coordination in a wide variety of ways.

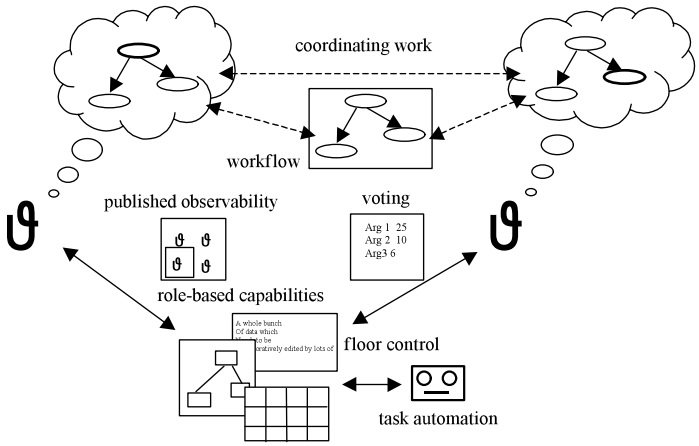

Figure 2. Coordination approaches to be supported for groupware applications.

For example, in CoMedi and Manicoral, users coordinate the degree of coupling between their views and plan transitions between them using social protocols. In JComposer, users may coordinate coupling and sharing using social protocols, or via an integrated workflow system which guides or enforces work. The workflow views and artifacts can be treated as elements in the production space, and thus provide different access rights, coupling levels and histories. In the chess learning system, coordination may be guided by agents which watch the progress of the students' games. Figure 2 illustrates the various kinds of coordination present in the applications of section 2.

Groupware development tools should support:

– *Role-based coordination.* Participants in groupware applications may have different capabilities depending on the roles they fill. The Virtual University has a strong distinction between lecturers and students, with these different classes of user having very different sets of views, view editing ability, access control

rights, and communication support. CSI differentiates between a moderator and reviewers, with the former having control over advancement of line inspection, controlling the overall inspection process. Chess differentiates between students and tutor, with the later able to review games, version the game history and suggest move changes.

- *Floor control*. Floor control coordinates production and communication aspects of groupware, giving one or more users control of audio/video channels, messaging, viewing and editing mechanisms, and coordination facilities. The Virtual University uses floor control to enable structured questions.
- *Task automation*. While cooperating users carry out many tasks, there is often a need for some automation in groupware environments. The Chess learning system requires the tutor to be able to specify a variety of notification and automatic annotation tasks, such as informing them when checkmate moves are missed via messaging and/or annotation. Agents may help in these tasks, and in notifying the instructor when students may require his/her attention. JComposer allows users to specify notification and simple task automation agents using a visual language, which can be enabled on the fly.
- *Published observability.* CoMedi allows users to view and hear other users and their offices. Users need control over what parts can be viewed and in what ways. Manicoral also needs to coordinate viewing of data, with some views at times invisible to others and at other times visible. In CoMedi, users publish the actions that they permit other users to perform. This published observability information allows users to coordinate what actions they may perform on another users data.
- *Voting.* Some applications require a formalised mechanism for reaching agreement. CSI uses voting to reach a consensus on whether or not a line has errors and on what action to take. Development tools should support facilities for negotiation and reaching agreement.

4.3 Communication and Awareness Requirements

In addition to the communication that is inherent through the manipulation of shared artifacts, groupware applications need to facilitate direct communication between distributed people. Communication techniques range over synchronous channels such as sound and video to asynchronous channels such notes and annotations. Figure 3 illustrates these communication mechanisms. Groupware development tools need to support communication through person-to-person messaging, annotations, and specialized awareness views.

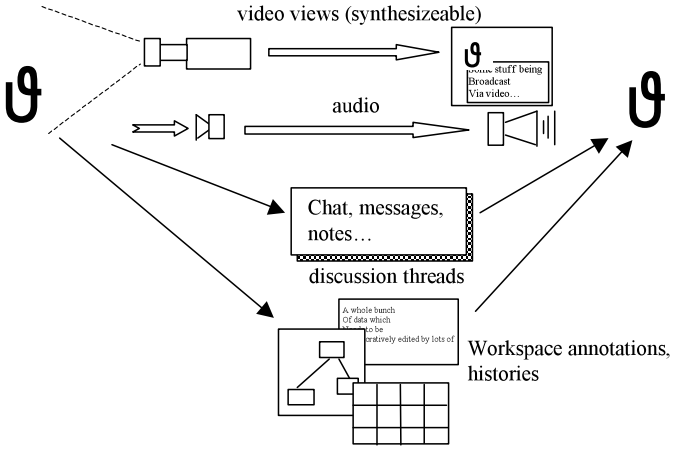

Figure 3. Communication approaches to be supported for groupware applications.

