EXHIBIT 1

Part 2

- Searching for activity by user or by properties
- Tracking updates and keeping an historical record of activities
- Create views on the fly, which are updated dynamically whenever new activity takes place
- Saving these views for later use

As we will see, each of these concepts can be applied to the DISCIPLE system in order to allow the user to more easily grasp the work that the team has performed on a given project.

2.3 Taligent

The former Taligent corporation produced a workgroup software product called Places For Project Teams around 1997. Since then, the firm has been dissolved, and their product line has, for the most part, been absorbed into IBM's software division. I have been unable, unfortunately, to locate any reference to Places in the IBM literature. However, this package introduced some concepts which could be helpful within DISCIPLE.

The goal of the Places environment was to facilitate workgroup discussions, track milestones, and share data. A key portion of its functionality is derived from a time-based view of the history of each discussion conducted within the system. Messages are

displayed using colored bars overlaid on a chart. Header information is displayed for each of these when selected, and full details can be retrieved as well. Milestones are shown in this view as well.[10]

This view provides a method of rapidly conveying to the viewer information about which documents have been contributed by each collaborator. It is easy to see how this is applicable to the DISCIPLE environment, where a major challenge is to allow the user to understand the involvement of all other team members.

2.4 DISCIPLE Archive Server

Previous work in CAIP Center resulted in an Archive Server for the DISCIPLE environment to populate a database and maintain a session history. [20] These sessions can then be replayed in order to allow users to review the work that had been performed previously.

Prior to this point, DISCIPLE was useful only for synchronous work, where all participants could find a common time during which to "meet" electronically. This can be especially difficult when users are distributed across multiple time zones. The Archive Server, through its ability to replay sessions, lets those individuals contribute at times that are convenient for them.

While this was a step forward, it was limited in its practicality. A method was needed for consolidating the reams of data that result from collaborative sessions, displaying it within its context, and transferring this knowledge to the user.

The EventStream interface provides this context. The Archive Server's replay feature is still quite valuable, and the two components complement each other.

2.5 MITRE Multi-Modal Logger

The MITRE corporation, a not-for-profit group, has done some work on recording activities during human-computer and human-human interactive sessions. They've termed their effort the Multi-Modal Logger, and its goals include recording, retrieving, annotating and visualizing data about collaborative sessions. [12]

The Multi-Modal Logger supports many types of content, including audio, text and graphics (although it runs atop a flat-file database – it does not store complex data, but rather a pointer to its location elsewhere). It provides this functionality via an API which is available for several languages and platforms.

Perhaps the most relevant portion of the project to the EventStreams environment is the visualization tool. It consists of a grid where horizontal lines represent datatypes, and time is represented by vertical lines.

As we will see below, EventStreams addresses many of the same challenges as the MITRE system, but takes a slightly different tack.

3 EventStreams Interface

The research being performed on the Lifestreams metaphor at Yale University and at Mirror Worlds Technologies has been described above. With some modification, these concepts can be adapted to fit within the DISCIPLE environment and greatly enhance its ability to meet its design criteria.

The Lifestream system is based on using the document as its atomic unit. All entities within the system are represented as documents – messages, appointment reminders, stock quotes, etc. Within the shared-workspace environment that is DISCIPLE, though, the concept of the document is meaningless. DISCIPLE's basic units are the object and the event.

The current whiteboard application lets users work with and see the state of the objects within their current project. What is needed is a method of viewing the events that have occurred thus far, along with other semantic information about them. This need can be filled using a derivative of the Lifestreams metaphor, which I've termed the EventStreams interface. As its name implies, the basic unit of this system is the event.

Whereas a Lifestream consists of the *documents* created by an individual, his EventStream would consist of the *actions* performed by that person. Perhaps the following example will help clarify the difference. The Lifestream of an investor might consist of a series of monthly statements (*documents*) that are received from his

brokerage company. His EventStream, however, would consist of a stack of purchases and sales (*events*), which would include information such as date/time, shares, dollar value, executing broker, commission charges, realized capital gains, etc. Each of these paradigms has areas of applicability, and other areas that it does not represent very well.

3.1 System Architecture

EventStreams is implemented as a new module within the DISCIPLE framework, designed to be used alongside the existing DISCIPLE shared workspace.[4] It is a read-only environment, meaning that all modifications to the state of the current project continue to be effected from within the workspace.

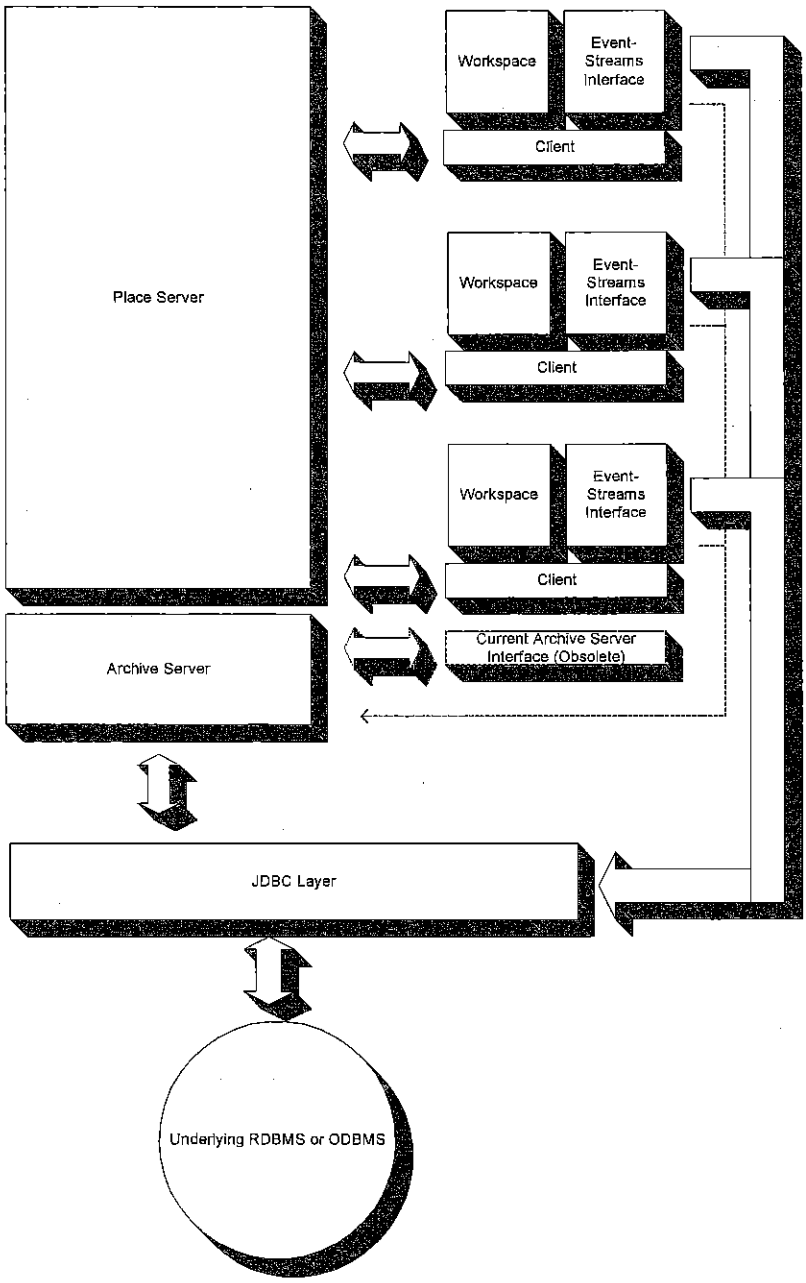

Figure 3 DISCIPLE/EventStreams System Architecture

It is linked to the other components in the system via the underlying relational or object-oriented database management system that is used to record the history of a collaborative effort. The interaction of these components can be seen in the Figure 3.

As the diagram shows, each user could have both the Workspace JavaBean container application and the EventStreams client active on their workstation (although there is no requirement that both be running – the user may choose to use only one during a work session). New contributions to the team effort (creation, modification, and deletion of JavaBeans) would continue to be made via the Workspace, but the EventStreams view would be available for use in reviewing all work done to this point in time. New activity would be communicated unidirectionally through the database. More detail is given on this in the sections covering database structure and communications.

