Colt: A System for Developing Software that Supports Synchronous Collaborative Activities

Lauren J. Bricker, Marla J. Baker, Emi Fujioka, Steven L. Tanimoto
Department of Computer Science and Engineering
University of Washington, Box 352350
Seattle, Washington 98195-2350 USA
{bricker, marla, emifuji, tanimoto}@cs.washington.edu

ABSTRACT
This paper presents Colt, a system designed to aid in the development of applications that support close collaboration between two or more users. The Colt system is comprised of three parts: a design methodology, a software toolkit, and visualization and analysis tools. The software toolkit includes computational objects that support simultaneous manipulations from multiple users. This paper defines and gives examples of these "Cooperatively Controlled Objects" (CCOs) and discusses design and interface issues for developing them. Finally, four example collaborative activities that were built using Colt and CCOs are described: a drawing program, a jigsaw puzzle, a coordination game and a color matching activity.

Keywords
Computer supported collaboration, multiple-user interface, synchronous, cooperatively-controlled objects

INTRODUCTION
Developing applications to support collaboration by two or more users involves special challenges. An application designer needs to anticipate the interaction between the user and the computer as well as the interactions between the users. Additionally, there is a lack of system support for development of software that accepts simultaneous input from multiple users on the same machine. The Colt system is designed to address both of these issues.

Colt is comprised of three parts, a design methodology, a software toolkit, and visualization and analysis tools. The design methodology is based on a series of questions to aid developers in the creative process of planning a collaborative activity. The toolkit supports the implementation of collaborative programs by providing a shell application, a hierarchy of "cooperatively controlled objects" (CCOs), and support for input from multiple users. CCOs are designed to support close collaboration among the users of a computer program. Each CCO contains a mechanism to record a history of how users cooperatively controlled the objects. The visualization and analysis tools present different views of the object histories. These tools can be employed after program testing to determine if the implementation meets the original design criteria. Colt is unique in its support for the development of co-present simultaneous computer based activities.

This paper describes the software toolkit and analysis tools in the Colt system. We begin by motivating the support for simultaneous cooperative interactions. Then we define cooperatively controlled objects with a discussion of some issues encountered in developing them. The third section describes the toolkit, including the objects and the framework in which cooperative applications may be developed, and the analysis tools. Finally, we describe four example activities that include CCOs and were implemented using the Colt system.

MOTIVATION

Examples of non-computer synchronous cooperation
Many common non-computer activities contain elements of highly synchronized cooperation, such as a three-legged race, a music recording session or moving large objects. In the case of a three-legged race, each runner is actually hindered by having one leg bound to a leg of the partner. They must work synchronously and collaboratively to make it to the finish line. The pair that collaborates most effectively (and runs the fastest) wins the race. The purpose of this type of race is not to permit peole to run faster than they would as individuals, but to foster entertaining human interactions.

Another example of an entertaining collaborative task is the recording of a musical symphony. This activity is one, in principle, that an individual person might be able to do (using modern technology); it would involve playing each instrument and recording each part of the piece separately, and then mixing the tracks to form a recording of the whole work. However, the complex and subtle changes of intonation that should result from the real-time interactions among the musicians would be missing from the final record. Performance of the piece in a cooperative setting permits the attainment of a richness that would otherwise be lacking.

The collaborations in the previous two examples are motivated either by their entertainment value or by expected benefits of interations within the group. In the physical manipulation of large objects, however, cooperative control is essential. When two or more people move a piano or a sofa, their highly synchronous

collaboration is facilitated by voice and gesture communication. By contrast, a less intensively collaborative method for accomplishing a task is based on coarse-grained sharing, as in, for example the cooperative solution to the problem of moving a dinette set: "You take out the table, and I'll take the chairs." This affords distributing the work, but it does not require the careful coordination required when a group of people must ease a sofa around a tight bend in a narrow staircase without damaging the sofa or walls.

