CS 290: Collaborative Systems [1]
Section 2: Collaborative Applications
Prasun Dewan

1. COLLABORATIVE APPLICATIONS

As we saw earlier, the goal of CSCW is to provide better support for collaboration than what is provided by traditional systems. Current support for collaboration consists basically of talk, mail, and files and databases. To understand what else is needed, let us compare these applications, identifying properties that distinguish them from each other.

A talk application supports the abstraction of a joint *session* which users may enter or leave. Text entered by a user user does not have to targeted at specific users, other users in the session are the implicit receivers of it. Moreover, the application makes a "best effort" at communicating user's input as soon as possible, keeping active connections among all users. As a result, it is well suited to support distributed synchronous meetings. In contrast, mail has no notion of a session, requiring *message-specific destinations*. Moreover, it does not maintain active connections, and can queue and batch messages to decrease network traffic. Thus, it is meant for asynchronous collaboration, though it is possible to use it (albeit clumsily) for synchronous collaboration. In both a talk and mail system, information is explicitly copied from between the (potentially) private spaces of the collaborators. A file or database, on the other hand, creates a *single shared (logical) copy* of some *artifact*, and communication occurs indirectly by monitoring the state of this object. Thus, a file is suitable for collaboration involving the joint creation of a shared artifact.

Thus, these three systems defined three classes of applications, *session-based*, *message-based*, and *artifact-based* applications. We can improve on the state of the art by (a) providing better support for each class, while retaining the basic properties of the class, and (b) combining properties of these classes, thereby creating new, hybrid, classes of collaborative applications. Let us look first at applications that provide better support for each class and later at those that combine properties of these classes. This classification is meant mainly to be a tool for organizing the tour of collaborative applications. It is not meant to be taken strictly, since many applications combine properties of multiple classes. The class in which we introduce an application depends on how it contributes to the flow of the discussion. Finally, this is only way to classify collaborative applications, allowing us to derive novel collaborative applications based on features of familiar applications. Later, we shall look at other classifications of these applications.

In the description of each application, we list new features that had not been presented before. It is important to note that these features are "new" based only on what had been described before, and are not necessarily the research contributions of that application, since we are not presenting applications in chronological order or even describing all details of each application.

1.1 Session-based Collaboration (Talk++)

1.1.1 *MUDs: Textual Virtual Reality .* Talk is an effective poor man's tool to support the principle of "being there". MUD (Multiuser Dungeons) [?] shows how it is possible to provide more sophisticated support for this principle by creating a virtual environment that simulates a physical meeting environment. Unlike several other applications we shall see later, the original intention was not to support predefined teams and goals but instead to provide a meeting ground for users unaware of each other who could play games, socialise, or so do some actual work. MUD provides the abstraction of rooms and allows users to enter and leave them. It offers each user two basic commands, `say` and `emote.` To illustrate, let us take the example from [?]: If user Munchkin, executes:

```
say Can anyone hear me?
```

he sees the echo

```
You say, ``Can anyone hear me''
```

while others in the room see:

```
Munchkin says, ``Can anyone hear me?''
```

If Munchkin executes

```
emote smiles
```

then everyone else in the room sees

```
Munchkin smiles
```

A variation of the `say` command, the `whisper` command, allows a user to direct a comment to a specific user. Thus, if Munchkin types,

```
whisper ``I wish he'd just go away ..'' to Frebble
```

then only Frebble sees the comment. Thus MUD is also example of a tool that integrates talk and mail, which we shall study in more depth later.

When a user enters or leaves a room or leaves/joins a conference, messages are printed on the screens of other users in the room. The name and gender used to identify users are those they explicitly told MUD about. Typically, users do not use their real name and sometimes not their real gender. A user also gives MUD a (typically fictitious) textual description, which others can examine by executing the `look` command. A special `@who` command tells a user the names, connect time, idle time, and location of the users connected to MUD.

MUDs have special users called "wizards" who correspond to superusers. An ordinary player can be transformed into a wizard by another wizard. Wizards can punish other users for socially unacceptable behavior. They can prevent users from connecting or executing certain commands, changing their names and descriptions to unpleasant ones, or "humiliate" them by moving them to a public place.

MUD is extensible and allows users to add their own objects to the database such as rooms, exits, and notes. Thus, the MUD idea also combines artifact-based collaboration with session-management.

Several variations and extensions of the MUD idea are currently being pursued [?].

1.1.2 *DIVE: 3-D VR .* MUD is an example of a textual "VR" application. The DIVE (Distributed Interactive Virtual Environment) [?] system shows how it is possible to provide a more graphical VR for meeting support. A user or application is represented by an icon, which can move in a 3D space. The icon is associated with a space around it called the "aura". The notion of aura is used to establish communication between agents (users or applications) - the communication is established between two parties when the auras of their icons intersect. Communicating with applications means enabling them and with users mean perceiving them (talking and seeing.) Communication establishment is transitive, so if A's aura intersects B's aura, and B's aura intersects C's aura, then A and C can communicate with each other. Auras may be typed, and transitivity applies only to auras of the same type. Transitivity essentially allows auras to be amplified.

Several DIVE tools use this concept:

—Talk: When the auras of the icons of two users intersects, voice talk is enabled.

—Conference table: The conference table application also has an icon and two auras - an aura of perception and an aura of distribution. Users whose auras intersect the aura of perception, can talk to each other (even though their auras may not intersect directly) Users whose auras intersect with the aura of distribution get distributed active (shared) copies of documents placed on the table. Other users get private copies of the document.

—Podium service: A podium has two auras, an aura of perception and an aura of communication. A user within the aura of perception can speak to the audience in the aura of communication. Members of the audience cannot speak to each other or the speaker unless their auras intersect each other (directly or indirectly.)

—Single-user document editor: This is a single-user application and has an icon and aura. The application is enabled when a user's aura intersects this aura.

—Multiuser whiteboard: Each whiteboard has an aura. Users can interact with the whiteboard and talk to each other when their auras intersect the Whiteboard aura. Thus DIVE is also an example of a tool that integrates session- and artifact-based collaboration, which we shall study in more detail later.

Currently, I believe DIVE is a non-immersive VR application. It is easy to imagine an immersive extension, that allows a user to use a head-mounted display.

1.1.3 *Video Walls.* Both MUD and DIVE create synthetic environments that have properties of a physical meeting environment. The synthetic environment, however, is independent of the actual physical environments of the users in the session, thus not quite giving the illusion of "being there". We will now see a series of applications that address this problem.

A simple solution is provided by Video walls [?], which provides two-way audio and video connections between two remote rooms. Each room has a "video wall," which shows the activities in the other room, thereby allowing the users in the two

rooms to interact with each other. A camera at each site sends video to a large projection monitor which displays the image. The audio endpoints are speakers and microphones that cover each room. Experience at Xerox shows that is is possible to use current technology to establish usable video walls connecting two geographically dispersed sites.