The dashed line to the Archive Server in Figure 3 indicates that upon request the EventStreams client will initiate a request for it to replay some portion of a project. The interface currently used to access the Archive Server is shown grayed-out, as it can now be subsumed into the more functional EventStreams package.

Each time a new event occurs at any node during a session, it must be communicated to all other collaborators. This process is handled by the DISCIPLE Collaboration Bus. A high-level overview of the system is shown here:

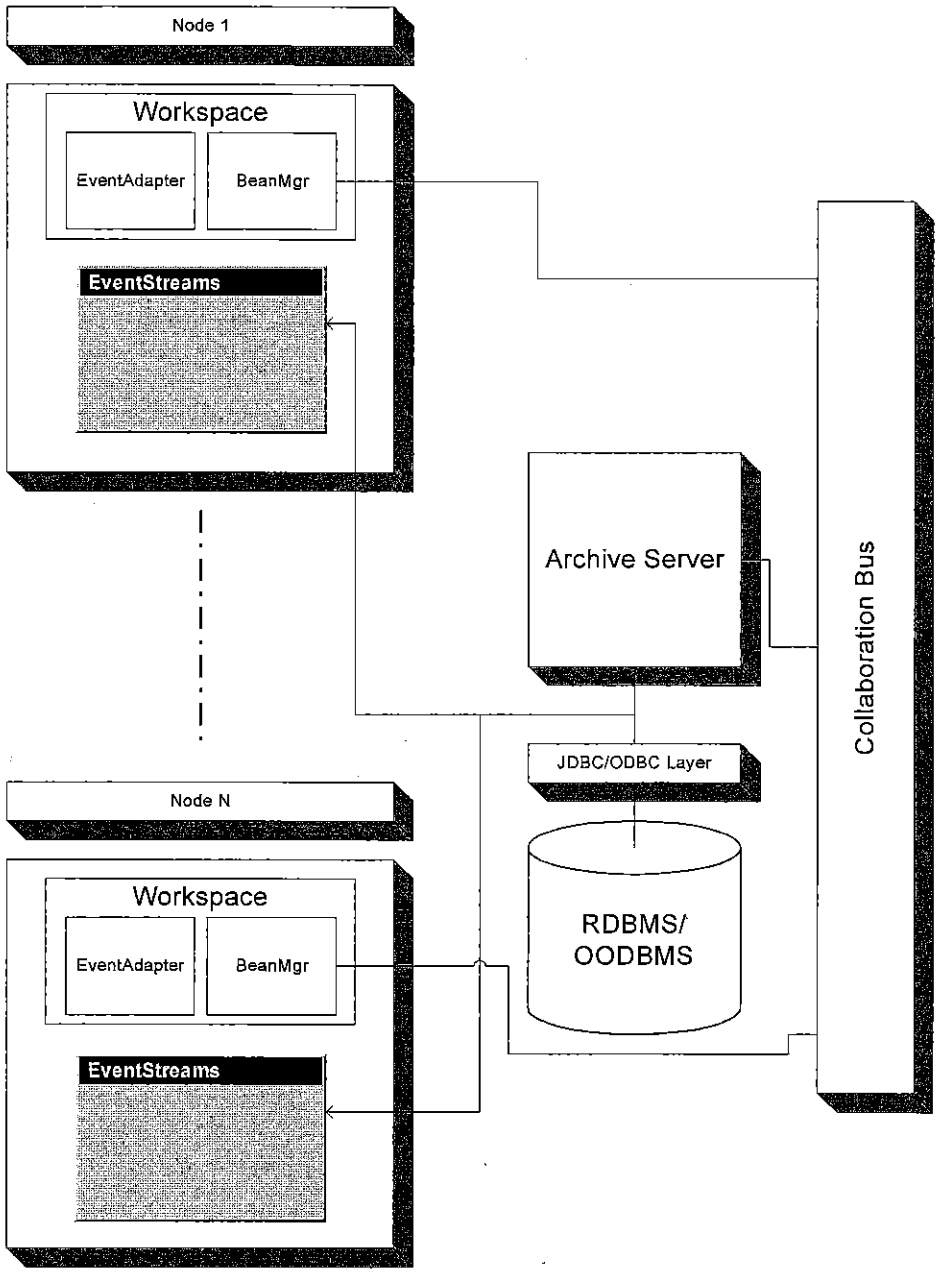

Figure 4 Event Propagation via the Collaboration Bus

When a user at any node generates a change in any object in the system, the component will inform any registered listeners about the change, including the EventAdapter module of the Collaboration Bus. The Event Adapter lets the Bean Manager know as well. This has all taken place locally on the user's workstation.[17]

The Bean Manager broadcasts the event over the collaboration bus layer to all other active Bean Managers, as well as the Place Server. The listening Bean Managers pass the message along to their local instance of the component to ensure that all the Workspaces are in sync. The Archive Server updates the underlying database to add the latest event to the history stream that it is building. Note that messages are only added to the database, never modified or deleted. This is the key that allows EventStreams to present a complete view of all the actions that have taken place during a project.

The local EventStreams clients on each node poll the database and learn of new events. These are posted to all appropriate views (views are described in Section 5.4 on Substreams). Ultimately, it would probably be desirable to simply register the EventStreams clients as listeners, which would improve performance over slow, widely dispersed networks.

3.2 Stream View

The Stream View, which is based on the Lifestreams format, is the workhorse of the EventStreams system. It is the default view that is shown when the user logs on, and

displays a chronologically ordered stack of pages, each of which represents a single, atomic action that a collaborator has performed during the project. A sample is shown in Figure 5.

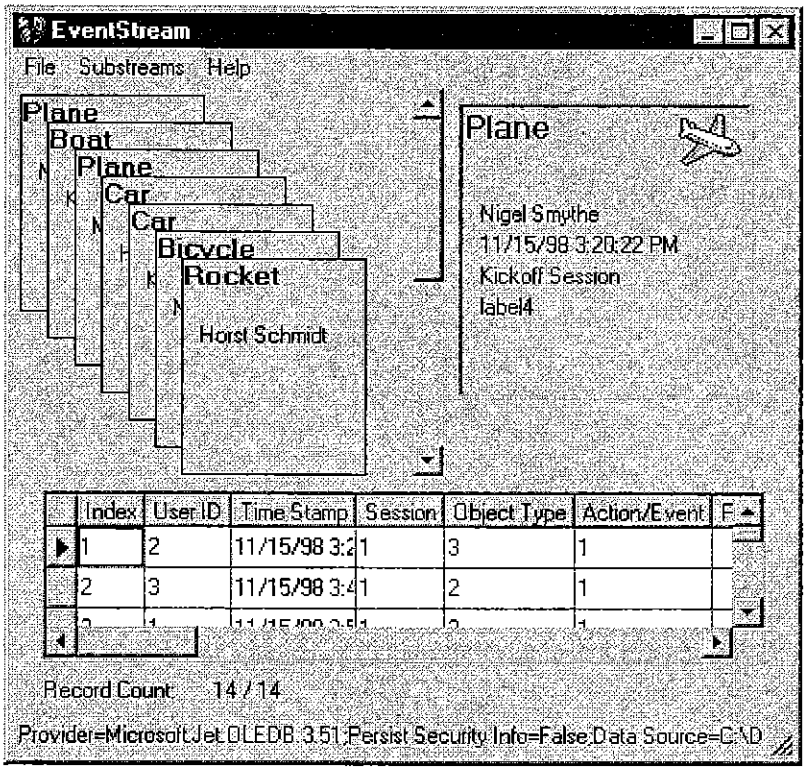

Figure 5 Sample Event Stream view.

The hypothetical project from which the sample screen is taken is the effort of four individuals from four different locations. They have collaborated in three separate sessions, and performed a total of fourteen different actions within the DISCIPLE Workspace.

The objects that the collaborators are working with in this example are five different transportation vehicles (car, boat, rocket, plane, bicycle). These real world objects are

represented within the DISCIPLE environment by JavaBeans (the user drags, for example, the boat JavaBean into the Workspace, and a boat object is then instantiated).