Reasons to support highly synchronous cooperation

There are two reasons why a designer might want to support highly synchronous collaboration in a computer-based learning activity. First, there is evidence of increased problem solving and enjoyment when users work closely together on computer-based tasks. For example, Inkpen [10] found pairs of students were able to solve more puzzles when working together on a single computer as compared to solving them alone or on separate computers side-by-side. Although the students enjoyed working together, they were hampered by their contention for a single input resource (the mouse). Secondly, the preliminary testing of four MultiIn-based, multi-player activities [4] provided anecdotal evidence that the students tend to communicate effectively about the problems posed while using highly synchronous co-present applications. Unlike the system used in Inkpen's study, the MultiIn activities allowed each user to control a separate input device to eliminate the contention for input resources. These activities also contain objects that allow close synchronous interactions between users working closely on a task

Increased problem solving, communication and a user's level of enjoyment can also be important in a work context. Although the studies described above were done with children and educational software, similar results might be found with adults and in work situations. One goal of this research is to explore whether or not this is true. This paper describes the computer support provided for explorations such as these.

BACKGROUND

Many of the applications written to support collaborative activities focus on the technological challenges, including handling input from multiple users, and transmitting information between the users. We feel that communication between the participants is a vital component of collaborative interactions that should not be overlooked. Furthermore, we feel it is important to examine how different technologies can effect communication during a collaborative activity.

Communication during collaboration

Communication and trust are key parts of a collaborative interaction. A person broadcasts a thought by essentially passing a "message" through one or more of three different channels: verbal (auditory), physical (gestural), or graphical (written, symbolic). Another person receives this message primarily through auditory or visual senses. The more channels of information that is past between people, the less likely it will be for them to have a miscommunication. Conversely, the lower the bandwidth of the communication channels, the harder it may be for the users to communicate their thoughts.

Technological support for communication during collaboration

There are many technologies that support communication, including telephones, electronic mail or newsgroups, electronic talk or chat, audioconferencing and videoconferencing. How useful these systems are in supporting a collaborative interaction depends heavily on the type of task that the users are working.

People generally cooperate for one of two reasons: either they are encouraged or required to cooperate by the nature of the task or application, or they simply enjoy working together. However, not every cooperative interaction requires that participants work at the same time or communicate often. Some collaborative tasks, such as designing and implementing a computer program, require occasional face-to-face meetings enriched with infrequent text based communication. Other collaborative tasks that demand more immediate feedback require that the users communicate in a more real-time manner.

Real-time or synchronous communication can occur using text-based chat, telephone, or audio or video conferencing software. Typing (through email or text-chat) has the drawback that it can take more time than auditory communication. Audio only communication is quicker and more natural to most people, but lacks the ability to gesture. Videoconferencing allows participants to see each other's expressions and gestures, but the low degree of apparent presence and the lack of actual contact can hamper the cooperative interaction. Admittedly, as technology improves and becomes less costly, the dichotomy between face-to-face and distance communication will become less clear. Until that time, nothing will completely replace face-to-face collaboration for transmitting non-verbal communication. When users work in a co-present situation, they can interact directly; they can see each other's expressions and gestures, and they may be able to communicate more effectively than they would without co-presence.

As an interesting note, researchers have found that groups who conduct a brainstorming session using computer-mediated communication are more prolific than groups that meet face-to-face [15]. Still, present day technology-based collaboration cannot completely replace some facets of face-to-face interactions, such as building trust. For instance, Rocco found that groups that meet face-to-face prior to doing a virtual task collaborated better than those who did not meet before [15]. The face-to-face

interactions helped them develop trust and an understanding of other participants' communication styles that then carried over into the technology-based interactions. Thus, computer-mediated communication should be used for those times when the participants are unable to get together physically, and not as a complete replacement for co-present collaborations.

In addition to the problem of low apparent presence, current communication technologies do not always guarantee that all of the users involved in the task have the same view of visual materials. Having the ability to view the same task materials may be important in reducing the amount of miscommunication that occurs between the participants. For example, if the users are communicating using a telephone or text chat and attempt to find the same information on the WWW, there is no way to guarantee they are on the same web page. Some systems, such as Microsoft NetMeeting [11], do allow a group to share a communal visual artifact, such as a whiteboard or web browser. But again, there is no guarantee that all of the users involved are actively focused on the shared information. In contrast, participants of a co-present collaborative effort would notice if a group member lost focus on the task when he or she looked or walked away. This natural sense awareness may increase productivity or aid in the group's ability complete a task.