1.1.4 *Audio and Video Conferencing.* Audio and video-conferencing (of the kind provided by See You See Me) is an extension of video walls allowing more that two users in a session to communicate with each other. Users see videos of their collaborators in separate video windows and can talk to them using audio channels.

1.1.5 *Media Space.* A media space allows users to "walk" to distributed rooms, thereby combining video conferencing with the MUD paradigm of navigating through rooms. Images captured by cameras in the rooms of participants in this space are switched by server software executing on one of the workstations. Authorized participants can ask the server to bring up images of remote rooms on their workstations, thereby becoming aware of their activities, much as a user is aware of the activities of a colocated collaborator whose office door is open. This is useful, for instance, when a user wishes to determine is a collaborator can be interrupted.

1.1.6 *Portholes: Quasi-Dynamic Media Space.* In comparison to web home pages with static pictures, a media space is more dynamic, allowing users to receive uptodate information about their collaborators. The concept of Portholes [?], shows that there is an intermediate ground between static images and dynamic video: instead of sending images continuously, one can send them every few minutes. Nynex Portholes [?] extend this idea by storing an hour's worth of images and letting users animate them. Moreover, it allows the user being animated to censor out certain periods of the animation.

1.1.7 *Hydra: Gaze Awareness.* Traditional videoconferencing systems, while allowing users to view remote collaborators, do not support "gaze awareness," that is, do not allow a user to make eye contact with a specific collaborator. When a user looks at the camera, it is not clear which collaborator he is directing his communication to. Hydra [?] provides special devices to address this problem, each of which has a camera and a display. Each remote collaborator is assigned a Hydra device which displays his image, and the camera in that device captures the local image transmitted to him. Thus, when the user turns to a device, all collaborators know who he is looking at. A single microphone captures the local audio transmitted to all users and all remote audio streams are combined into one stream played through a single speaker.

Hydra, thus, better creates the illusion of "being there," but still comes up a bit short in some dimensions. The hydra devices are small, therefore users see miniature images of their collaborators. Moreover, since the devices are independent with distinct boundaries, the system does not create the illusion of the collaborators

being located in one room. Finally, the single microphone and speaker does not allow audio directionality.

1.1.8 *MAJIC: Seamless User Boundaries .* The MAJIC system [?] extends the Hydra concept to address these limitations by by giving the users in a session the illusion of sitting around in a round table projected on a special curved, transparent screen. Each collaborator is assigned an arc in the screen onto which his image is projected. Behind that arc is a camera and a microphone, which capture the image/audio transmitted to him, and a speaker, which plays his audio. An overlaid background ensures that users do not see seams between the images of their collaborators.

1.1.9 *GestureCam: Remote Surrogate.* The cameras in MAJIC and Hydra cannot be controlled by remote users. As a result, a user cannot point to or look at arbitrary objects in the room of a remote collaborator. GestureCam [?] illustrates how this problem may be overcome. For each remote collaborator (currently the systems supports only one) it places a surrogate device at the local site that can be controlled by the remote user. The device has three degrees of freedom, carries a camera and a laser pointer, which points in the same direction as the camera. The remote collaborator can move the camera in three dimensions to view and point at different objects in the room. Two different user interfaces are provided to allow a user to control a remote surrogate. First, a physical replica of the surrogate (without the camera and pointer) is placed at the local site, which can be turned and moved to invoke corresponding actions in the remote robot. Second, an image of the remote room captured by a wide-area lens is projected onto a touch-sensitive screen. When a user touches an object displayed in the screen, the remote surrogate automatically points towards it. The system has been used to instruct operators how to assemble a breadboard circuit, connect two portable computers, and share files between two computers.

1.1.10 *Telepresence.* The systems come close to but do not quite support the illusion of being at a remote location. In particular, a user is not immersed in the remote environment and cannot physically manipulate remote objects, which would be required, for instance, in remote surgery. This is the goal of the telepresence project at UNC, which will use a "sea of cameras" to dynamically create a remote environment as a virtual environment at the local site.

1.1.11 *3-D Simulation.* A shared space need not be synthetic or based on the physical environments of the collaborators - it could instead be a simulation of some physical object that multiple users need to manipulate or walkthrough. This idea is used in Shasthra [?], Stotts's VR work [?], and in Bechtel's design review application. The former provides users with a non-immersive user interface to jointly manipulate 3-D objects such as automobile designs, each user seeing the changes made by the other. The other two allow multiple users to independently walkthrough virtual environments such as a ship or building design, seeing icons representing the view points of other users.

1.1.12 *Internet Foyer: Mixed Reality.* We have seen above two kinds of systems: some constructing a synthetic space in which users meet and others providing a

virtual space based on a physical environment. The internet foyer [?] show how the two paradigms can be combined in a mixed reality allowing one to go "beyond being there".

The system tries to unite the concept of a physical foyer with an electronic foyer. A foyer can be considered the public region in some space that informs visitors about what is available in that space and guards access to private regions in the space. A physical foyer is simply a foyer in a physical space (building) while a virtual foyer is a foyer in some electronic space. An example of a virtual foyer is a set of public web pages of an organization, which advertise the information about the organization available on the internet and provide password-protected mechanisms to access this information. The same organization may have both a physical foyer in a building and a virtual foyer in the web, and the internet foyer shows how these concepts can be united.

The system supports three classes of users:

—*physical-foyer visitors*: those situated in the foyer of a physical building,

—*virtual-foyer visitors*: those using a VR application to view parts of the web rendered in a 3-D space,

—*regular web-users*: those using a slightly modified standard web browser to navigate the space.

Visitors in the physical foyer see a projected image of the 3-D web space with the locations and images of the users of the virtual-foyer visitors. Visitors in the virtual foyer can also see this space and also a window in the space that shows an image of the physical foyer. They can navigate through this space, zoom in on a node on the space, and select a web node for browsing. When they zoom in on a node, they can see 2-D representations of regular web-users who have opened the node using the modified browser.

1.2 Message-based Collaboration (Mail++)

As we earlier, a message-based system copies messages between the potentially private spaces of the sender and receiver, and allows message-specific recipients. We discuss here some systems with these properties that provide novel support missing in traditional mail.

1.2.1 *Information Lens: Typed Messages.* We often create different kinds of messages such as exam change notices, classroom change notices, bug reports, old exams, assignment solutions, requests for class absences, requests for postponement of exams, requests for placing papers/books in the library, and so on. However, traditional mail supports completely untyped and unstructured mail, and thus cannot capture the differences between different kinds of messages. Information Lens [?] overcomes this problem by supporting typed messages, allowing users to define these types, and arranging the types in a type hierarchy.

In general, typing a message offers several benefits: First, a type defines a structure that is common to all its instances, which can be created automatically by the system. For instance, an exam change notice would have fields for new and old dates, which can be automatically created by the system. Messages in Information

Lens are considered semi-structured since they are associated with text fields that can have arbitrary contents. Second, a type can be used as a basis for selecting messages from a mail box. For instance, we can ask the mail system to show all "notices" or all "exam notices". Third, if we want to exchange objects created by other applications such as spreadsheets and wordprocessors, then we are not forced to convert between their representations and the textual representation understood by traditional mail systems. Finally, the mail system can directly display multiple representations of these objects, as illustrated by contemporary mail systems supporting MIME attachments.