It is the responsibility of the JavaBean author to provide a good simulation of the entity that he is attempting to model. It must possess a range of properties and methods which allow the participants to express to each other exactly what actions they wish to perform on it.

The Car object might have associated in a database table such properties as make, model, year, miles per gallon rating, and capacity. It might also provide methods to allow the user to perform such tasks as get its location, set its destination, and start and stop the engine. The fact that all the functionality of the real-world object is encapsulated within its JavaBean means that the collaboration environment is indifferent to the content of the project that it is being used for, and can handle anything that can be modeled by a Bean.

The Stream View window is divided into four major areas (Figure 5). The first of these is the menu bar, which is configured as shown in Figure 6.

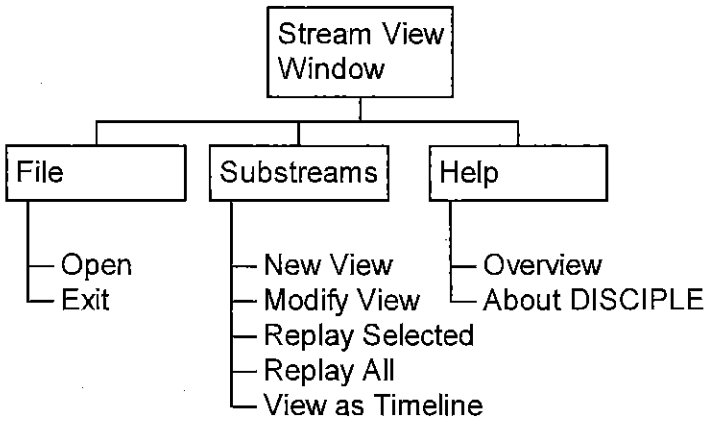

Figure 6 Stream View menu structure

The functions included in the menu will be discussed in turn in the following sections.

The upper-left region is a stack of pages, each of which represents a single event (Figure 5). In a project of any significant size, there may be more events to be displayed in the view than will fit within the window. Therefore, to the right of the stack is a vertical scroll bar which can be used to move through the entire set of events.

Due to the need to display many pages in this pane, very little information can be shown for each event (for all intents and purposes, only the type of object that the event occurred on is visible to the user in this area). Thus, the upper-right hand area contains the Detail Panel (Figure 5). When the user selects any event in the stack by clicking on it, full details are provided in this panel about that event. This includes the user who performed the action, the date and time that it occurred, and the session name.

It is important here to recall that the Detail Panel is displaying information about a particular *event*, and not about an *object*. This information is generic and is the same for all events, regardless of the underlying object. And it is also important to remember that this is not an attempt to visualize the event itself, but to provide a *description* of the event.

One of the items displayed about the event is an icon (the purpose of which is to help users assimilate the information a bit more easily than if only a textual name were displayed). This icon is that associated with the JavaBean of the underlying object on which the event occurred. The Detail Panel, knowing the underlying object, retrieves and displays the icon related to that object (it is not stored in the event database).

The final section is the status area, which is located on the lower portion of the form (Figure 5). The key items are:

- **Record Count** – This line indicates the number of records in the current view, and the total number of records in the project overall. The number of records in the current view is determined by the selection criteria for the window. This is discussed in the section on substreams.
- **Data Grid** – Provides a complete presentation of all the data retrieved by the current query. This may or may not be considered necessary when the interface is ready for final release, but it is extremely useful during development.

- **Connection String** – Indicates the database to which the user is connected, along with relevant connection parameters.

More detail on the Stream View is given later when queries and substreams are introduced.

3.3 TimeLine View

The Stream View provides a great deal of information and is very flexible in allowing the user to determine how he would like to have the data presented. However, the one piece of contextual information that it does not provide is a sense of the timing of the events. I've attempted to address this shortcoming by introducing the TimeLine View.

The TimeLine View is arranged along a scrolling grid, with time running along the X-Axis and the list of collaborators running along the Y-Axis. Each event that occurs is plotted graphically on the grid using the icon representing the object. A sample screen shot is shown in Figure 7, taken from the sample project described above.

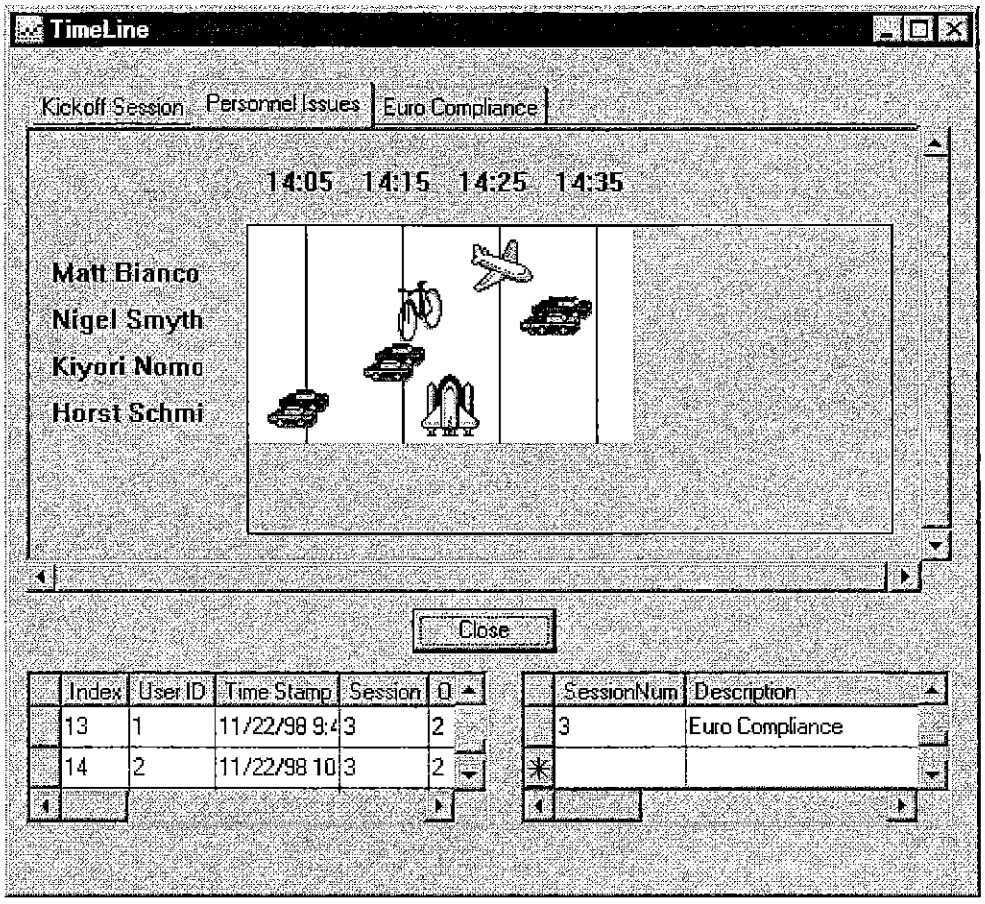

Figure 7 Sample TimeLine View screen shot

You'll notice that the form uses a tabbed-page metaphor, with each page representing a single collaborative session (with the title of the session appearing on the tab). The reason for this is that a single session generally has a duration of minutes to hours, whereas sessions may be spread over weeks or months. The tabbed-page construction allows the user to quickly zero in on the particular point in the collaborative process which is of interest.

The TimeLine View can be selected from any Stream View window by selecting the Substreams menu item and clicking *View as TimeLine* (Figure 6). The new window that is created will represent the same set of events that have been selected in the Stream View. This view provides more of an overview of the collaborative process than a great deal of detail. It simply indicates the times at which each user created or modified an object of a given type. For a more in-depth analysis, one would need to return to the Stream View, or perhaps replay the events via the functionality provided by the Archive Server.

The lower portion of the form is devoted to two data grids, which display the full results returned by the active query, and the set of sessions to be displayed (Figure 7). This shows much more data than can be displayed on the TimeLine itself. It is very useful during the development process, but may or may not be considered practical in the final release.

3.4 Substreams

To this point, we have been primarily concerned only with a project's complete EventStream. However, as a project grows large, this single view becomes increasingly unwieldy. When the stream consists of hundreds or thousands of events, it is not realistic to expect users to be able to locate items of interest or to be able to glean any understanding of the relationships among them. Collaborators must be provided with a tool to "slice and dice" the data to put it in a more manageable format.