4.3.1 Synchronous and Asynchronous Messaging

The most fundamental form of communication is through messages sent from one person to one or more others. E-mail asynchronous messaging and IRC-style synchronous messaging are two of the most common text-based messaging facilities available in groupware applications. All of the applications in section 2 make use of such messaging facilities, and some, such as Manicoral, JComposer and CSI assume users can easily switch from one mode to the other. In addition to these traditional mechanisms, groupware development tools should support:

- *Controllable video views*. Communication via video may not only include people's faces, but also views on whiteboards, office space, etc. For example, CoMedi provides a variety of video communications, including portholes indicating the presence of others, fovea views, and automated camera tracking based on blink detection.
- *Audio communication*. Many applications utilise audio communication, providing a synchronous, real-time chat system. Audio can also be used in voice-mail systems to provide asynchronous messages. CoMedi, the Virtual University, Manicoral and JComposer all utilize synchronous audio communications. The Virtual University can also use recorded speech for replying in lectures and voice-based annotation of slides for students.

4.3.2 Annotation

Annotation is a useful asynchronous communication system which may take the form of textual "sticky notes", graphical items overlaying artefact views and history and message item textual annotation. Manicoral uses textual annotation of data views; the Virtual University textual and graphical annotations of slides and messages; JComposer uses textual annotation of editing histories and graphical and textual annotation of diagram views; CSI annotation of discussion threads and code

lines; and Chess annotation of game histories. Tools for developing groupware should support annotation of all media and artifacts.

4.3.3 Gestalt Views

To collaborate effectively with a group, people require knowledge of the activities of other group members. A number of the applications made use of views synthesizing information about the group or its members to aid in communication. The virtual university uses several of these *gestalt* views: a timeline view showing the current location in the lecture, a question queue showing how many questions have been posed, and a mood view showing the general level of understanding of the class. CSI provides a view showing the results of the voting so far. Gestalt views synthesize information, both to provide a quick mechanism for summarizing information useful to group interaction, and as a mechanism for abstracting information that may be private. For example, the virtual university's mood view does not reveal the mood of individual students; CSI's voting view need not reveal the individual people's votes. A groupware development tool should support the easy creation of gestalt views summarizing information.

CoMedi provides synthesized views based on sophisticated image processing, leading to more demanding requirements. For example, CoMedi provides gestalt views showing who is available in their office, synthesized facial images filtering out non-facial data.

4.4 Technological Requirements

In addition to the requirements relating strictly to the functionality of groupware, the applications suggested a variety of technological requirements related to the use of available hardware, networks and software available to groupware applications. Groupware development tools should support:

- *Resource adaptivity:* At times groupware applications can be run by users with very different hardware and resources available. For example, it would be useful for CoMedi to be used with one user with a high-end workstation and high-resolution video camera, and another user with a Palm-top with limited CPU power and I/O devices. The Virtual University has a lecturer with high-end workstation and I/O devices, some students with similar hardware and networking, and others with low-end PCs and modem connections. JComposer is often used by groups with one user on a PC with a fast LAN, and another a slow modem connection. Thus groupware toolkits should facilitate applications adapting to variable hardware in graceful ways.
- *User preferences:* Often users have different preferences as to what hardware resources should be used. For example, some Virtual University students want full-motion video and rich audio, while others just want low-resolution audio and sampled video stills. Groupware tools should allow such user requirements to be handled in seamless ways, and to be easily configured by end users.
- *Reusability of existing applications:* It is often far too difficult to replicate commercial software applications in order to make them group aware. Most

groupware systems require existing applications to be integrated with their capabilities in appropriate ways. For example, Manicoral should use existing database, spreadsheet and visualisation software; the Virtual University uses an existing pluggable application; JComposer uses existing workflow and programming environments; and CSI uses existing compilers and debuggers.

