

## A Desktop Design for Synchronous Collaboration

Bogdan Dorohonceanu and Ivan Marsic  
{dbogdan, marsic}@caip.rutgers.edu

Center for Computer Aids for Industrial Productivity (CAIP)  
Rutgers — The State University of New Jersey  
Piscataway, NJ 08854-8088

### *Abstract*

This paper presents a novel graphics user interface for desktop management of a synchronous groupware client. The interface is part of the Rutgers University DISCIPLER framework that enables sharing of applications. The interface presents an individual view of a collaboration space that contains collaboration artifacts, collaborators, their groupings and relationships. The conceptual model of the collaboration process is described since it strongly influences the design of the user interface. We establish the requirements, describe the components of the user interface and then discuss alternative approaches. JavaBeans applications are shared by being imported into the shared workspace, but additionally, importing Beans allows user tailoring of the interface and thus supports end-user programming. Interface customization is demonstrated with multimodal human/machine interfaces and the collaboration components (such as group awareness widgets, concurrency controllers, etc.). Another activity supported is multi-user visual programming using the JavaBeans technology. Users at geographically separate locations can collaboratively build complex applications using pre-existing components. This interface has been implemented and tested on a variety of Java applications.

*Keywords:* CSCW, synchronous collaboration, user interfaces, end-user programming, JavaBeans.

### **1 Introduction**

A user interface for synchronous groupware should allow the collaborators to quickly grasp the contents of the collaboration space, and the relationships between the collaborators, as well as allow them to communicate and share applications and data. The interface must be capable of presenting the following essential components of the collaboration process at any time during the collaboration: representation of and access to the set of collaborators and their groupings, artifacts used to accomplish the collaborative tasks, and images of the working or meeting places.

Additional components that are not directly related to collaboration, but are necessary for collaboration, are

the tools for discovery of collaborators and the tools for reviewing the past collaborative sessions.

The main objective of DISCIPLER (*D*istributed System for Collaborative Information Processing and *L*earning) project is application sharing. The conceptual collaboration model strongly influences the design of the user interface. An appropriate theoretical model of work and workgroups is thus of critical importance and the model used in our work is reviewed in Section 2. The guiding principles for the design and implementation of graphics user interface for synchronous collaboration presented here are simplicity and familiarity with the existing concepts of graphics interfaces (mainly for single users). The interface is part of our DISCIPLER framework that provides mechanisms to control its cooperative features in an application-independent manner.

This paper is organized as follows. We first briefly review the DISCIPLER collaboration framework. Section 2 presents our approach for collaboration space visualization and management. Section 3 continues with the collaboration artifacts and tools we provide. Sections 4 and 5 describe our implementation for the user interface of a meeting place and design-time collaboration, respectively. Finally, we discuss the related work of the existing collaborative frameworks (Section 6) and conclude on the contribution of our project in its current state of development (Section 7).

#### **1.1 DISCIPLER Collaboration Bus**

The graphics user interface presented here is the client view of the DISCIPLER system [9]. The objective of the DISCIPLER project is to simplify the development of multi-user collaborative applications, in particular real-time synchronous groupware. It includes conversion of single-user applications to multi-user domain with minimum effort—possibly without modifying the original code or requiring the user to write additional code. This is achieved by relieving, to the largest degree possible, the application programs from performing collaboration tasks, that is, those tasks that must be performed to allow more than one user to interact with these programs. We target a particular class of applications—Java applets and more generally, Java Beans [15]. The DISCIPLER framework is

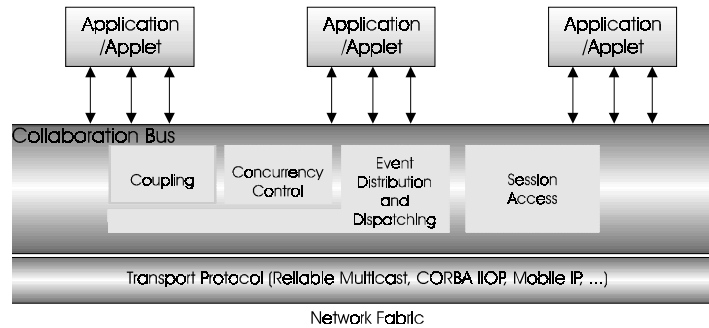


Figure 1: Architecture of the collaboration framework. Each conference participant runs a replica of the application. The arrows symbolize the user's actions that cause the transitions of the application's states. The collaboration bus provides various group- and communication-related services.

conceptualized as a collaboration bus (Figure 1), which assumes the collaboration tasks. The user plugs the applications or applets into the bus by pointing to their URLs and thus makes them shared with the other users that participate in a collaborative session. An applet is a small application program that provides a graphics user interface and accomplishes a user task (e.g., extracting image features or computing and visualizing a spreadsheet table). A user would typically use multiple applets, independently or linked into a more complex applet, to collaborate with other users. Since multiple users can simultaneously interact with their respective applications, the interactions need to be coordinated. The bus therefore contains the software components that manage synchronous group work (e.g., concurrency control of simultaneous activities, degree of sharing the application (coupling), and degree of awareness about the originators of the activities).