1.2.2 *Coordinator/Action Workflow: Structured Conversations.* We can associate a well-defined, repeated, structure not only with the contents of a message but also our conversations. For instance, if I send you a message requesting a book, then I expect a positive or negative response. If a response does not arrive after a while, I will probably send you a reminder message. The Coordinator [?] shows how conversation structure can be captured in the mail system. It is based on a general theory of conversations called the "speech-act" theory. A successor to the Coordinator, ActionworkFlow has been presented as a workflow system – a system that automatically routes objects among team members and performs user-defined actions on these objects. Let us look at ActionWorkflow to understand the "speech-act" theory and its potential benefits.

The application assumes that flow of information among users is a set of interrelated conversations, where each conversation consists of:

—*Proposal* - a request made by a customer to a producer for the completion of some action.

—*Agreement* - declaration of conditions of satisfaction.

—*Performance* - declaration by the producer of the completion of the service.

—*Satisfaction* - acceptance by the customer of service completion.

Winograd argues that this model allows one to view a network as a collection of links with shared nodes rather than a collection of nodes with links in between. ActionWorkFlow provides the following computer support for such conversations:

—notifying users about actions that need completion,

—providing forms etc to allow users to initiate tasks and communicate with each other,

—generating alerts, reminders, follow-ups etc,

—displaying different views of the conversation status to customers, producers, managers etc.

—automating standard procedures and individualized responses.

The ActionWorkFlow paper presents an example workflow application built using the system. The task is to coordinate the interview of some candidate. The personnel directory makes a proposal to the personnel manager to manage the review. The manager starts the main "manage candidate review" conversation/workflow by filling a form that states the name of the candidate, position, skill required, and the interviewers. The system automatically starts the workflow/conversation, "schedule interview" to schedule the interview. The completion of the scheduling of the

interview triggers agreement by the manager to do the candidate. The agreement, in turn, starts, for each interviewer, a "submit evaluation form" workflow, where the consumer is the manager and the producer the interviewer and the forms are electronic.

An interviewer may be responsible for evaluating several candidates, and can ask the system to show the name, recommended action (schedule interview, check status of evaluation, recommit to evaluate), and the date by when the action is to be completed. Selecting a candidate displays a form for the candidate, which can be filled and then submitted to declare performance of the action. Reminders are sent to interviewers if they do not fill the form in time.

The receipt of all workflow forms triggers completion of the original "manage candidate review workflow", which in turn triggers a "declare assessment" workflow in which the manager is the customer and the director is the customer. The director is sent a mail message regarding this workflow.

The system also allows a manager to see the status of the workflows by listing for each workflow loop, which candidates have not been processed, have interviews scheduled, have completed evaluations, and so on.

The speech-act theory is only one model on which workflow may be based. As pointed out in [?], it is suitable for non-procedural, unstructured work but is too heavyweight for structured, repeated processes wherein agreements do not have to be negotiated for each conversation. We shall see later an example of a simpler workflow system, POLITeam [?], which provides a simpler workflow model based on the idea of a routing slip, and also integrates message and artifact-based collaboration.

1.2.3 *Computational Mail.* Computational mail [?] is a concept similar to workflow in that it associates mail operations with user-defined actions. In computational mail, one can not only mail data but also computations or executable programs, and the receipt of a computation message triggers the activation of the mailed program. The program can be used to display arbitrary information to the user or, perhaps more interestingly, gather information from the user and mails it back the sender. To illustrate its use, suppose an instructor is interested in gathering from the students their preferences regarding exam times. He can send computational mail to the students that triggers a program that gathers their preferences and mails them back to the instructor.

The program executed on the receipt of a computation mail message is similar to a Java applet executed in response to the downloading of a web page in a browser. Thus, computation mail, like Java, must support portability and security. ATOMICMAIL is an implementation of this concept that addresses these issues [?]. It offers a Lisp-based programming language and the Unix text-based Curses package for portability, and imposes several restrictions for security: It restricts file access of the program to a single subdirectory of the receiver's file system, puts bounds on the number and size of the files created by the program and the mail messages sent by it, does not allow overwriting of existing files, and allows output of only printable characters. Thus, it seems to be less paranoid about security than Java, allowing the writing of more useful programs.

1.3 Artifact-based Collaboration (File++)

These are systems that allow collaborators to share some logical object/artifact and provide specialized support for collaboration that is missing from traditional file and database systems. They include not only a back-end, storage manager but also a front-end, user-interface agent to access the stored objects, which corresponds to a command-language/query interpreter in file/database systems.

1.3.1 *Hypermedia.* The success of the Web has shown the various ways in which hyperlinks can be useful in distributed collaboration. For instance, they make it easy to associate with an artifact the design rationale for and comments on it.

1.3.2 *Quilt: Writeable Hypermedia and Typed Links.* The Web supports untyped hyperlinks and does not allow modification of the artifact. Quilt [?] is an example of a system that supports typed hyperlinks and modifiable, hyperlinked objects. It has been designed to support collaborative writing of a document, but users are expected to serialize their access to the shared data.

A Quilt document has a tree structure consisting of a root base node and annotation nodes. It supports several types of textual/audio annotations including revisions, suggestions, public comments, and directed or private messages and allows users to add new annotation types. Annotations are associated with attributes, which can be predefined (such as creator and creation time) or user-defined. Attributes can be used in queries requesting different views of the document.

Quilt supports several roles such as co-author and commenter, each of which defines a set of permissions. Each user takes a particular role and is given the access rights associated with the role. The set of roles users can choose from and the permissions associated with them are determined by the collaboration style chosen by them. Quilt supports a set of predefined collaboration styles and allows users to extend this set. An example of a predefined collaboration style is the shared style, which defines three roles, Reader, Commenter, and Co-Author; puts these roles in the following hierarchy: A Commenter has all the rights of a Reader, and a Co-author has all the rights of a Commenter; and associates each role with permissions such as a Co-Author can create a base document, a Commenter can attach a comment to it, and a Reader can attach a private comment to it.

Quilt logs user actions at both a primitive, system-defined level and a more abstract, user-defined level. The user-defined level is generated in response to abstract statements explicitly made by users such as "Reorganized section 2" and "I am thinking about a new data analysis for this section ..." [?]. It supports per-user electronic "mailboxes" by allowing users to see annotations directed to them without directly accessing the document. Like active databases [?], it allows users to associate user-defined actions with certain conditions. For instance, users can ask Quilt to send them messages when their partners make "substantial" changes to a document or to start nagging them as the document deadline approaches. Thus, Quilt is also an example of a system that combines message- and artifact- based collaboration.