This tool is provided by the substream feature. When the user selects the *Modify View* choice from the Substreams menu, they are able to specify the criteria which an event must meet in order to be included in the current view. They design this query using the form in Figure 8.

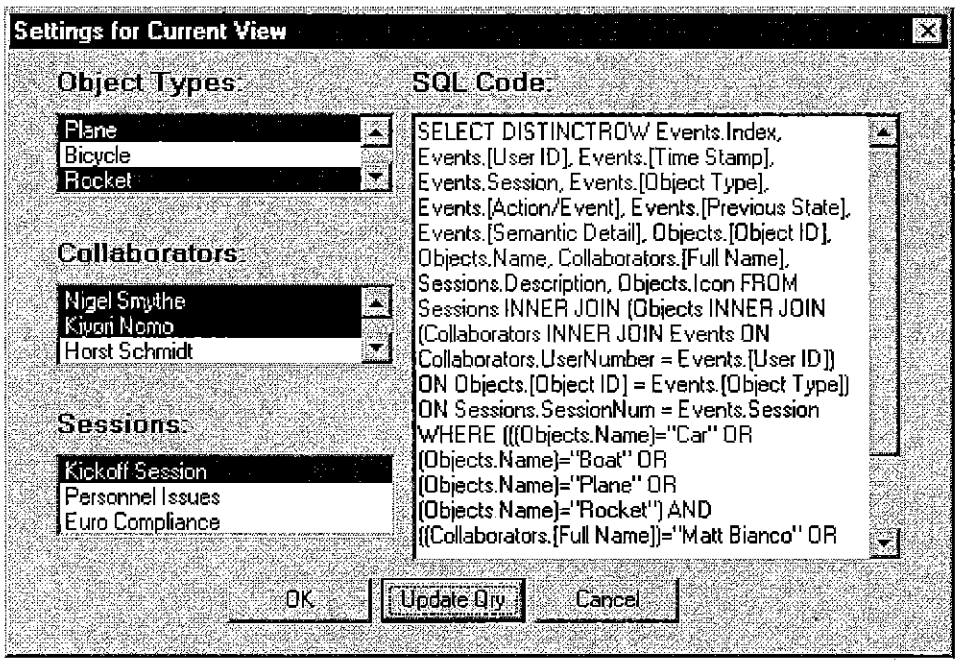

Figure 8 Substream query criteria screen shot

This interface uses an easy to understand query-by-example metaphor to enable the user to specify items of interest. The entire project database is searched, and each type of object, participant, and session is retrieved. Each of these may be selected in any combination (e.g. all the events initiated by a given user in either of two sessions,

regardless of the object type). These choices are displayed in three scrolling multiple-selection boxes on the left of the form.

The right-hand portion of the window is dedicated to the query display window. After the user has made his selections, he may select the "Update Query" button. The application then formulates an appropriate SQL query which will produce the desired results, and displays it within this panel for review. At this time, changes can be made to the query specifications, and the SQL code can be regenerated. Alternatively, the Cancel button can be pressed, and the application will ignore any changes that have been made. Finally, if the user is happy with the new filter, he can click OK, and the results of the effort will be returned to the parent window.

When a new query string is returned to the parent Stream View window, the application reloads its list of events to reflect this. The updated stack is displayed in the left hand pane, and the record count is updated to indicate how many events have been selected out of the total number in the project.

In addition to modifying the current window, the user may elect to spawn a new, separate substream window. This is accomplished by selecting *New View* from the Substreams menu (Figure 6). There is virtually no limit to the number of windows which may be opened, and the selection criteria for each is independent from all others.

3.5 Persistent and Transient Views

One of the strengths of the EventStreams interface is the ease with which new views can be created on the fly by specifying new query criteria (thus generating a custom-designed substream). This gives the user the freedom to look at a problem in many different ways, without becoming discouraged by having to complete a complex and time-consuming procedure to do it.

Thus, a user might quickly create a view of a project to allow him to see a very specific aspect of the effort. This view might very likely be useful only under the current situation, and will probably never be used again. After using the substream for a brief period, the collaborator would then delete it altogether. This type of query is referred to as a *transient view*, as it is created, utilized, and deleted in a short period of time. An example would be "Show me all the instances where User X interacted with any instance of Object Y."

Alternatively, there will be a set of views that will always be useful. After creating one of these, the user will want to refer to it over and over, and not lose the definition. These are referred to as *persistent views*. Although not implemented at this time, these queries would be saved back into the database to allow them to be recalled at the user's convenience. A case where this might be useful would be if Object Z represented a particularly sensitive piece of information (say, a salary file for a corporation), and the user always wanted to know whenever anyone accessed this data. A view could be

created which would dynamically update whenever this was the case. The query would be given a name, and saved for further use.

3.6 Session Replay

Despite the features of the EventStreams interface, it would still be useful under certain circumstances to have the ability to replay all or part of a collaborative session. A mechanism to achieve this already exists within the DISCIPLE environment.

A previous paper written by the DISCIPLE team describes a module for recording participant actions and replaying them.[20] This is accomplished by re-executing the stream of events that have been stored in an object database.

Rather than re-implement this existing functionality, it would be more productive to create hooks into the existing components. This work has not been completed to this point, but it should be straightforward to pass a start and end time to the Archive Server from EventStreams, rather than having the user enter this information into the replay system's current user interface.

Also, the previous Archive Server was developed in C++, and needs to be ported to Java in order to be fully integrated into DISCIPLE.

3.7 "Squish" Technology

The DISCIPLE collaboration-enabling framework was designed to accommodate all general-purpose JavaBeans. No special properties or methods need to be built into a component to allow it to be utilized within the shared workspace. However, if an object is built with the collaboration framework in mind, certain features can be included that provide extra functionality to the user. The DISCIPLE framework supports two types of collaborative applications: "Collaboration Transparent" and "Collaboration Aware."[17,18]

To this point, it hasn't mattered whether each of the JavaBeans being utilized within an EventStreams session was collaboration-aware or –unaware. All the features described thus far are equally applicable to both types, but we would like to take advantage of collaboration-awareness when it exists.

One of the most powerful features of the Lifestreams interface is something that the designers refer to as "Squish" technology.[6,16] This is a means of condensing the contents of many separate documents down into one summary document. The content of the summary document depends on the type of document being condensed. For example, given a stack of documents representing each of the stocks in a portfolio, the Squished document could provide a summary of profit/loss, graphs, and perhaps a statistical summary of the risks involved.

While not implemented at this time, it should be fairly straightforward to provide a set of hooks from the EventStreams interface which would call the methods in a collaboration-aware JavaBean to have it summarize the events contained in an active substream. This addition would dramatically enhance the functionality of the system and the amount of value that it can add to the collaboration process. The only limits are what features can be included within the Beans that are part of the project.

4 Collaborative Features

As mentioned above, the DISCIPLE system is being developed under the auspices of DARPA as part of their effort to build a groupware environment that enables geographically dispersed participants to cooperatively manipulate multimedia data.[3] The EventStreams system is an attempt to address several of the DARPA requirements that are not satisfactorily fulfilled by the existing collaboration framework architecture. The following is a description of some of the specific goals and requirements set forth in the contract, and how EventStreams can be used as a solution.

4.1 Addressing DARPA Goals

DISCIPLE is meant to address the requirements of the DARPA Intelligent Collaboration & Visualization (IC&V) program, which are set forth in an overview paper of the same name. This document lists the following tasks as the key challenges for the project:[14]

- Develop Adaptive Session Management Software
- Develop Semantically-based Tools for Sharing Meaning
- Develop Team-based Visualization Software for Sharing Views
- Evaluation Metrics and Methodologies

The EventStreams paradigm answers, to some degree, the second and third challenges. These are discussed in the following two sections.