The DISCIPLER collaboration is based on a replicated architecture for groupware [5]. Each user runs a copy of collaboration client, and each client contains a copy of the applet that is to be collaborated on. For each object within the applet, there will be a counterpart in all the other users' applets. Collaboration in this type of architecture essentially translates into intercepting the state changes occurring in a user's applet and replicating the state changes in all the peer users' applets. Event interception is based on the property of Java Development Kit (JDK) version 1.1 event model. Any object can register as an event listener to the event sources and all listeners get notified upon the event occurrence [15]. When an applet gets loaded into the shared workspace, the workspace registers to listen for all the applet's events. Upon the event occurrence, the collaboration bus distributes the event to all peer workspaces. The bus achieves synchronous collaboration through real-time event delivery, event ordering and concurrency control. A more detailed description of the DISCIPLER collaboration bus is available in [8].

## 2 Collaboration Space Visualization

### 2.1 Meetings vs. Places for Collaboration

From the user's viewpoint, the act of synchronous collaboration may emphasize either the meeting itself with no concern for the location or the place where the meeting occurs. Human collaboration includes both meeting places with specialized resources to support the meetings, as well as spontaneous discussions that occur regardless of the location and often do not need special meeting support from the environment. Places contain persistent content and may get more elaborate and complex with time.

Our current model emphasizes meeting places, where the user selects a place and, by entering it, obtains access to the resources in the place. The shared workspace window gets automatically launched as the user enters a place. The model is very similar to the rooms models described in [6, 13] as well as to the locales model in [10]. A place, however, in our model is linked to the *topic* of collaboration rather than to the physical *location* of the place.

Although this model does not preclude spontaneous collaboration, it is more difficult to establish a spontaneous meeting since the model is not tailored for such type of collaboration. In fact, the concept of places conflicts with spontaneous collaboration since it requires the conferees to meet at a *specific place*. The lack of spontaneity is in the sense that users cannot start sharing any application on the screen at any time. Rather, the user needs to enter a place and use applications therein or initiate new ones. This is different from collaboration with a set of formal rules (workflow) vs. an informal meeting.

One of the reasons for inadequate support for spontaneous meetings in DISCIPLER (as well as other Java-based approaches) is due to the way the Java Virtual Machine (JVM) is currently implemented. JVM runs as an application and it affects neither the window

manager nor other applications. Unlike this, collaboration systems based on shared window systems (e.g., [5]) get interposed between the user and the operating system—the entire user interaction goes through the window system. This provides for sharing the entire user’s desktop, and any application can become collaborative at any moment. An equivalent to a shared window system would be a shared JVM running on a Java Operating System. DISCIPLÉ is not a shared JVM—it is an application running in JVM, several levels above the window system. Spontaneous collaboration is hindered by the fact that nothing but the application Beans that run in one of the DISCIPLÉ workspaces can be shared.

## 2.2 Hierarchical Space Visualization

The *collaboration space* is essentially a “phone book” of all people that a user can collaborate with. It is structured to reflect various people groupings as in everyday life. It may be visualized in different ways, for example, using an abstract model or using a physical model. The space may be represented using an abstract model or using a physical model. The simplest abstract representation shows a plain list of collaborative places. A more complex representation structures the places into a tree or a graph, where the nodes at higher hierarchical layers correspond to buildings, cities, etc. Even more complex representation positions similar places proximally according to certain distance measure. On the other hand, in a physical representation the space is represented as a 3D virtual world, where the user walks through streets and corridors to reach a collaboration place. This representation is more intuitive than the abstract ones, but its disadvantage is that the user can be in only one place at a time. The length of time to move from one

place to another depends on their distance. In addition, creating new places can be fairly difficult. Since the places in our model are characterized by the topic of collaboration, rather than by their physical location, an abstract representation is more appropriate than a physical one. As a result, the user can simultaneously be in several places (think about several topics) and can quickly move from place to place (switch form topic to topic through the mind’s “hyper-space”).