1.3.3 *PREP: Zero-Cost Hyperlinks.* In hypermedia systems, the cost of traversing a link is low but not zero. The PREP database/editor [?] illustrates how this cost can eliminated. The database stores chunks of textual/graphical data and allows users to group these chunks into drafts. Each draft corresponds to some shared document that the users are collaborating on. The editor allows the chunks in a draft to be displayed in a variable number of variable-sized, named columns. Typically, one column contains the contents of the document and the other columns contain comments input by different users about the columns to their left. Moreover, typically, the contents column is shared by the users but not the comments columns, each of which contains the comments made by a particular user. Thus, reading left to right, one sees how the discussion about a part of the document progressed; and top to bottom one sees the comments input by a particular user about the different parts of the document. PREP can be considered a special case of a hypertext system in that a comment on some chunk can be considered a node linked to the latter. Unlike traditional hypertext systems, PREP requires no user effort to traverse hyperlinks (since the comments are always visible), a feature hypertext users desire [?]

Several versions of PREP have been built, this discussion applies to the initial version. We shall study features of subsequent versions later, under the issues of diffing and coupling.

1.3.4 *IBIS: Structured Discussion.* PREP columns provide only one way to organize the discussion of an issue. IBIS (Issue-based Information System) [?] is a more sophisticated method to capture the rationale behind decisions. It formally supports the notions of issues, positions about these issues, arguments for positions, and captures the relationships among them. IBIS is a textual tool that displays these relationships using indented text. The following example from [?] illustrates how IBIS displays these relationships:

```
*I: Which processor should be used?
      ?P: Processor A.
             AS: Fast
      *P: Processor B.
             AS: Already in use, thus cheaper.
      -P: Processor C.
             AO: Won't be available in time.
```

The keyword I indicates an issue, P a position, AS a supporting argument, AO an opposing argument, - a rejected position, * an issue that has been resolved or a position that has been taken, and ? a positions about which no decision has been taken. A graphical version, called gIBIS, displays these relationships graphically. Another variation, rIBIS [?], allows real-time argumentation, thereby, like other applications discussed below, combining session- and artifact- based collaboration.

1.3.5 *CLARE: Structured Discussion + Process Model.* Based on issue-based argumentation, IBIS is only way to structure a discussion. Designed for collaborative learning of a research paper, CLARE (Collaborative Research And Research Environment) offers an even more structure for organizing a discussion and integrates it with a process model. Thus, it consists of two components, RESRA, which defines the discussion structure, and SECAI, which defines the process model.

RESRA (REpresentation Schema of Research Artifacts) defines several kinds of linked nodes including research papers or *sources*, which address one or more *problems*. *In response to* a problem, a *claim* may be made regarding how it may be solved, which, in turn, may be supported by *evidence*, which are generated by research *methods*, which, in turn, define *concepts*. All of these nodes and the relationships among them can be associated with *critiques*. (What are the analogues of these in IBIS?)

SECAI(Summarization, Evaluation, Comparison, Argumentation, and Integration) defines five steps in the discussion. The first two steps, summarization and evaluation are carried out privately, while the next three steps, comparison, argumentation, and integration, involve others. During summarization and evaluation, a user develops a representation of the paper and critiques it, respectively. In the comparison public step, the different users compare the nodes and links of their representations. Next, during argumentation, they critique each others' representations. Finally, in the integration phase, each user links his representation with other representations to form a collective representation of the paper. The system provides special support for these phases. For instance, during comparison, it shows in one window the union of the problems identified by all users; and also gives the number of instances of each kind of node created by each user.

CLARE has been used in some pilot experiments, but the results have been somewhat inconclusive. While users overwhelming felt CLARE helped them betters understand papers, they also felt they did not quite know how to use it.

1.3.6 *Chronicle: Application-Specific Versions.* A problem with traditional file systems is that the unit of naming, storage, and versioning is a physical unit supported by the system rather than a logical unit specific to an application. Chronicle [?], addresses this issue within the context of a spreadsheet. It has been developed as an extension to Lotus 1-2-3, and allows users to collaborate on spreadsheets. It supports multiple versions of the spreadsheet state, and allows users to share spreadsheet information.

Chronicle supports the abstraction of a named range, which is a named rectangular block of cells in a spreadsheet, and allows a user to ask for the values in the range to be stored as a new version. A version is associated with the range values, the range name, a user-defined annotation, a user id, date, and time stamp.

Users can include different named versions in a spreadsheet and perform computations on them. This feature can be used in at least two different but related ways: (1) Users can include data composed by different users such as "Sally's expense data" and "Joe's sales data" into a single spreadsheet and (2) they can try different versions of a named range such as "Sally's expense forecast" and "Joe's

expense forecast" to try different alternatives. Note that the second benefit is useful also in the single-user can who can create different versions for "my best case sales forecast" and "my worst case sales forecast". It is often the case that features added to an application for supporting collaboration are also useful in the single-user case. A spreadsheet that includes named ranges is called a scenario.

Chronicle allows versions and scenarios to be stored in databases shared among distributed users and also allows them to be mailed to each other explicitly. It also allows users to browse through the various versions of a named range stored in a shared database and indexes them by range name, user name, and scenario. It also allows users to simultaneously access a spreadsheet but not a version. Simultaneous changes to the same part of a spreadsheet create different versions of it that can later be used as alternatives.

1.4 Session- and Message- based Collaboration (Talk + Mail)

Session and message support can be combined in a collaborative application in at least two ways. The first, as illustrated by MUDs, is to allow users in a session to "whisper" some information to a subset of the users. A similar facility is provided by InternetChat, which allows users in a session to use regular mail to send URLs and other information to each other. Second, as suggested in [?], a workflow system can automatically start a collaborative session as one of the tasks in the workflow. Taking the workflow example of candidate review, after all review forms have been mailed in, the system can automatically start a session-based application to hold a meeting among the reviewers.

1.5 Session- and Artifact- based Collaboration (Talk + File)

In all of these applications, the abstraction of a session is combined with the notion of sharing an artifact. The applications differ in the artifact shared and the method of sharing.

1.5.1 *RTCAL: Real-Time Artifact Sharing.* RTCAL (Real-Time CALendaring) [?] was developed as part of Sunil Sarin's thesis work at MIT under Irene Greif and is one of the first applications that showed the possibility of real-time sharing of artifacts. The application is a decision support system that allows users to collaboratively schedule meeting times such as a thesis defense. It displays both private and public data: for each time slot, it shows to each user, in a shared column, whether slot is free for all users, and in another, private column, the details of any appointment of that user at that time. It supports common scrolling of these so that as users scroll to different time slots, the displays of both the private and public appointments is updated. It does not allow users to privately scroll to private time ranges, a feature the authors suggest. It gathers and tallies votes for the users and allows a user to leave and then later join the conference. Users are given the option of ignoring meeting times committed by the group in their absence.