4.2 Tools for Sharing Meaning

The goal here is to "develop structures for describing collaboration objects and the relationship between those objects in a form that can be understood jointly by human collaborators". These are the objectives of the Stream and TimeLine views – allowing the user to easily grasp the state of a project, and the path that the collaborative effort has taken to this point. Some of the more specific sub-tasks into which this challenge can be decomposed include:

Sub-Task	**How is it addressed in EventStreams?**
Need methods for capturing, summarizing, and indexing collaborative sessions.	Sessions are archived by the combination of the collaboration bus and archive server. EventStreams doesn't, strictly speaking, index the contents of each session, but it does provide a means of rapidly locating points of interest. Summarization can be accomplished through the implementation of Squish agents as described above.

Develop tools to help users structure and annotate irregular information associated with collaborative sessions.	The Stream and TimeLine views provide a means of visualizing the sequence and timing of participant interactions. Annotations are not implemented in the current version, but it would be straightforward to add a mechanism for users to attach comments to events which would then be stored in the database for future reference.
Create semantic structures and algorithms that enable automated assistance in the discovery of relevant collaborators and information.	Substreams and query-by-example features allow a small subset of events to be retrieved which meet some specific criteria. For example, a collaborator could create a view which would show all instances of particular actions being performed on a certain object. Only those items of interest are displayed. Once this information is obtained, it would be obvious who else among the participants may have similar areas of investigation. For example, an engineer working on modifications to a rocket's guidance system might want to create a view which shows which of his peers have worked on the same system. Thus, he can coordinate with them to ensure that his changes are compatible

	with their requirements. The EventStreams User Directory could then be employed to obtain further personal information about that participant and to understand their role in the project. Additional on-line or off-line sessions could then be arranged. This feature is described in detail in the following sections.

The key to EventStreams' ability to aid in the collaboration process is the fact that it can convey the context in which events occurred. This allows participants to understand relationships to other events and among participants. This support for the valuable semantic information associated with the project workflow is lacking in most groupware applications today.

4.3 Visualization Software for Sharing Views

The challenge in this case is to use visualization techniques to improve the collaborative effort. More specifically, it can be broken down into the following components:

Sub-Task	How is it addressed in EventStreams?
Can individual views of a situation be adapted based on roles taken within a collaboration to improve the effectiveness of a problem-solving team?	I think that this challenge is more directly addressed to the DISCIPLE workspace. However, EventStreams does offer the ability to create customized views for each individual (which, consequently, would be role-based). This can be accomplished through the substream and query functions, whereby each user can create an array of streams which are tailored to his or her role in the project (see the section on Persistent and Transient Views for more detail). This allows the user to focus solely on the data that is relevant to him, and therefore should increase his effectiveness.
Can novel means of representing collaboration spaces and shared objects improve the effectiveness of collaborations?	I'm biased, so I leave this question for the reader (and EventStreams user) to decide. But one point: in addition to the built-in Stream and TimeLine views, the "novel means" of representing data will be provided by the summary views generated by the Squish agents.

EventStreams provides a set of powerful tools to aid a user (who may or may not be familiar with work done previously on a project) visualize and internalize the events that have transpired. This lets the participant come up to speed more quickly, and stay current with more aspects of a project than would otherwise be possible.

4.4 User Directory

As important as it is for a participant to grasp the events that have occurred during the course of a project, perhaps just as vital is an understanding of the identities of the other team members, their roles, and their areas of expertise. The DISCIPLE environment already offers a means to gain this insight [4], but the EventStreams system gives further information through a User Directory.

The User Directory is accessible from any Stream View window. When one selects an event from the stack, several facts are displayed about it within the Detail Panel (Figure 5), such as the time it occurred. One of the pieces of information that is displayed is the name of the individual who performed that action. The user may double-click on this item, and the system will display the creator's biographical record from the database. A sample directory record is shown in Figure 9.

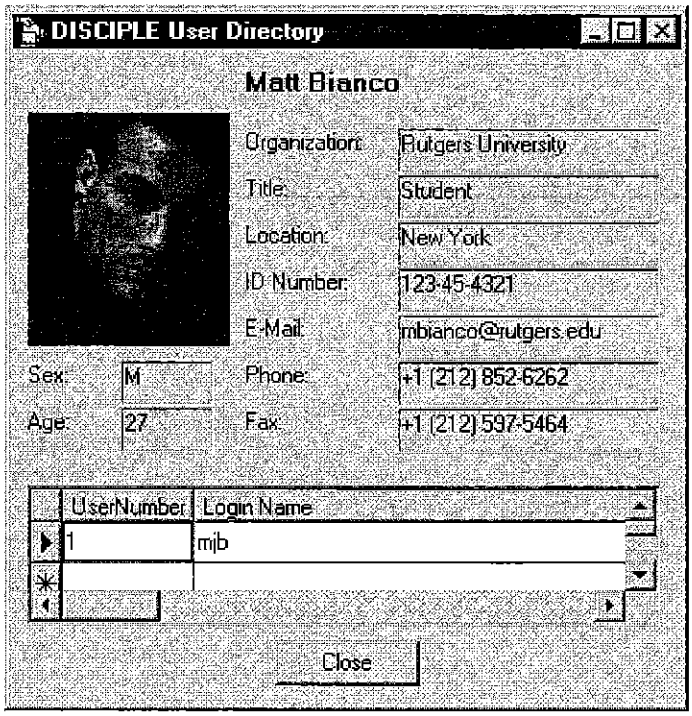

Figure 9 User Directory sample screen shot

Each user's record displays important personal and contact information on the upper portion of the form. The lower portion contains a data grid which holds all of the user's ID record from the database. This is primarily for development purposes, and may very well be removed in the production release.

The user directory is an important part of the EventStreams interface. It opens the door for participants to seek out those with overlapping responsibilities and interests. Ultimately, it could hold much more detailed data on each user describing this. Team members could then contact each other (either through DISCIPLE or by traditional means) to arrange for further cooperative work. This allows all participants to leverage

the knowledge of other collaborators, and the whole becomes more than the sum of its parts.

4.5 Security and Auditing

Many organizations (the military definitely among them) require a strong set of internal controls for all systems. An audit trail must be generated that will enable an investigator to recreate the sequence of events as they took place.

EventStreams can be used as a platform for examining past sessions to ensure that no improper activity has occurred. Every creation, modification, and deletion is duly recorded and can be reviewed at any time. Periodic audits can be performed, or a "post-mortem" can be undertaken after a situation has blown up.

The query functionality built into the EventStreams environment can be a great time-saver in this regard. It allows the auditor to quickly distill a mass of data down to just the interesting components, and effort can be spent where it is most useful.

5 Class and Database Structure

The EventStream system has been designed to easily integrate into DISCIPLE. It is written in Java to allow portability across multiple platforms. Its interaction with the other components, detailed above in Figure 3, is via the database created by the Archive Server. Further design details are given here.

5.1 EventStream Classes

The EventStreams environment is written entirely in the Java programming language. The application is comprised of a handful of major classes that provide it with the functionality described in this document. A brief overview is given here.

Class Name	Description	Important Methods
EventStream	This is the central class for the application, and contains the code for the Stream View. When an EventStreams session is started, an instance of this class is presented. Additionally, it contains the code for the application's main menu. Also handles maintenance of the	EventStream() – generates form, calls other methods to build the data structures BuildEventIconArray() – creates an instance of the EventIcon class for each event selected by the current

	information in the Detail Panel. One should keep in mind the fact that the Detail Panel shows information about the *event* (time, user, type of action, etc.), and not about the *object.*	query SetDetailPanel() – populates the fields of the Detail Panel with information about the selected event MenuItem_Click() – calls the functions listed in the application's main menu
EventIcon	Each of the pages in the event stack in the Stream View is an instance of an EventIcon. When a user clicks on an EventIcon, it displays detailed information about itself in the Detail Panel, such as time of event, user who executed it, session during which it occurred, etc.	EventIcon() – creates data structure EventIcon_click() – updates Detail Panel
SetView	Creates the query-by-example form that is used to specify the criteria for items to be included in the current substream. Also contains the logic to generate the SQL code	SetView() – generates form, retrieves data for QBE boxes

	that is sent to the database server to retrieve these records, and presents this in a window for review.	UpdateQuery() – examines user selection in QBE boxes and generates appropriate SQL code
TimeLine	Builds a graphical workspace on which the TimeLine view is drawn. It retrieves the icons associated with each object type in order to do this and plots them in the appropriate location. Smooth scrolling via the horizontal and vertical scrollbars on the form is handled as well.	TimeLine() – generates form, builds bitmaps for all sessions Tabstrip_SelectedIndexChanged() – determines which session is now active and displays it *Various*_Paint() – displays the appropriate bitmap, taking into consideration tabstrip selection and scrollbar positions
UserInfo	Retrieves whatever personal and professional information about any project participant is stored in the user directory.	UserInfo() – generates form, retrieves and displays all fields

	If a photograph is available, it will display that as well.	