The main window of the client part of the DISCIPLÉ system is shown in Figure 2. This is the first window that appears after the user logs into the system. The window visualizes the information about the collaboration space as present at the available place server(s). The hierarchy is as follows: place servers > organizations (hyper-places) > (meeting) places > users (participants) and is represented with a tree, shown on the left side of Figure 2 (similar to a file system representation). For example, in Figure 2 one can see the communication node (*disciple*) with one organization (*CAIP*) that contains two meeting places (*Military* and *Medical*). The users that participate in meetings (*boi*, *dbogdan*, *latha*, and *juth*) are shown in places, and there is one user (*francu*) that has connected but presently is not participating in any meeting. Using the main window users can navigate the collaboration space, create new meeting places, or enter existing ones. This current representation is space-centric rather than user-centric, i.e., the entire tree has to be inspected to find out what places a particular user has entered.

A place server supports public and private places. Public places are grouped under organizations and users are grouped under the places they have entered. A user can enter multiple public places at a time. Only its creator can enter a private place, and other users are not aware of its existence.

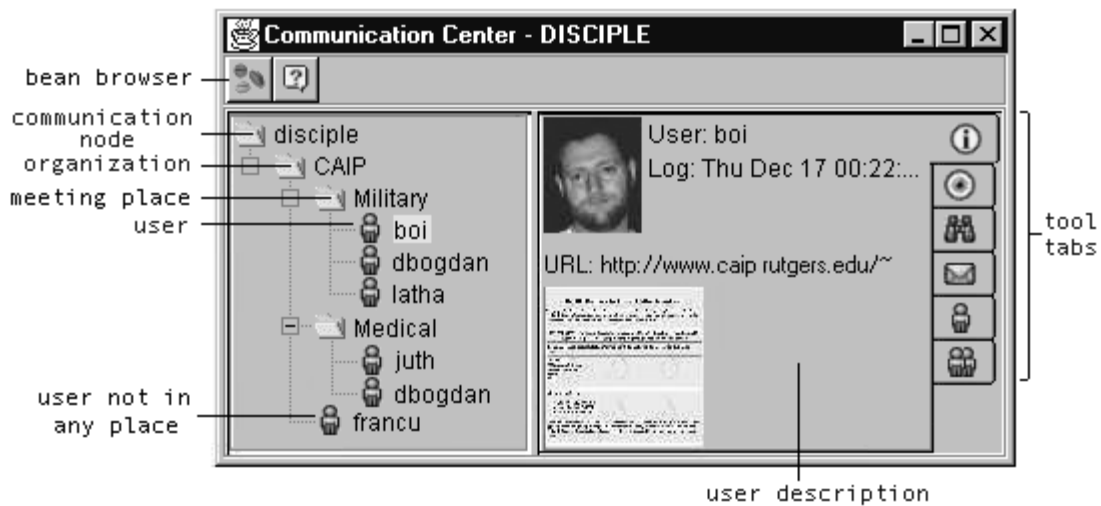


Figure 2: Screen snapshot of the hierarchical representation of the entire collaboration space. Each organization, place, or user is associated with a description (image and/or text).

On the right side of Figure 2 there is a stack of pages representing different utilities available to the user. Current snapshot shows a description of or information about the selected item. The description is provided for only one item at a time. A different approach has been put forward in [10] that shows simultaneously a short description of all places that the user is involved in; however, the hierarchy of the entire collaboration space is not visualized. A way to have both is to have (on the right in Figure 2) a scrollable window with the description of all places the user is currently in. The information item is shown in a tabbed pane that also contains other common tools like chat window, video/audio conferencing startup, e-mail, query window for finding users, etc.

### 2.3 Place Creation, Entering and Leaving

Creating and entering a place is easily done with a mouse click—there is no requirements to specify any network address or a port for connection. A user creates a public place in an organization by a right-button mouse click on the organization's icon (left side of Figure 2). The user is prompted to enter the place description via a dialog box. As the view of the meeting place is created, the user automatically enters the place. A user enters a public place by clicking the place's icon. A workspace (Section 4.1) is an ephemeral representation on user's desktop that corresponds to a place on a place server. It is generated automatically as the user enters a place to display the current artifacts in the place. A workspace is bound to a place and cannot be re-bound to another place. As the user leaves the meeting place, a private place (snapshot of the meeting) gets created and filled with the artifacts currently in the workspace. The same workspace now represents the view for the private place.