Thus there are tasks or instances where there is currently no replacement for the ease of communication and natural sense of awareness in co-present collaboration. Unfortunately, most desktop computers today are designed to accommodate input from one user at a time. Groups who wish to collaborate using one computer are forced to share control awkwardly through a single mouse and keyboard. Social and physical conflicts over these input devices, such as shown in Figure 1, may arise since current operating systems and application software only support a single input stream. One solution to the conflict problem is to provide a separate physical device for each user. Many video game systems, such as the Super Nintendo Entertainment System (SNES), do support multiple joysticks. However, the multi-user games for these systems often employ either a turn-taking mechanism, or simultaneous competitive play. The risk in enforcing turn-taking is that inactive users will lose focus on the task, as shown in Figure 2. The risk in supporting simultaneous competetive play is that it does not appeal to many people, particularly young girls. Colt was developed to support simultaneous collaborative interactions in order to address these issues.

Figure 1. Conflict over a single mouse often occurs when two children attempt to use a single desktop PC.

Figure 2. Even with multiple input devices, one user may lose focus on the task if he does not have some control on the screen.

Domains for collaboration

There are two primary domains that have been studied in context of computer support for collaboration. *Computer Supported Cooperative Work* (CSCW) and *Computer Supported Collaborative Learning* (CSCL). CSCW is the study of software and systems designed to support activities that are coordinated between two or more people. CSCW is generally focused on how to make a group work more efficiently or productively together.

Similarly, CSCL is the study of systems and software

designed to support collaborative learning. Collaborative learning is designed to encourage communication of concepts, discussion of solutions, resolution of social and cognitive conflicts, and promote problem solving and higher-order thinking skills. In order to work successfully on a collaborative task, students must be able to communicate their thoughts. This requires that they understand and clearly formulate ideas in their own minds before they can describe them to others in the group. The goal of a collaboratively learning task may not be to make the users work more productively together, but rather to increase the amount and quality of communication between students. Collaboration also may add an enjoyable aspect to a task.

Collaboration is also an important issue in entertainment. Some tasks may be more difficult for the users in a forced collaborative environment than they would be for a single person, and yet this may be desired. Just as in running a race, where an individual runner could easily beat a pair of runners who each have one leg bound to one of the other's (as in a three-legged race), a computer-based collaborative activity may have social, team-building, and entertainment value of its own. Many people, especially children, enjoy themselves more when they can play computer games together. Unfortunately, many of the multi-user games on the market today are fundamentally competitive in nature. For example, the goal of first-person perspective worlds, such as Doom or Quake, is to traverse the levels of the world, collecting ammunition, food and medical supplies, while killing the opponents that get in the way. These games allow users to play together across a network. The users compete against one another, or form cooperative teams that fight against other teams of cooperating players. We use the phrase *Computer Supported Collaborative Entertainment* (CSCE) to describe software and systems designed to support collaborative interactions while not specifically encouraging productive work or learning.

This research explores the design and development of collaborative activities in all three domains, CSCW, CSCL and CSCE. Rather than concentrating on a specific domain of collaboration, our focus is on cooperation where the users are simultaneously working together. Although simultaneous interactions can occur in either a face-to-face or distance situation, we are primarily interested in the examining co-present collaborations. This type of interactions may increase the amount of communication between the users, a feature that may be best suited for learning and entertainment activities, but can be useful in a work context.

THE COOPERATIVELY CONTROLLED OBJECT

Many common non-computer activities contain elements of highly synchronized cooperation. Some activities are done collaboratively purely for the social enjoyment. Some are done because they are easier to complete with additional help. One of the goals of this research is to define a class of interesting objects that helps to keep the users focused during collaborative activities. One can expect that there will be an increase in total or certain types of communication between the users if the users become more focused on a task.

Cooperatively Controlled Objects Defined

In object-oriented terminology, an object contains state and has behavior. A *controlled object* is an object containing methods that allow one or more users to manipulate properties of that object. Typically these properties are manipulated directly through input devices such as mice, joysticks, keyboards, etc. A *cooperatively controlled object* (CCO) is a controlled object that is designed to be manipulated simultaneously by more than one user based on certain relationships that hold between user inputs and components of the CCO.

Each CCO is assumed to have a set of components or "properties." Each property may take on one of a possibly infinite number of values. In the physical world some properties, like the location of a spot on the floor, may have an infinite number of possible values. We will assume that the computer representation is simplified with a finite set of discrete values.