These classes encapsulate all the functionality described in this document, and due to the object-oriented nature of the Java programming language, the application is portable across many environments and is easy to extend.

5.2 Relational vs. Object Databases

For the proof-of-concept stage, the EventStreams system has been developed to utilize a relational database. However, when a production-ready version is written it will likely utilize an object database, or a hybrid object-relational system.

A relational database is poorly equipped to handle the complex relationships which characterize the objects in a Java application. In an object database, the tight integration of code and data can be represented.[2] Many of the reasons for adopting an OODBMS for use within the DISCIPLE system can be found in the Archive Server document.[20]

Recently, hybrid object-relational systems have become available both commercially and from academic research projects.[1] Several of the large commercial database vendors have grafted object functionality onto their core systems. Informix, with their DataBlade technology, is a good example of this. Poet Software, with their namesake Poet database,

have come from the opposite direction: they've added relational capabilities onto an object database.

The hybrid model, I believe, will ultimately become the correct choice for DISCIPLE. In addition to serializing the actual objects themselves, other metadata needs to be stored (a description of the kinds of things that must be recorded are described in the next section). Thus, the strengths of both types of DBMS are required.

5.3 Current Database Structure

In order for the EventStreams interface to provide any useful functionality, as much information as possible should be recorded for each event. The more data that is recorded, the more sophisticated a query can the user make against the database.

There is a certain base level of detail that the system knows about any action performed against a component, regardless of whether the JavaBean is collaboration-aware or – unaware. However, if a component is collaboration-aware, it may be able to provide extra semantic information about a state change. It is important for the system to take advantage of any extra detail that can be provided by collaboration-aware Beans, since this will provide the user with a much richer and more informative experience. The database schema must be engineered with this in mind.

An example of extra semantic information that we might like to record would be, in the case of a trading system for an investment bank, limit violations. At the point when a trader entered into a transaction, the JavaBean would determine whether any applicable limits had been breached, and record this fact. This is specialized intelligence that a collaboration-aware object could make available to the EventStreams system so that it could be utilized later as part of the query criteria.

The entity-relationship diagram for the EventStreams database, as currently designed, is shown in Figure 10. It is really a skeleton at this point in time, but was built to accommodate future expansion.

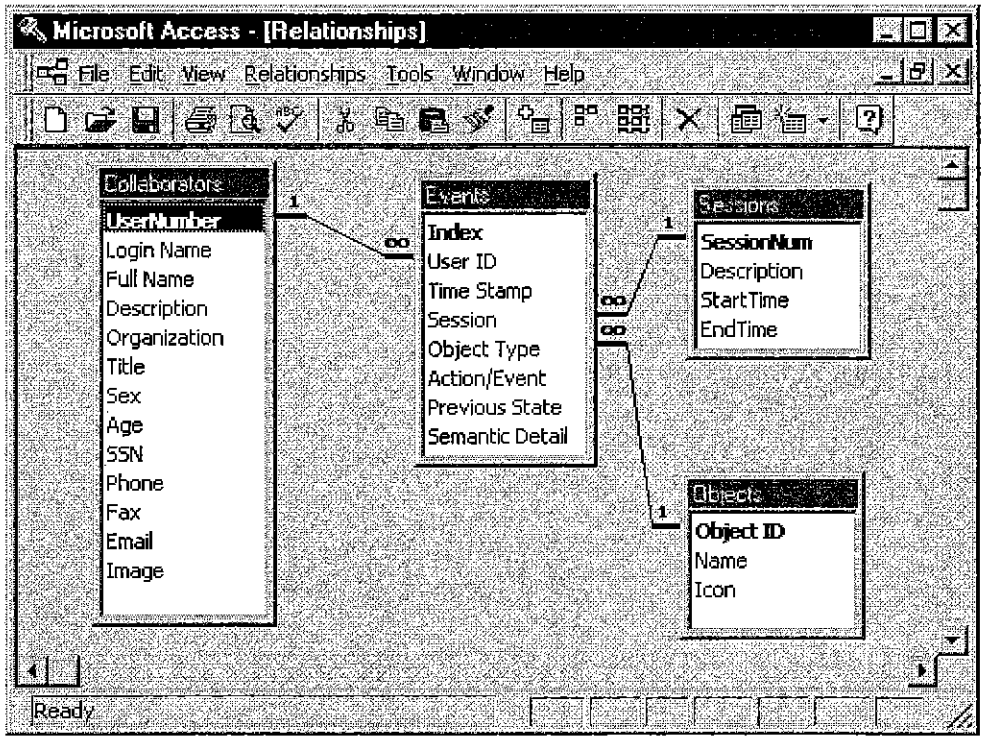

Figure 10 Entity-relationship diagram for EventStreams database

The following is a brief description of the detail that must be recorded about each event for it to be fully described within the system:

Property	Description
User ID	The login name of the user who performed the action. This information is linked to the user directory, which contains full details about all participants on the project team. In addition, this information is very valuable for auditing purposes (determining who performed a particular action, or who was involved in a certain sequence of events). Recording identities is often a prerequisite for system acceptance in any secure installation.
Date/Time Stamp	Exact time when an action was performed, synchronized to one standardized time zone (GMT, for example).
Session Name	Name of the collaborative session during which a given action took place. This is linked to a Sessions table which can hold further details, including start and end

	times.
Object Type	The class of the object upon which the event has taken place. This information provides the user the capability to create a substream consisting of only those events which were performed on objects in which he is interested. For example, the user can specify that he only wants to see events that involved objects of type "Plane" or "Bicycle" in the running example.
Action/Event	This is a description of the activity that actually has taken place. It is a notification that an object has been created or deleted within the workspace, or a description of a particular property which has been modified.
Previous State Information	It may be desirable to record the status of a component before a change is made. This would allow a user to query for situations where an object was modified from one particular state to another (which would otherwise be difficult to ascertain).
Semantic Detail (for collaboration aware objects)	If a Bean can provide us with additional detail about a given action, we should record this and make it available

	for later inspection. Give example This could include things such as calculated data (e.g. distance, time and fuel requirements) or "expert system" type notices (e.g. warnings about suspicious activity). This intelligence would be completely built into the JavaBean, and made available via the collaboration bus interface. Thus, it requires collaboration-aware JavaBeans.

The same pieces of information that have been stored for the proof-of-concept version will need to be accommodated in the completed release, even though it will likely be built using a hybrid object-relational DBMS as a foundation (as opposed to the purely relational prototype system used in this version).

5.4 JavaBeans and Introspection Features

The more information that can be obtained about a Bean in use within a collaborative session, the better EventStreams can represent its events. Luckily, the JavaBeans API

includes a set of introspection facilities by which an object can make its internal structure visible to any interested application.[12] Though introspection was designed primarily with application-building systems in mind, it has a wider range of uses.

These services can be used to determine, typically, the public properties and methods which are available to the inquiring application (though not limited to just these). This ability will be very useful within EventStreams, as it will allow the system to determine which Squish methods are built in. Thus, a Bean can support as many different Squish techniques as the Bean author wants to build in. For example, in the case where the EventStream represents a string of stock purchases and sales, the Bean author might want to write Squish routines to calculate profit and loss, draw graphs, or even try to detect insider trading patterns. The EventStreams framework would know about each of these through the Bean's introspection feature.

For example, a group of objects which represent empirically observed engineering data might contain compression methods that perform a statistical analysis, creates several types of charts, and flag values that meet certain criteria. EventStreams can use the introspection facilities to learn of these methods, and can therefore be very flexible in the way it displays objects.