The user can specify a list of users for whom s/he is interested in being notified when they are present in the collaborative space. A message dialog window announces them whenever they log in. A user can search for and invite other users to enter his/her public places, or specify a rule/filter for automatically inviting users to enter his/her places. When a new user enters a place the collaboration space hierarchy is updated with his or her icon and descriptive information so that all place inhabitants who have the place's folder open can notice the change. We are currently exploring a means (an audio or short visual message) for notifying the existing inhabitants of a place about the newcomer.

Places are persistent, so when a meeting gets suspended (e.g., the last user leaves the meeting place), the content of the place gets automatically stored on the server, so the meeting can be resumed in the future starting with the same content. Persistence provides for asynchronous collaboration.

### 2.4 Individual Desktop View

The overall desktop view can vary for different conferees, i.e., it is not a strict WYSIWIS (What You See Is What I See) mode [14]. All of the collaboration desktop windows (i.e., the communication center window and meeting place windows) are freely floating on the user screen and are individually customizable. An alternative that we explored earlier was to have all windows glued together as children of the main window. This has some advantages of being able to move or iconify all windows simultaneously. On the other hand, this solution potentially uses too much of the screen space and constrains the user as to how to arrange the windows on the screen. Notice, however, that this principle is not held all the way to the level of the artifacts (Java Beans). All the artifacts in a place are shown in a single workspace window corresponding to a place as will be seen in Section 4.1 below (unless a Bean launches its own window).

Help is provided through a small HTML browser that visualizes the project documentation available in the HTML format from our Web site. This way, the users can access the most up-to-date help information.

### 3 Collaboration Artifacts and Tools

The users collaborate by sharing artifacts, tools and resources. In DISCIPLE these are software applications of one type—Java Beans [15]—which can be Java applets or applications. The DISCIPLE framework supports both collaboration-aware and collaboration-transparent Beans. Collaboration-transparent Beans are usually developed by a third party and they are completely unaware of the fact that they are interacting with multiple users.

*Bean Browser* displays all the Beans that are available to the user at launch time or have become available during a collaborative session, see Figure 3. User's own Beans, located in the local directory `jars`, will be loaded by default when the user logs in. Other Beans may be acquired during the meeting. At present, the system automatically distributes the Beans to the other participants so that all users in a place have the same set of Beans. The Java Beans are packed in Java archives (JAR) for fast transmission over the Internet. When a user acquires new JAR files or enters a place having JAR files that are not available in the place, they are dynamically loaded through the network and added into the Bean browsers of all the users in the place.

Using a Bean in collaboration is as easy as pointing to its URL. A Bean can be loaded from a local file system or, given the Bean's URL, it can be loaded from any Web server. The loader loads a JAR file that may contain multiple Beans, e.g., the multimodal JAR in Figure 3 contains the Beans for speech recognition,

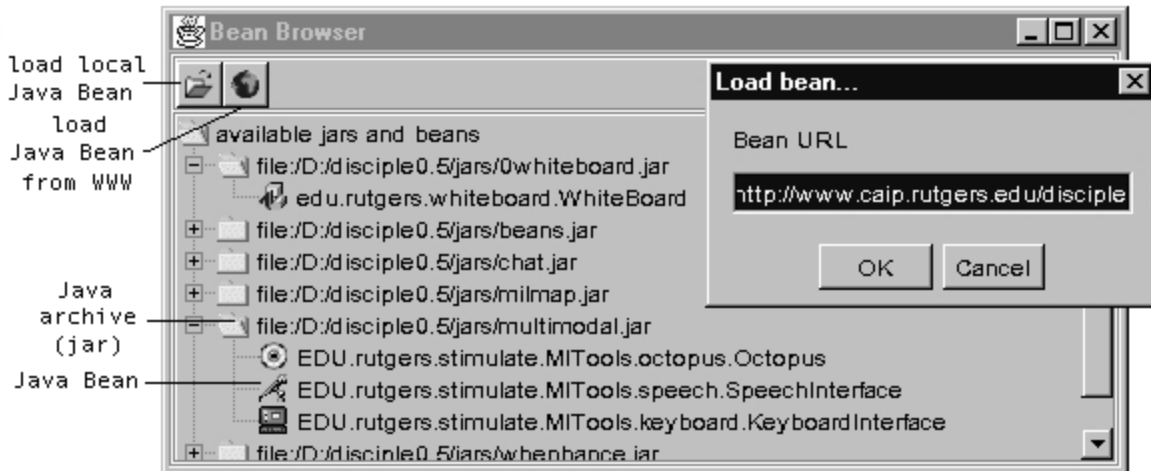


Figure 3: Screen snapshot of the user interface for loading and browsing the Beans.  
The loaded Beans can be used in collaboration.

keyboard and pen interaction, and multimodal data fusion (octopus). Ordinary Java applets can be shared in the DISCIPLE framework as long as the applet is packaged in a JAR file.