User commands are divided into application and conference-control commands. Application commands manipulate the calendar and only the controller can enter these commands at any one time. Conference-control commands are used to pass control and enter and leave the conference. These could be entered at any time by a participant. A special role, called the chairperson, has the authority to terminate

the conference and determine who the current controller is. A special window displays conference status informations such as the topic of the meeting, who the current participants are, and who the chairperson and controller are.

How should this application be implemented? There are several approaches possible, which we shall study later. RTCAL creates a collaboration-aware replica on the workstation of each user and keeps these replicas consistent. This approach is used in almost all of the applications discussed in this section, except TeamWorkstation and ClearBoard, which use a collaboration-unaware centralized process, and CAIS, which uses a combination of the replicated and centralized approaches. Interestingly, when we see collaboration infrastructures, the opposite will be true, that is, most of them will be biased towards centralization. (What could be the reason for this?)

1.5.2 *Cognoter: Private/Shared Windows & Process Model.* Cognoter [?] is the most mature of the applications developed as part of Xerox's Colab project [?] which developed software that could be used collaboratively by small groups (of two to five members) co-located in the Colab room. Cognoter allows its users to collaboratively flesh ideas and transfer them into structured text documents. Instead of mixing private and public information in a single window and thereby creating the private scrolling problem of RTCAL, it creates separate windows for each user: a shared item organization window and a private edit window. The item organization window is used to display a set of text items, which are catch phrases describing the various ideas. They can contain other items and be associated with annotations. The edit window is used to create and modify items and their annotations. An item is transferred from the private edit window to the shared item organization window when it is committed by the user. Once an item appears in the item organization it can be moved by any user. As in some of the other applications we will see below, other users do not see intermediate positions of the item and only see the final position after the movement was completed.

An interesting feature of Cognoter is its automatic enforcement of a meeting process. The Cognoter supports three physically distinct phases of collaboration: brainstorming, organizing, and evaluation. In the brainstorming phase, the participants propose various ideas; in the organizing phase, they collect related ideas into possible alternatives; and in the evaluation phase, they evaluate the different alternatives and deleted those that were not acceptable. Commands special to a phase are not allowed in other phases. For instance, deletion of items can be done only in the evaluation phase. Compare this process with the CLARE process.

1.5.3 *GROVE: Access-Controlled Views.* GROVE [?], like RTCAL is a group outline editor, but unlike RTCAL supports fine-grained access control and allows private items to be viewed within the context of public information. A GROVE outline is a recursive structure consisting of structured items. Each item is associated with some text and can contain subitems of its own. Each user displays a view of the outline in a viewport. A view is a subset of the outline to which the user has access and a viewer is a window that displays an outline view. This distinction

allows users to share views without sharing the viewers displaying them.

A view may be private, public, or shared. A private view is readable only by the user who created it, a public view contains items readable by everyone, and a shared view contains items readable only by a subset of users. When a Grove session is started, the user initiating it specifies whether a private, public, or shared view is desired. A new viewer is created on the screens of all users who share this view. The viewer shows the entire contents of the current outline that are accessible to its user. Any additions of to the outline are shown only to the users who share the view. (Images of these users are displayed at the bottom of the viewport displaying the view.) The additions do not have unique numbers but do have labels that show their place relative to the nearest public ancestor (e.g. 1.2.*). In contrast, an entry made in a public view is numbered (e.g. 1.2) since it is readable by all users.

Grove also allows private and shared views to be dynamically forked from existing shared and public views. Whenever a new view is created by a user, viewports displaying it are created on screens of all users who can read it. This feature allows users to divide a meeting into several parallel submeetings. It is not clear how the work done in these submeetings are merged back into a common meeting.

An item displayed in a view is associated with permission to read, write, and change permissions. At any time, a user with the change permission right to an item can make an item readonly or writeable for a selected set of users. Writeable item are displayed in black and readonly items in grey in the viewports of the users. Users can bring up forms that show what kind of permissions the different users sharing the view have to an item. Grove provides commands to textually edit leaf items and insert, delete, open and close non-leaf items.

Grove also supports the concept of a a telepointer, a pointer that can be moved by a remote user. In Grove, a single telepointer is created for all collaborators.

A change made by a user to an item is immediately transmitted to the users sharing it. Grove allows multiple users to change an item simultaneously and ensures that they see these changes in the same logical order. It does not use centralization or aborts to ensure global order, optimistically expecting few inconsistent concurrent operations. When such operations do occur, it transforms them to ensure consistency, as we shall see in detail later. Moreover, it does not support locking, and relies on social protocol to prevent conflicts.

1.5.4 *CES: Tickle Locks and Version Histories.* Yet another group outlining system, CES (Collaborative Editing System) supports fine-grained locks, delayed transmission of changes to other users, and distributed versions of the document. It supports collaborative editing of a structured outline whose leaf level nodes are variable-length text segments. Each text node in the document is owned by a particular user, who is considered the primary author of the node, and resides on the machine of the owner. Secondary copies of the node are cached on other authors' nodes and are kept consistent with the primary copy.

Changes made by user are transmitted to others, not immediately, but when they are committed. To reduce user overhead, CES supports implicit commitment, that is, commits changes automatically as a side effect of "significant editing actions" such as as word deletions and carriage returns. Thus, a user viewing a node being

written by another user sees an out of date version of it until the changes made by the latter are committed.

CES supports "tickle locks" to ensure that two users do not write to the same node simultaneously. A user locks a node when he first edits it and retains the lock as long as some editing activity occurs. Some other user can take the lock away from the holder if the latter has been idle for a certain amount of time. At this point, all changes made by the holder are automatically committed.

CES keeps a stack of recently committed versions of the document and supports the regular stack operations, `top, push,` and `pop.` In addition, it supports the `fasttop` operation, which is not serialized with the other operations. As a result, it supports more concurrency but is not guaranteed to return the "true" top. This approach of optimistically expecting that distributed race conditions will not occur is taken by Grove and several other collaborative systems to support good response times.

CES is implemented using a distributed object system called Argus, which we shall study later.

1.5.5 *Aspects: Multiple Locking Modes.* So far, we have seen two forms of locking: coarse-grained floor control in RTCAL and fine-grained concurrency control in CES. We now see an example of a commercial multiuser application, Aspects [?], which provides multiple models of concurrency control and a special user-interface for explicitly getting and releasing locks.

In Aspects, a conference is created by a special user called a moderator, who has special access rights. It supports three modes of concurrency control or "mediation" and the exact one used is determined dynamically by the moderator. In the "free for all" mediation mode, locking is at the paragraph level. A user editing a paragraph sees a black bar on the side of it and other users see a grey bar. In the "medium mediation" mode, access to the complete document is serialized and users use a pen passing mechanism to pass control. A user can be writing, waiting for a pen, or just observing. Pen icons are used to pass control. A user can click on his closed pen icon to request the pen. The user's icon starts blinking and turns into a raised hand and the writer's open icon also starts blinking. The writer can click on the "pen in hand" icon to pass control to the first person waiting. Clicking on a black arrow shows the pen status of the different participants. In the "full mediation" mode, access to the document is again serialized but this time the moderator determines who gets the pen next after it has been relinquished.