6 Conclusion

DISCIPLE is a strong framework that provides collaborators with a means of communication and interaction. The EventStreams environment extends the functionality of this environment by providing participants with several new tools to help them rapidly internalize what has taken place thus far (even if they have been uninvolved to this point), and to understand the role of each of the other team members. These include:

- The Stream View, which is a representation of a set of actions, graphically depicted as a stack of pages. Details can be obtained by clicking on a given page.
- The TimeLine View, which displays less detail than the Stream View, but quickly imparts an understanding of who the participants were in a given session and their areas of responsibility/interest.
- Substreams, where large numbers of events are interactively sliced and diced into small portions that answer some specific question via a user-defined ad hoc query.
- The User Directory, which help team members find (and possibly contact) others who share a related area of interest.
- Squish technology, where a collaboration-aware component can contribute extra semantic information about itself.

These features fill a multitude of gaps in DISCIPLE, and help it to better address the challenges required of it by the DARPA contract specifications.

6.1 Limitations

The architecture of the system currently relies on all EventStreams sessions being able to connect to either a central database, or a local replica of the main database. This limitation detracts from the distributed nature of the DISCIPLE system, which is built to perform all communication over the collaboration bus via an object request broker.

At some point, the DISCIPLE system will need to be re-engineered so that it links into the collaboration bus as well so that the scalability of the system can be maintained.

6.2 Future Work

There are a few interesting areas which might be extended in EventStreams in future releases that will enhance its usefulness.

The most important of these is work on the addition of support for the "Squish" technology. This feature will have the most extensive impact on the system. The extent to which this is useful will be limited only by the functionality that can be built into a JavaBean (the Bean would need to be collaboration-aware). The summaries produced will provide a perspective on the activity which otherwise might not be apparent, even with careful scrutiny.

Collaboration-aware beans could also provide the Stream View with a visualization of the event. For example, if the Bean were a graphics editor, and the event were the drawing of a polygon, then the collaboration-aware editor could return an image of this. The Detail Panel of the Stream View could render this, making it abundantly clear exactly what that event represents. In more complicated events, perhaps a short animation could be returned to represent the action that took place.

Another helpful addition would be to add a comment feature to the system. EventStreams is currently read-only, but it might help a team to be able to add a "Post-It" type of note to one or more events in the database. These comments would then be displayed in all appropriate Stream and TimeLine views, allowing new participants to assimilate the situation even more rapidly.

Currently, the hooks are in place to allow the EventStreams application to spawn a replay session, but the links are not complete. It would seem that a rather minimal set of parameters would need to be passed (start and end times, along with a replay speed).

6.3 Summary

The DISCIPLE system developed at CAIP is a powerful groupware environment that gives geographically dispersed team members the ability to work together to create and

manipulate multimedia content. Up to this point, its significant limitation was that it did not provide a convenient way to access the wealth of contextual information generated by team's collaborative effort.

The EventStreams interface that I have implemented and described in this paper addresses this shortcoming by providing a range of intuitive, graphical views of the collaborative process. Its dynamic querying capability allows information overload to be avoided by distilling a large mass of data into small, useful, related pieces. I feel that EventStreams greatly enhances the utility of the entire DISCIPLE environment.

7 References

1. "A Comparison Between Relational and Object Oriented Databases for Object Oriented Application Development." POET Learning Center White Papers, 1997. http://www.poet.com/products/oss/white_papers/rel_vs_obj/rel_vs_obj.html

2. Bloom, Paul I. *Object Databases versus Universal Servers: Reality and Myth.* Volpe, Brown, Whelan & Co. 1997

3. "DARPA Project Summary: The DISCIPLE System, Rutgers University" 1997 http://www.caip.rutgers.edu/disciple/

4. Dorohonceanu, Bogdan and Marsic, Ivan. A desktop design for synchronous collaboration. In *Proc. Graphics Interface '99 (GI'99)*, Kingston, Ontario, Canada, pp.27-35, June 1999.

5. Fertig, Scott Freeman, Eric and Gelernter, David. "Finding and Reminding Reconsidered." Yale University Department of Computer Science 1995. http://www.cs.yale.edu/~freeman/papers/SIGCHI/paper.html

6. Freeman, Eric and Fertig, Scott. "Lifestreams: Organizing your Electronic Life." *Northwest Artificial Intelligence Forum Journal*, Vol. 6

7. Gelernter, David. "The Computer of the Future." Mirror Worlds Technologies 1998. http://www.mirrorworlds.com/horizons/index.html

8. Gelernter, David. "The Cyber-Road Not Taken." *The Washington Post.* April 3, 1994

9. Johnson, Steven. "Tech's Missing Link." *The Industry Standard.* November 2-9, 1998

10. Marshall, Patrick. "Taligent provides clean places for workgroup talks." *InfoWorld.* Vol. 19, Issue 42, October 20, 1997

11. Marsic, Ivan. DISCIPLE: A framework for multimodal collaboration in heterogeneous environments. To appear in *ACM Computing Surveys*, 1999.

12. "The MITRE Multi-Modal Logger." The MITRE Corporation 1997. http://www.mitre.org/research/logger/release/1.0/html/execsum.html

13. Morrison, Michael, et al. *Java Unleashed*, Third Edition. Sams Publishing 1997

14. Petreley, Nicholas. "Readers reject Band-Aids and believe Lifestreams will cure the broken UI." *InfoWorld.* September 29, 1997

15. Scholtz, Jean. "Intelligent Collaboration and Visualization." DARPA Information Technology Office. http://www.darpa.mil/ito/research/icv/index.html

16. Sener, John. "Current Educational Trends and Concepts, and their Relation to ALN." *Asynchronous Learning Networks Magazine*, Vol. 1, Issue 1, March 1997

17. Steinberg, Steve G. "Lifestreams." *Wired Magazine*, Vol. 5, Issue 2, February 1997

18. Sundaram, Sentilkumar. "A Collaboration-Enabling Framework for JavaBeans." Master's thesis, Department of Electrical and Computer Engineering, Rutgers University, New Brunswick, NJ, January 1998.

19. Wang, Weicong, Dorohonceanu, Bogdan, and Marsic, Ivan. Design of the DISCIPLE synchronous collaboration framework. In *Proceedings of the 3rd IASTED International Conference on Internet and Multimedia Systems and Applications*, Nassau, Grand Bahamas, October 1999.

20. Whitaker, Randall. "Computer Supported Cooperative Work and Groupware: Overview, Definitions, and Distinctions." 1989. http://www.informatik.umu.se/~rwhit/CSCW.html

21. Wu, Dawei. "Archive Server for Real-Time Collaboration in DISCIPLE." Master's thesis, Department of Electrical and Computer Engineering, Rutgers University, New Brunswick, NJ, October 1997.

EXHIBIT E

Collaborative Virtual Workspace Overview

Introduction

The Collaborative Virtual Workspace (CVW) is a prototype collaborative computing environment, designed to support temporally and geographically dispersed work teams. From a user's point of view, CVW provides a persistent virtual space within which applications, documents and people are directly accessible in rooms, floors and buildings. From a technical point of view, it is a framework for integrating diverse collaborative capabilities.

To a user, a CVW is a building that is divided into floors and rooms, where each room provides a context for communication and document sharing. CVW allows people to gather in rooms to talk through chat or audio/video conferencing and to share text and URLs with one another with their chat. (See figure 1 below.) Defining rooms as the basis for communication means that users are not required to set up sessions or know user locations; they need only enter a room. If users choose to communicate through audio, video or text, then the communication session is established automatically for them. Users can also lock rooms and communicate privately within and between rooms.

Rooms are also the basis for document sharing. Users can place documents of different types into a room, allowing anyone else in that room to read the document or view information about the document (such as creator, description, creation date, modified date, last modified by). Persistence is supported because the rooms exist even when no one is in them. Consequently, the document remains in the room for future visitors to see until some authorized user moves or deletes it.

Document types include whiteboards, URLs, notes and other documents edited through the user's local applications (e.g., word processor, spreadsheet). Documents that can be edited through local applications are managed through a document server within CVW. The document server provides a universally available file space (ensuring a document is available even if the owner's file space is not) and enforces single-user editing through document lock while editing. It also tracks information about who has edited a document and when, and allows each editor to summarize changes upon saving.