A user can drop a Bean in a workspace by selecting it in the Bean browser. All the users in the corresponding meeting place will see the Bean appear in their workspaces. The users can also copy the Beans from a workspace to another workspace. This operation maintains the Bean's current state. This way, the user will often draw from the relevant places the things they need, as they need them, to get their tasks done. The interface provides for relatively seamless movement between places.

The *Property List* tool introspects automatically the selected Beans and finds out their exposed properties and the associated editors. Using the Property List Editor, the user can visually modify the properties of the Beans dropped into the place, all updates being automatically sent to other users in that place.

The user interfaces of the Java Beans are expected to be written using the Swing toolkit of JDK 1.2 as the GUI of the DISCIPLE framework is built with Swing, since this package provides for setting the look-and-feel of the applications without changing their code.

## 4 Collaboration Workspaces

### 4.1 Workspace View

A workspace is launched automatically as the user enters a meeting place. It provides an individual view of the place, showing the artifacts that are currently in the place and their relationships (Figure 4). The toolbar at the top provides for opening the Beans browser, loading the Beans, and cut/copy/paste operations on Beans. Clicking the second toolbar button loads into the

Workspace the Bean that is currently selected in the Bean browser.

As is the case with the overall desktop view, the Workspace also works in a relaxed WYSIWIS mode in the sense that each conferee can position the Beans at different locations within the Workspace. However, Beans themselves are presently limited to consistent views and actions for all the conferees, because it is not possible to uncover the entire semantics of the collaboration-transparent Beans. A Bean can be displayed in a Workspace window or it can launch its own independent window.

Each place is dedicated to one topic having its own workspace. This enables the users to both run and/or build new applications incrementally in a place at runtime and allows for interface customization, as discussed below. A well-known paradigm is that a user has a single workspace for all the locales that he/she participates in [10]. The user resides in one "office" while he/she can be working on several problems simultaneously; thus one workspace for multiple meetings. Orbit has an icon and a label (indicating the meeting it belongs to) identify each artifact in the workspace. This representation is suitable since Orbit deals with documents represented by icons, but not with the document editors and viewers, i.e., applications. Unlike that, the main objective of DISCIPLE is sharing of application, and place labeling of the application windows would be inconvenient.

### 4.2 Collaboration Components

The idea of collaboration components is that, even in specialized collaborative applications, a common set of functions exists that can be extracted and implemented as self-contained software components. These include such things as concurrency control, user awareness, etc.