- *Network state reporting:* Often it is important for users to be aware of whether or not others are seeing/hearing them and/or their modifications of work artifacts or messages and annotations. The Virtual University lecturer wants to be aware of the number of students engaged in the lecture and when students arrive/leave. JComposer users need to be informed when collaborators loose their connections or log onto a session.
- *Fault-tolerance:* All groupware systems leverage network connections and computer hardware which can fail unpredictably. Thus all need mechanisms for recovery from people going off-line or rejoining cooperative work sessions. Manicoral needs shared data being modified to be kept "safe" from failure-induced corruption; the Virtual University needs to accommodate students who loose their connection during a lecture and rejoin, or who arrive late or leave early; and JComposer needs to support "lost" view edits and rejoining of users in asynchronous editing mode.

5. CONCLUSION

This report has summarized the conclusions of the EHCI Workshop on Requirements of Groupware Development Tools. We have outlined the general external requirements of groupware development tools, as driven by the features of the applications surveyed in section 2. We do not claim to have a complete coverage of all possible external requirements of groupware systems, but as illustrated above, many applications exhibit common requirements. Thus the workshop participants believe that developers of new CSCW architectures should endeavour to address all of the requirements outlined in this section, or at least ensure their architectures and implementations can be extended to accommodate them.

ACKNOWLEDGMENTS

This report is the result of a workshop held at EHCI'98. The report was edited from written contributions by Joëlle Coutaz, Prasun Dewan, Morten Borup Harning, Roberto Moriyon and the authors. Other participants contributing to the report were Remi Bastide, Patrick Girard, Jocelyne Nanard, Philippe Palanque, Fabio Paterno, Franck Tarpin-Bernard and Claus Unger.

REFERENCES

Begole, J., Struble, C.A., Shaffer, C.A. and Smith, R.B. Transparent Sharing of Java Applets: A Replicated Approach. In *Proc. ACM UIST '97,* pages 55-64. ACM Press, 1997.

Calvary, G., Coutaz, J. and Nigay, L., From Single-User Architectural Design to PAC*: a Generic Software Architecture Model for CSCW. In *Proc. CHI '97,* pages 242-249, 1997.

Cockton, G. *IFIP Virtual University Case Study.* http://osiris.sund.ac.uk/~cs0gco/IFIP/ifip_index.htm, 1998.

P. Dewan and R. Choudhary. A High-Level and Flexible Framework for Implementing Multiuser User Interfaces. *ACM TOIS,* 10(4):345-380, October 1992.

Duce, D.A., Gallop, J.R., Johnson, I.J., Robinson, K., Seelig, C.D., and Cooper, C.S. Distributed Cooperative Visualization - The MANICORAL Approach. In *Proc. Eurographics UK*, pp. 69-85, 1998.

T.C.N. Graham. GroupScape: Integrating Synchronous Groupware and the World Wide Web. In *Proc. INTERACT '97,* pages 547-554. Chapman and Hall, 1997.

T.C.N. Graham and T. Urnes. Integrating Support for Temporal Media into an Architecture for Graphical User Interfaces. In *Proc. ICSE '97.* IEEE Computer Society Press, 1997.

Grundy, J.C., Hosking, J.G., and Mugridge, W.B., Visual Specification of Multi-View Visual Environments, In *Proc. VL'98,* Halifax, Canada, Sept 4-7, IEEE CS Press, 1998.

Grundy, J.C., Hosking, J.G., Mugridge, W.B., Apperley, M.D. An architecture and environment for decentralised, internet-wide software process modelling and enactment. *IEEE Internet Computing* 2(5), IEEE CS Press, September/November, 1998.

Gutwin, C. and Greenberg, S. Effects of Awareness Support on Groupware Usability. In *Proc. CHI'98,* ACM Press, pp. 511-518, 1998.

Mashayekhi, V., Drake, J., Tsai, W.T. and Riedl, J. Distributed Collaborative Software Inspection. *IEEE Software,* pp. 66-75, Sept. 1993.

Munson, J., Dewan, P. A Flexible Object Merging Framework, *ACM CSCW,* pp. 231-242, Oct. 1994.

Roseman, M. and Greenberg, S. TeamRooms: Network Places for Collaboration. In Proc. *ACM CSCW,* pp. 325-333, 1996.

M. V. Stein, J. T. Riedl, S. J. Harner, and V. Mashayekhi, A Case Study of Distributed, Asynchronous Software Inspection, In *Proc. ICSE'97,* May 1997.