Like Cognoter, Aspects provides support for gesturing by associating each user with a unique telepointer whose position is viewed by all users. In Aspects, upto 16 people can be in a conference simultaneously. When the conference is closed, everyone is sent a message informing them about the termination

1.5.6 *Groupsketch and GroupDraw: Multiple Gestures and Optimistic Locks.* So far, we have seen joint manipulation of textual documents. Groupsketch and Groupdraw [?] are two collaborative programs developed at Calgary by Saul Greenberg's group that allow group editing of drawings. Groupsketch is a simple drawing pro-

gram that allows users to draw the sketch pixel-by-pixel while Groupdraw is a structured or object-based drawing program allowing users to create geometric objects such as lines and circles and perform object-specific commands on them.

Groupsketch has many of the features of the previous applications: It displays identical images of the sketch window on the screens of all participants, displays cursors of all participants in each of these windows, lets any user enter any command at any time, immediately broadcasts all actions to all users, and supports modelessness. In addition, it provides special support for gestures. A cursor (or telepointer) movement can be used as a gesture, so cursors are made larger than their normal size. Four gesture kinds are supported, pointing, writing, erasing, and directing attention, and each has its own cursor shape. A cursor is labelled with the name of its user.

Groupdraw is similar with a few important differences, some of which have to do with the fact that the editor is object-oriented: Like Grove, it provides fine-grained access control, and allows an object created by a user to be writeable, readonly, or unreadable by others. It allows users to scroll to different parts of a shared window. Moreover, it provides a pallete for selecting and manipulating objects, and does not link different users palletes, which can cause the collaboration to be less fluid. It also supports implicit locking/unlocking by locking an object when it is selected/unselected by a user. When a user tries to manipulate any object, its agent checks with a remote server about its locked status. During the time it is listening for a reply, it performs the action, and undoes it if the object was, in fact, locked. This optimistic locking approach is similar to the GROVE optimistic broadcast and CES fasttop operation, and works well in practice.

1.5.7 *Ensemble: Implicit Sessions & Multicasting Telepointers.* Ensemble [?] is another group object-based drawing application. Users do not have to explicitly start sessions in Ensemble, they simply use Ensemble to access the graphics file. Multiple users who access the same file are automatically put in a common session. Moreover, users does not see the pointers of all other users in the session. They selectively export pointers to other users and import pointers from them. As in GroupKit, locking/unlocking is done implicitly at the object granularity, but while an object is locked, other users do not see changes to it. They see all changes to the object occur atomically in the remote displays when the object is unlocked.

1.5.8 *GroupDesign: Multiple Collaboration Modes.* GroupDesign [?; ?], another locking-based collaborative drawing tool, provides a less jerky interface for committing a series of changes to remote users. When an operation is initiated on an object, an icon is displayed in the object to other users indicating that the object is being manipulated. The shape of the icon indicates the kind of operation being executed and the colour identifies the initiator. When the operation completes, animation is used to explain to others the result of the operation. GroupDesign also provides a more intelligent locking protocol that allows users to execute operations on locked objects that are compatible with the operation that locked it. For instance, it allows simultaneous execution of rotate and colour operations on a

rectangle.

As in GroupDraw, each user can scroll to a different part of the drawing and place his window at a different location. A user's action on an object is is echoed audibly to those who have scrolled the object away by identifying the kind of the operation performed on it. GroupDesign provides two special modes to allows users to synchronize views that have been scrolled to different parts. In the Localize mode, a user is "teleported" to the views of one or more users, from which he can return to his view. In the Teleconference mode, users views are linked so that they see each others scrolling and window changes. The pointers are not shared in this mode, but a GroupDesign object can be used as a telepointer.

It provides two other useful modes to support awareness. In the Identification mode, it displays every object with the colour of the person who last modified it. In the age mode, it uses colour to indicate the age of the object. Finally, in the Time Relaxed mode, users actions are not seen by others until they explicitly commit them. Users can also offline play the history backwards or forwards.

1.5.9 *ClearBoard: Face-to-Face Awareness.* In all of the applications we have seen so far, a user is indirectly aware of the actions of another through telepointers, encoding of users by colours, audio feedback, and other features. Clearboard [?], designed for two-user collaboration, shows that it is possible for a user to be directly aware of the actions of a remote use. It appears to each user that other user is behind a transparent whiteboard, and the drawings made by the two users are shared rather than overlaid. Thus, one user can, for instance, erase a drawing made by another user or refine it. [2].

Mirrors, projectors, and a multiuser shared whiteboard application are used to create this illusion of face-to-face awareness. The displays of the users are tilted 45 degrees, and the image of a user is is reflected by a mirror, captured by an overhead camera, and transmitted to a rear projector at the other site which overlays the image on the screen. The shared whiteboard application ensures that users share changes to the objects drawn on the whiteboard.

1.5.10 *TeamWorkstation: Seamless Desktop/Computer Awareness.* Clearboard works well as long as the only shared objects the two users care about are the ones drawn on the whiteboard. What if they want to refer to both the physical objects on the distributed desktops and electronic objects created by computer applications. We can display these objects in separate windows or screens, but that forces a user to focus on one set of objects at a time. TeamWorkstation [?], instead, creates a seamless environment in which the two sets of objects are overlaid on a single shared display.

TeamWorkstation uses the CaptureLab [?] approach of sharing a single cpu (running a single set of applications) between the local and remote displays. It captures all video signals sent by the cpu to its local display and sends them over the network to the remote display. It also send images of each desktop captured by

[2] In an earlier version, Clearboard-1, the drawings were overlaid [?]

a video camera to the other site, where it is mixed with the cpu-generated stream to create the overlay. Finally, to support multiuser input, it captures the keyboard events generated by the remote user and sends it over the network to the local site, where it is multiplexed with the keyboard events of the local user before being delivered to the applications.

1.5.11 *Digital Desk: Integrated Desktop/Computer.* The need to overlay images of the physical desktop and computer "desktop" arises only when they are distinct. The DigitalDesk [?] shows that this does not have to be the case. In this system, the physical desktop serves also as the electronic desktop. A projector displays electronic documents or windows on the physical desktop, which can be intermixed with physical papers. Users can interact with both by pressing and dragging their fingers, which corresponds to pressing or dragging a pointing device. An overhead camera scans the paper documents and detects finger movements while a microphone attached underneath the desk detects finger presses.

A special application, the DigitalDesk Calculator, shows how this technology is useful for single-user interaction. User can interact with the calculator by either pressing on the projected keys, or more interestingly, pasting numbers directly from a paper to the calculator.

This technology is particularly interesting from a multiuser point of view since one can project the electronic and paper documents on a remote desktop, thereby giving awareness of both objects without the need to overlay the images. A multiuser version of this system has not, however, been built so far. (Can you integrate the TeamWorkstation and DigitalDektop designs?)