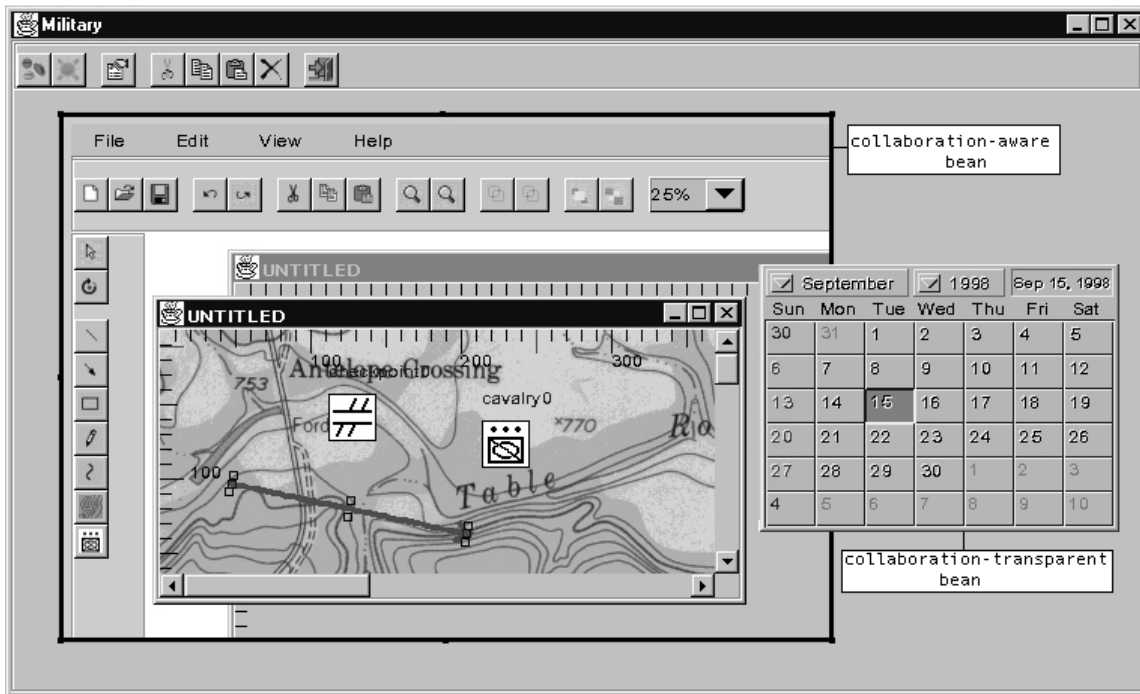


Figure 4: Screen snapshot of the workspace. The workspace contains two examples of Java Beans: a (collaboration-aware) whiteboard and a (collaboration-transparent) calendar.

It would be ideal to enable an application developer to focus on the application itself and not on these multi-user aspects. By providing a set of reusable software components that encapsulate such multi-user features, the collaboration components allow the user to easily configure the shared workspace [4].

The DISCIPLÉ framework currently provides awareness through telepointers and radar views that are included in the set of Beans loaded by default at start-up. A *telepointer* represents the position of a remote user's mouse cursor, providing location awareness. It has unique color, and a name, and both identify the user. A *radar view* displays a miniaturized image of the workspace with highlighted regions that the participants are currently viewing. Collaboration components also include several concurrency control algorithms (non-optimistic locking, optimistic locking, and undo/redo) [4].

### 4.3 Interface Customization

Componentization of the collaboration framework introduces the possibility for interface customization. Figure 5 shows an example where the workspace is augmented by collaboration component Beans [4] and by multimodal human/machine interaction Beans [11]. By loading different collaboration component Beans, the user can vary the degree of awareness about the other conference participants or select the concurrency control algorithm that applies to a particular Bean. Similarly, by loading and activating different

multi-modal Beans, the user can dynamically choose the modality for interacting with the workspace (e.g., speech, keyboard, eye gaze pointer, etc. [11]).

The users can also edit their description (picture, personal information, URL address of the Web page, etc.) and the properties of the awareness widgets representing them remotely (such as telepointers, radar views) at any time during collaboration.

## 5 Design-Time Collaboration

A special feature of our framework, which is not present in other collaborative frameworks, is support for design-time collaboration, where users can customize at run-time the application or compose more complex applications from simpler ones. A Bean is a software component with a well-defined set of inputs and outputs and the outputs of a Bean may be fed as inputs to another Bean. *Design-time collaboration* is the act of collaboratively assembling a composite JavaBeans application from several JavaBeans components. Once the application is composed, all the state changes in the application will be shared. This stage is the *run-time collaboration* of applications.

The DISCIPLÉ collaboration-enabling framework is a simplified Java Integrated Development Environment (IDE), augmented with a collaboration bus and the mediators between the bus and the applications. The framework is not intended to be a full-featured IDE. The design-time collaboration feature is intended for user-extensibility and incremental programmability of



Figure 5: An example of workspace customization. The workspace contains the task Bean (whiteboard), some of the collaboration component Beans (user actions view, radar view, and telepointers), and some of the multimodal interface Beans (speech and text input).

collaborative applications, rather than as a development tool. It is provided for advanced users to be able to rapidly add extra features to an existing application or to prototype an application using simpler components. Although we do not expect that users will frequently use this feature, we believe that it can be very valuable on occasion. System administrators or other super-users who do not see themselves as programmers would use this feature and save the results for future use by ordinary users.

Design-time collaboration can be compared to the macro language for spreadsheets, where the advanced users are able to create powerful macros to carry out complex and useful work, without the need to require upgrades and modifications to the application from the original developer.

Although workspaces are hierarchical, linking (using the JDK event model) the Beans from different workspaces or hierarchies is flat. Any Bean can be linked to any other Bean in any of the workspaces on the user's desktop. The linking can be accomplished while the Beans are in one workspace, and then they can be dragged to different workspaces. It can also be

accomplished when the Beans are already imported into different workspaces.

## 6 Related Work

Traditional and currently available commercial user interfaces for synchronous collaboration usually include simple whiteboard and audio/videoconferencing<sup>1</sup>. Conventional systems for sharing collaboration-transparent applications, like Microsoft NetMeeting [7], are also lacking in terms of support for key groupware principles: concurrent work, relaxed WYSIWIS, and group awareness. Few systems that do deal with general application sharing are available only as research tools and the design principles for the user interface are rarely explicitly formulated. Some general guidelines for developing multi-user interfaces are discussed in [3], but design for specific components of a multi-user interface is not described.