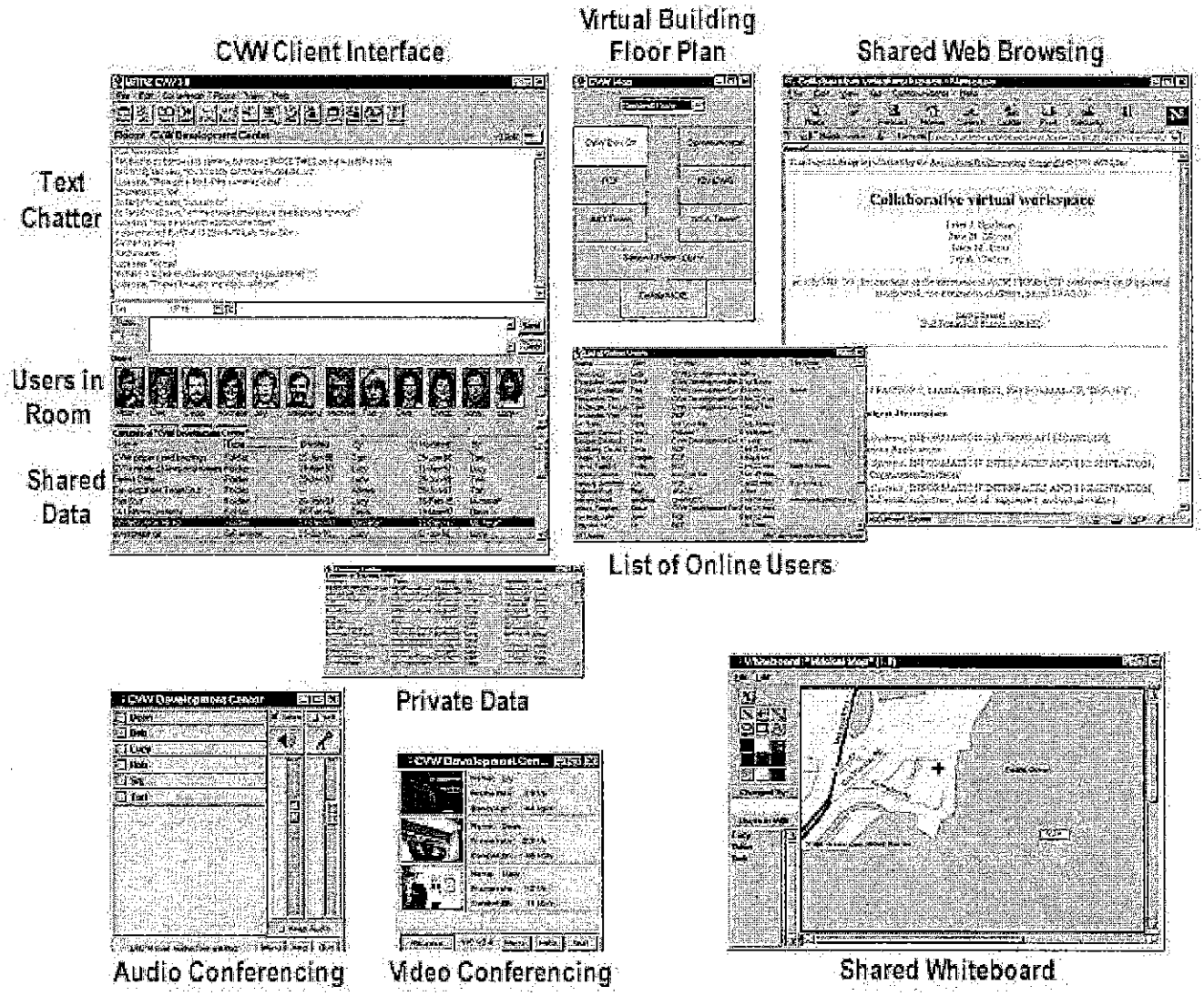

Figure 1. CVW Client Interface

Rooms and Floors

The metaphor of a building with rooms and floors provides the representations of the collaborative space within CVW. Each room provides context for collaborations within it. Floors and rooms may be named as appropriate, and room descriptions can be provided to provide additional context for the room. A graphical "map" display of the floor plan is provided to help visualize the collaboration space. Users can navigate through the virtual building with the graphical map or with a textual command. Additionally, users can navigate by joining a specific user, without knowing their current location in CVW.

Rooms contain people and objects. The types of objects CVW supports are:

- folders
- room recorders (e.g., note takers) that record public text chatter that occurs within the room
- notes (simple text notepad)
- documents (e.g., word processor, spreadsheet, graphics, binary, postscript, URL)
- whiteboards (shared annotation surface)

Room Access

CVW provides the ability to restrict room access, based on an access control list. Often, the access control list is defined by a "group", although specific individuals may be added to the room's access control list. For example, a group can be created (and assigned a set of users) and assigned access to a particular room, and any users not belonging to the group are restricted from accessing the room.

Text-based Communications

CVW's textual communications capabilities are extensive, providing the ability to communicate and express one's self in a manner similar to verbal communication. The textual communications capabilities that CVW provides include:

- the ability to direct communication from one person to another or from one person to a group of people
- the ability to privately communicate with people in the same room or another person in another room
- the ability to express in a 'non-verbal' manner (e.g., John nods) to all people in a room or privately to another person
- the ability to paste a text selection from another application to the people in the room
- the ability to privately paste a text selection from another application to a particular person
- the ability to send a URL (web page reference) as a hot link to the people in the room (clicking on the hot link in the room scrollback opens your web browser and automatically displays the web page)

Audio and Video Conferencing

CVW provides multipoint audio and video conferencing capabilities. The audio and video conferencing is self-configuring on a per room basis, providing conferencing capabilities with other the other people in the room. Users do not have to establish conference sessions or know other users' locations to use audio and video; they need only enter a room. CVW also provides a phone capability for private audio discussions between two people.

Room Scrollback

All interaction that occurs within a given room in CVW is displayed to the user in the textual scrollback window. This includes all text communication and activities which occur within the room (e.g., notices of when people enter and leave the room, notices of when someone places a document in the room). Users can customize the appearance of the text that appears in their individual scrollback by color-coding it. A users can set different colors for:

- text chat contributed by a specific person
- text chat directed to the user
- private chat directed to the user
- specific text expressions

CVW also provides the ability to save the text scrollback or a selection of the text scrollback and export it to a file. CVW also provides the ability to perform basic searching of the scrollback window contents.

Room Recording

CVW provides a room recorder (e.g., note taker) capability which captures the public text communications that occur within a room. Users have the ability at any time to create a personal transcript of the current recorder session.

Data Objects

CVW supports the creation and sharing of a variety of data, including notes, documents (e.g., word processor, spreadsheet, graphics), URLs, folders, and whiteboards. Users can import files into CVW's shared document space to share with other team members. Data objects in CVW can be manipulated in a variety of ways, including:

- the ability to set an access list on the item to control who has edit permissions
- the ability to place a reference item in a room and lock it down so that no on can take it
- the ability to move an item between the room and their personal (private) folder
- the ability to give an item to another person in CVW
- the ability to copy any item to which the user has permissions

Shared Whiteboard

CVW provides persistent shared whiteboards that enable multiple people to view an image (such as a map) and annotate the image together in real time. The whiteboards do not disappear after the session ends, and can be opened and re-used to continue the work process. Whiteboards maintain attribution of the annotator, so it is easy to see who marked up the whiteboard surface and how. The contents of whiteboards can be printed or exported to a file so that it can be included in another product such as a report.

Locating Users, Rooms, and Data

With many users navigating around the many rooms that can exist in CVW, it can be difficult to know how to find someone, know if that person is available, or where that person left a document. To simplify finding people, rooms, and data, CVW provides:

- a search capability that allows the search of different document types, rooms, or people based on name matching (All matches are displayed in a window with descriptive information about each match.)
- the ability find out which users are currently logged in, how long since they have been active, and their current location within the CVW

Proxies

Proxies enable people to be in two rooms at the same time in the virtual building. A person can place his/her proxy in any room that they have permissions to enter. Through the proxy, the user can

communicate textually and share text and URLs with anyone in the proxy's room. If more interaction is required, users can quickly switch places with their proxy and use the other features of CVW in the proxy's room (e.g., audio/video conferencing, documents).