1.6 Message and Artifact-based (Mail + File)

1.6.1 *News Bulletin-Boards: Shared Mailboxes.* Pure message-based systems provide a mechanism to append messages to users' private mailboxes. Pure artifact-based systems allow multiple users to share an artifact. News combines features of these two kinds systems: As in the former, users send requests to append messages to a mail box, but the mailbox, as in the latter, is a shared object accessible to multiple users. In fact, news bulletin-boards have to be far more scalable than other shard artifacts we have seen so far, both because of the number of user sharing them (including potentially all of internet users) and the number of news messages generated (around 100MB per day).

How do we build such scalable objects? Fortunately, the problem is not as difficult as in file/database for two reasons. First,news messages are immutable, Second, and perhaps more importantly, there is no need to ensure that at any time users see the same state of the shared artifact, it is enough to ensure that a news item be eventually accessible to all users. Put another way, when users starts a news session, they are not expecting to receive all items posted before the session was started. They are happy if these news items are received in later sessions. critical dependencies among their actions,

News uses a simple approach forming the basis of Lotus Notes, which we will discuss later. Each user accesses then news system using a news client such as the NN and Gnus clients. The news client, in turn, contacts a news server, which keeps

a repository of all news items. The news client downloads these items for its user from the server and sends all news items posted by the user to the server. There are several replicated news servers in the internet which are connected either directly to each other or indirectly through other news servers. They keep their repositories consistent by propagating the postings they receive to other news servers to which they are directly connected. This approach allows a news server to receive duplicate postings along different paths, but the duplicates can be easily detected since each news posting has a unique id.

A news server may delete items to save disk space, which interferes with the guarantee that a posted news item should be received by all users.

1.6.2 *Message Filtering.* Even if the system can handle such large amounts of news traffic, how does a poor user sort through it to find information of interest. The solution is to use message filtering techniques to categorize messages according to different criteria. This categorization can be directly used by the user as an awareness feature or automatically by "agents" to eliminate certain messages.

We can use several criteria for classifying a message [?]:

—*News Group*: The news group to which it belongs, the most common criterion for selecting news messages.

—*Discussion Thread*: The discussion thread in which it appears, a feature supported by some news clients and also by Notes Mail, which link the messages in a discussion thread.

—*Urgent Responses*: Whether it is a response to a receiver's message and much time was taken to send the response. Users may be more interested in messages that are part of some urgent conversations in which they are involved.

—*Sender/Subject/Contents:* Strings appearing in the sender, subject and contents fields of a news item. Certain news clients support "kill files" that identify such strings that are not interest to the user such as certain subjects and authors, and these clients automatically eliminate the selected messages. Conversely, others allow users to specify strings selecting messages the user wishes to find.

—*Sender effort*: How much effort was required by the sender to mail the message: we can distinguish between messages mailed to mass mailing lists and those send individually.

—*Moderator selected*: Whether some moderator has selected it, a feature supported in moderated news groups.

—*Rating*: The ratings given by arbitrary users at the local site, as illustrated by the Tapestry system [?]. This is generalization of the moderator idea, allowing multiple users to pass judgement on a message.

1.6.3 *GroupLens: Extensible Aggregation-based Filtering.* In comparison to the moderator-based approach, Tapestry is more sophisticated in that it considers the input of multiple users but is harder to use and less efficient since the user must consider and the system must store all the ratings with a message. It is perhaps for this reason that Tapestry restricts the raters to a single site. GroupLens [?] shows that is possible to attain the benefits of both approaches.

Like Tapestry, it allows multiple users to rate an article, but unlike Tapestry, allows these users to be at arbitrary sites and aggregates, for each new reader of

the article, all the ratings of the message into one number that guesses how well that reader would like the message.

How does it guess a readers preferences? The idea is simple and intuitive: How well a reader likes a message is correlated to how well other readers with similar preferences have liked it, and how similar are the preferences of two readers can be determined by correlating the ratings they have given to the same messages.

To be more precise, given two users, A and B, who have rated some messages M1 to Mn, the correlation coefficient CAB, can be computed as follows

```
Sum (i = 1 to n) ((Ai - Amean)* (Bi - Bmean))
-----------------------------------------------
Sqrt ( (Sum (i = 1 to n)  (Ai - Amean)**2) * Sum (i = 1 to n) (Bi - Bmean)**2))
```

where Amean and Bmean are the means of the ratings given by A and B to the n messages.

Now given some set of users S who have rated an article, i, that A has not read, we can predict the rating of A as:

```
Amean +  Sum (over all users B in S) (Bi - Bmean)*CAB
          ----------------------------------------
         Sum (over all users B in S) CAB
```

Users can associate with a rating not their real names but pseudo names, an idea borrowed from the refereeing process typically used in the evaluation of conference research papers wherein each reviewer is assigned a unique pseudonym.

The GroupLens system shows how this idea can be implemented in an extensible manner on top of the current news architecture. It adds to the news clients and servers, another process, called a "better bit bureau," which is responsible for implementing the aggregation semantics. A modified news client displays aggregated news ratings to its user and collects ratings from the user. It sends/receives the message directly to/from the news server and the associated rating to/from a better bit bureau. The better bit bureau strips the rating from the message and sends it to the news server. In addition, it sends the ratings in a separate message to a special newsgroup, from which other better bit bureaus can retrieve and aggregate the ratings before giving them to the news clients connected to them. To reduce message traffic, they batch all ratings produced by a user in one news session in a single message. (Is the space required to save ratings significant? If so, can be alleviate this problem by perhaps reducing the functionality?)

This approach provides a nice partioning of function in a news system: News servers are responsible for message distribution, news clients for user-interface, and better bit bureaus for aggregation. It does not require changes to existing news servers but does require some changes to the news clients so that they can customize the filtering user interface for its user. By implementing the semantics of aggregation in better bit bureaus, it allows all news clients to share a common implementation of these semantics. Moreover, users can provide their own better

bit bureaus to change the aggregation semantics. Thus, this system is completely extensible and is designed to reuse the message distribution facilities of existing systems.

1.6.4 *POLITeam: Interleaved Message- and Artifact- based Collaboration.* Sometimes message-based and artifact-based collaboration are two collaboration modes for achieving the same task. Consider the joint development of a document. We can mail each other the document or create it as a shared artifact. In these situations, it useful to integrate the two modes by allowing users to easily switch between them.

This is exactly the kind of task for which POLITeam workflow system [?] was designed: developing a minister's speech. Several organizations are involved in the composing and commenting of the speech, and traditionally this collaboration task was achieved by routing the document through all the relevant organizations. With the decision of Germany to move the capital from Bonn to Berlin, there is a need to distribute this task during the transition period. POLITeam automated and distributes it by providing a workflow system that automatically routes the document to its next destination once a user was done processing it. The traditional method, however, serializes access to the document in that till a user is done with the document, no one else can access it. Based on user input, POLITeam extended this scheme by allowing the current owner of a document to temporarily share it with other users (such as the previous recipient), thereby reducing the overhead of continually sending it back and forth between them. Thus, it allows message-based and artifact-based collaboration to be interleaved in the processing of a document, thereby providing another way to integrate these two kinds of collaboration. It can be thought of a hybrid between mail and news that allows us to dynamically switch between their features: The former supports private messages in mailboxes while the latter supports public news items in bulletin boards. POLITeam provides a scheme for dynamically transforming an object from a private message to a shared artifact, and vice versa.