The model of collaboration presented here has certain similarities to the *locale* concept and the *Orbit* implementation of the concept [10]. In the *Orbit*

<sup>1</sup> See e.g., MBONE multiparty conferencing tools available at <http://www.mbone.com/mbone/software.html>.

framework a user has a single workspace for all locales (i.e., places) in which he participates. The paradigm is that the user resides in one place (e.g., office) while he/she can be working on several problems simultaneously; thus one workspace for multiple locales. Artifacts are represented in the workspace by icons and labels associated with the locales they belong to. Orbit focuses on visualizing the content of locales, with some information about the artifacts' states, but it is not concerned with the viewers and editors for interaction with the artifacts. Unlike this, as we discussed in Section 4.1, DISCIPLE focuses on application sharing, i.e., viewers for Java Beans that allow interaction with the Beans, distribute the Bean events, and provide customization of the user interfaces. Therefore we proposed a different approach.

*GroupKit* toolkit [12] is used in groupware prototyping for experimenting with different types of group awareness and modeling shared applications (called conferences) such as drawing tools, text editors, and meeting tools. GroupKit intends to support different types of session managers, tailored to the group needs to create, locate, and join conferences. The "open-registration" session manager uses small separate dialogs to represent and modify the information about conferences and participants. A shared application can be selected from a list and, once opened, it has its own window on the screen. The "room-based" session manager represents the rooms as icons in a collaboration space. A room groups several shared applications (a set of tools needed to accomplish a task). As the user enters different rooms only the tools from the current room are displayed on the screen. Our framework uses a more compact representation through a tree hierarchy. GroupKit has several awareness widgets particularly relevant to relaxed WYSIWIS groupware awareness (identity, location, and actions) and offers telepointers, radar-views, location and action viewers, and multi-user scrollbars, most of them implemented for the "open-registration" approach.

*Collaboratory Builder's Environment* (CBE) [6] provides both a toolkit for creating collaboration environments and a complete collaboration system. To support extensibility, the client application software is a structured set of applets, which includes services (such as data viewers) and tools (such as multi-user chat, whiteboard, mail, audio-video conferencing tools) for collaboration. CBE uses *rooms* to partition the shared workspace consisting of multiple applets and data sources represented by URLs. The session manager supports one shared workspace, which corresponds to one organization in our notation, and keeps information about the users and objects in rooms, represented by a tree-like hierarchy. Our framework goes further by

supporting multiple organizations (each of them containing multiple places/workspaces and users). In CBE, a user can dynamically add new applets and data to their rooms and move both tools and data between the rooms. CBE offers support for user roles in a room (observer, member, administrator, restricted) and access control for use over the network. The shared workspace may contain applets, users, applet-groups, and rooms. An applet-group denotes an applet shared among the users in a room. Rooms are persistent and provide for synchronous and asynchronous collaboration. CBE relies on the Web browser in order to display the session manager, the rooms, and the applications (applets) in separate windows.

The *Habanero* framework [2] provides support for sessions, similar to rooms in CBE, which can contain URLs of data sources and collaboration-aware applications. A session requires predefining the set of applications to be shared and does not allow dynamic adding of applications. Unlike this, DISCIPLE allows for moving Beans or active applets among places and organizations.

*Flexible JAMM* [1] supports sharing of single-user Java applets in synchronous collaboration. It relies on a custom-modified version of JDK 1.1, which makes it non-portable. JAMM primarily targets *unanticipated sharing* or spontaneous collaboration where a user is able to initiate sharing at any time during the execution of the application, not only before application is started. Due to the specific collaboration model, JAMM user interface does not represent the collaboration space. The workspace accepts only one applet at a time, but offers a radar view and different types of telepointers.

Although some of the components of the design presented here exist in other systems, end-user programming (through the run-time customization of the user interface and design-time collaboration) is to our best knowledge unique to our framework.

## 7 Conclusions

This paper has presented a design and implementation of a graphics user interface for synchronous collaboration that allows interaction with shared JavaBeans applications. The interface is part of the DISCIPLE framework that provides mechanisms to control its cooperative features in an application-independent manner. The interface is not a complete, monolithic system but rather a component system that is open and customizable (to the context in which it is placed and particular user needs).

Our project is still evolving and there are unsolved problems that we are currently investigating. For example, if an application (by mistake) exposes chained events, where an event causes the next event in the

chain, the collaboration bus is not able to detect this, and causes duplicate events in remote applications. An example is when in design-time collaboration a collaboration-aware application is chained after any other application. Another issue is to what class an application belongs when it is composed of two simpler applications, one collaboration-transparent and one collaboration-aware. This is more than a philosophical issue, since at the time of Bean loading the framework registers different event adapters based on the application type.

The DISCIPLER framework developed to date has been implemented and tested on both collaboration-transparent Java Beans (available on the WWW) and our collaboration-aware Java Beans. The applications include whiteboarding, collaborative mapping, speech signal acquisition and processing, and image analysis. We are currently running human performance experiments to evaluate the interface design (as well as other aspects of the framework). For updated information or software download, check this: <http://www.caip.rutgers.edu/disciple/>.

### Acknowledgments

The authors had many inspiring discussions with Cristian Francu, Stephen Juth, Boi Sletterink, and Weicong Wang who contributed to the implementation of the DISCIPLER framework. Marilyn Mantei Tremaine and Jeff Hand of Drexel University helped with early evaluation of the user interface. The research reported here is supported by DARPA Contract No. N66001-96-C-8510, NSF KDI Contract No. IIS-98-72995 and by the Rutgers Center for Computer Aids for Industrial Productivity (CAIP). CAIP is supported by the Center's Corporate Members and by the New Jersey Commission on Science and Technology.

### References

- [1] J.B. Begole, C.A. Struble, C.A. Shaffer, R.B. Smith. Transparent Sharing of Java Applets: A Replicated Approach. In *Proceedings of the 1997 Symposium on User Interface Software and Technology (UIST'97)*, pages 55-64, 1997.
- [2] A. Chabert, E. Grossman, L. Jackson, S. Pietrowicz, C. Seguin. Java Object-Sharing in Habanero. In *Communications of the ACM*, 41(6):69-76, 1998.
- [3] P. Dewan. Principles of Designing Multi-User User Interface Development Environments. In *Proceedings of the IFIP TC2/WG 2.7 Working Conference on Engineering for Human-Computer Interaction*, pages 35-50, 1992.
- [4] S. Juth, Collaboration Components for Programming Real-time Synchronous Groupware Applications, Master Thesis, ECE Department and CAIP Center, Rutgers University, 1998. (At <http://www.caip.rutgers.edu/disciple>).
- [5] C. Lauwers. Collaboration Transparency in Desktop Teleconferencing Environments. Ph.D. thesis, Technical Report CSL-TR-90-435, Computer Systems Laboratory, Stanford University, 1990.
- [6] J.H. Lee, A. Prakash, T. Jaeger, G. Wu. Supporting Multi-User, Multi-Applet Workspaces in CBE. In *Proceedings of the ACM 1996 Conference on Computer-Supported Cooperative Work (CSCW'96)*, pages 344-353, 1996.
- [7] Microsoft Corporation. NetMeeting 2.1 Resource, (at <http://www.microsoft.com/netmeeting>), 1996.
- [8] I. Marsic, B. Dorohonceanu. An Application Framework for Synchronous Collaboration using Java Beans. In *Proceedings of the Thirty Second Hawaiian International Conference on System Sciences (HICSS-32)*, 1999.
- [9] I. Marsic. DISCIPLER: A Framework for Multimodal Collaboration in Heterogeneous Environments. To appear in *ACM Computing Surveys*, 1999.
- [10] T. Mansfield, S. Kaplan, G. Fitzpatrick, T. Phelps, M. Fitzpatrick, R. Taylor. Toward Locales: Supporting Collaboration with Orbit. To appear in *Journal of Information and Software Technology*, 1999.
- [11] A. Medl, I. Marsic, M. Andre, C.A. Kulikowski, J.L. Flanagan. Multimodal Man-Machine Interface for Mission Planning. In *Proceedings of the AAAI Spring Symposium on Intelligent Environments*, pages 41-47, 1998.
- [12] M. Roseman and S. Greenberg. Building Real-Time Groupware with GroupKit, a Groupware Toolkit. *ACM Transactions on Computer-Human Interaction*, 3(1):66-106, March 1996.
- [13] M. Roseman, S. Greenberg. TeamRooms: Network Places for Collaboration. In *Proceedings of the ACM 1996 Conference on Computer-Supported Cooperative Work (CSCW'96)*, pages 325-333, 1996.
- [14] M. Stefik, D. G. Bobrow, S. Lanning, D. Tatar. WYSIWIS Revised: Early Experiences with Multiuser Interfaces. In *ACM Transactions on Information Systems*, 5(2):147-167, 1987.
- [15] Sun Microsystems, Inc. JavaBeans 1.0 API Specification (at <http://www.javasoft.com/beans>), 1996.