1.6.5 *Linking Artifacts and Messages.* Yet another way to integrate message- and artifact-based collaboration is to link private messages and public artifacts. Some mail systems link artifacts to messages in that they understand artifact names (such as URLs) in messages and allow users to click on them to launch artifact viewers. Conversely, as we saw in Quilt, messages about an artifact can be linked directly to the artifact. Moreover, in Quilt, messages about a user's activities on an artifact (such as "Reorganized section 2") can be automatically mailed by the system to collaborators who have expressed an interest in them.

1.6.6 *Mercury: Automatic Error Notification .* Mercury [?] also generates artifact-based notifications for collaborators but displays them within the context of the artifact and automatically determines the users who receive them based on the artifacts they are manipulating. The artifacts are programs comprised of distributed modules, with a single programmer responsible for a module. The system allows different programmers to simultaneously view and modify different modules of the program, but allows a particular module to be viewed and modified by only its owner.

Mercury is derived from the technology of language-based editors. These editors understand the structure, syntax, and semantics of the underlying program and inform users incrementally about syntax and semantic errors. Mercury extends this notion by incrementally informing the, possibly distributed, programmers responsible for a program about errors in the interfaces of the modules they are editing. When a programmer makes a change to the interface exported by a module that is inconsistent with an importing module being viewed by another programmer, Mercury incrementally informs the latter about the error. Mercury also supports "what-if" scenarios that allow programmers to see if changes made by them would cause errors in the modules of others.

Mercury puts a "firewall" around a module that is in an inconsistent state and does not check its consistency with other changing modules until the module is committed by its owner. As a result, spurious error messages will not be generated if two programmers simultaneously make changes that are individually but not collectively inconsistent with the original program. If an interface change in one module causes an error in a module whose machine is not connected or whose owner is not viewing it, then Mercury stores the information and displays the error message the first time the module is viewed by its owner after the connection is reestablished.

1.7 Session-, Message- and Artifact- based Collaboration (File + Mail + Talk)

Finally, let us look at applications that integrate session-, message-, and artifact-based collaboration. In all of these applications, users in a session can share changes to some artifact and send private messages other users.

1.7.1 *Astro-VR: MUD++.* Astro-VR [?], an extension of MUD, is an environment to allow astronomers to meet each other and give short presentations. Specific facilities include a "self-contained" electronic mail and bulletin board restricted to the participants, shared, user-defined links to astronomical images, an editor/viewer for short presentations of text and images in special conference rooms that keeps track of the discussions, and collaborative access to existing programs used by astronomers.

1.7.2 *Forum: Distributed Presentations.* A similar tool, Forum is designed to allow a speaker to make a presentation to a large distributed audience. It allows each audience member to view the slides and a video of the speaker. Audience members can press a button to raise their hands to ask questions, and the speaker sees the names of those who have raised their hands. The lecturer can call on one of these users to speak and a stored image of the student is displayed to the speaker and all other audience members. Students can also vote in response to a poll, and the results of the poll are shown to all users. Finally, they can mail private comments to the speaker and other students.

1.7.3 *IRI: Video Windows and Application Sharing.* IRI (Interactive Remote Instruction) [?] is similar in that it allows the audience members to view the speaker and slides and mail to the speaker anonymous answers in response to surveys. Instead of showing static pictures of audience members, it shows videos of students who have raised their hands. Like Astro-VR, it also allows users to share an existing

application such as AutoCad.

1.7.4 *CAIS: Interleaving Session- and Non Session-based Collaboration.* In all of the tools we have described so far, users either manipulate an artifact privately or in a joint session. CAIS (Collaborative Asynchronous Inspection System) [?] illustrates the benefits of intermixing of these two modes of collaboration. Traditional inspections assume that the complete inspection task will be done in meetings. CAIS changes this paradigm to allow part of this task to be done in asynchronously, thereby reducing the overhead of scheduling meetings. The application supports discussion threads, voting on faults, and fault summaries. At the end of a user's individual asynchronous session, other users are automatically sent mail regarding the comments and votes entered by the user. If any user votes to hold a meeting, then that fault is set aside for the next synchronous meeting. Moreover, during a meeting, the application allows a comment to be broadcast incrementally, on every keystroke, or in batch when it is committed.

Like CES, CAIS distributes the shared information among distributed autonomous objects. In particular, it creates a separate distributed object for each annotation, the document being browsed, the history log, the hierarchical discussion threads, the collection of faults, and the inspection summary. It is built on top of the Suite infrastructure (which we shall study later) and uses its distributed object primitives and flexible coupling facilities for implementing its features.

Experience with CAIS shows that most faults can be identified asynchronously, and meeting are held only for the few controversial cases that occur. It also shows that the inspection process takes longer when asynchronous inspection is allowed but finds more faults. The duration of the inspection process was measured by determining the periods during which a user was active. It is not clear what fraction of these periods users actually performed useful work.

1.7.5 *VisTool: Editable, Piped, Collaborative Views .* VisTool [?] is an application for automating the process of scheduling external speakers to a department. It associates with each visitor information about the title, time, room, and abstract of the talk, the schedule of meetings with the members of the department. Users can create views of this database by inputting queries or patterns specifying the information of interest. Like databases, these views can be cascaded or "piped", that is, new user views may be derived from other views. For instance, we can derive from a view showing all programming language talks a new view showing only those programming language talks scheduled on Mondays. Unlike database views, however, these views are editable in that updates made to them are reflected in the views/databases they are derived from. These updates also cause related objects to be modified. In particular, scheduling meeting between the visitors and department members automatically result in corresponding updates to the private calendars of the latter. Moreover, users concurrently modifying the same object are implicitly put together in a joint session wherein they can detect conflicts and, as in a RTCAL session, negotiate a non conflicting meeting times. In addition to editable views, VisTool also provides shell commands to modify and visualize the database and the views displaying it. For instance, the a user can execute the command:

```
setvis -dates  "Mon/2/3/97" Sproull
```

to set the dates on which visitor Sproull would be visiting the department. All views displaying this data are updated immediately. Finally, a user can interactively associate a programmer-defined operations on a view such as AddAppointment or DeleteAppointment with handlers (shell scripts) that are automatically invoked when the operation is executed. These handlers, typically, use mail or xmessage to send a notification message to the users and take as arguments the parameters of the operation. Thus, end-users can augment the functionality provided by the application programmers.

Like CAIS, this application was implemented using Suite and and is composed of several communicating distributed objects. In particular, it has a separate object for each visitor, view, and private calendar, and keeps these objects consistent.