A control mechanism for a CCO provides a means for a group of users to provide a value for one or for several (or even all) of the properties of the CCO. The control mechanism for a property in a CCO can be described as a function:

$$V = f((v_1, v_2, \ldots, v_n), (u_1, u_2, \ldots, u_m)),$$

where V is the output, or new value of the property, each v_i is the input value of the properties on which this function depends, and each u_j is the input-device states corresponding to the multiple users simultaneously manipulating the object. By providing functions for each of the CCO's modifiable properties, a complete control mechanism for the CCO can be specified.

Fine-grained sharing (FGS) is a previously studied approach for defining one class of CCO [16]. In FGS, an object is expressed in terms of its properties, each of which is controlled by one of the users. Although fine-grained sharing can offer users a clear delineation of responsibility and control for shared objects, it lacks the ability to compel tight cooperation among the users. Users who work on separate pieces of a "shared" document do not necessarily coordinate in real time. Instead the work is done asynchronously and are the users not forced to synchronize their activities to the same degree that one does in musical performance, dance, three-legged races, and the like.

Our goal is to compel users to interact synchronously. CCOs support close collaboration though complex interactions with objects by adopting a more general framework than fine-grained sharing.

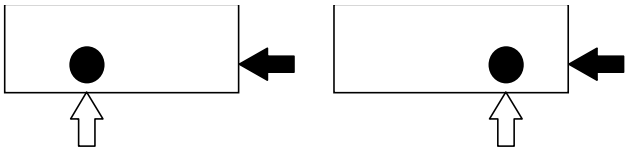

Figure 3. Controlling the location of a point using a fine-grained sharing technique.

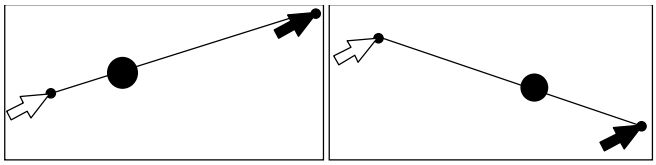

Figure 4. Controlling the location of a point using a cooperative control technique.

Figure 3 depicts an example of a FGS technique in which a point is controlled by decomposing it into its x and y coordinates and each coordinate is modified by separate users. This could be described more formally as

$point.x = u_1.x$
$point.y = u_2.y$

A type of cooperatively controlled point, shown in Figure 4, could involve a more general functional relationship between the inputs from the two users and the controlled properties. In this example the controlled point is defined as the midpoint a line and the two users control the point by moving the endpoints of the line. More formally this could be described as a series of functions, such as:

$point.x = (line.endpoint_1.x + line.endpoint_2.x) / 2$
$point.x = (line.endpoint_1.y + line.endpoint_2.y) / 2$
$line.endpoint_1.x = u_1.x$
$line.endpoint_1.y = u_1.y$
$line.endpoint_1.x = u_1.x$
$line.endpoint_2.y = u_2.y$

There are many ways geometrical objects, such as lines and polygons, can be controlled using fine-grained sharing at the level of coordinate values, vertices, etc. They can also be controlled cooperatively via other geometric relationships between input values and the controlled properties. More complicated spatial objects, such as fractals, can be cooperatively controlled by allowing users to jointly manipulate their parameters or properties. Yet another example of cooperative control can be found in the Curve Fitter program, described in [4], in which two to four users manipulate the shape of a degree-n polynomial curve by moving n+1 control points located on the curve. For each geometrical or abstract object there usually are a number of different plausible methods for cooperative control.

Specifying the types of interactions

Developers may also wish to control how closely the users in a collaborative activity work together. In order to assist in the specification of the users' interactions, we have developed a sub-classification of synchronous activity based on how simultaneously the users interact with a screen object.

An *asynchronous cooperative interaction* is one in which two or more users work independently on a task, then synchronize or coordinate their work by delivering information through messages such as conversation, snail mail or email. Participants of this type of cooperative interaction may not expect an immediate response from questions or requests. A *required asynchronous cooperative interaction* is one in which each user must complete their part of the activity before the other user may proceed. An example of this type of interaction in the real world is two people trying to go through a doorway that can only fit one person at a time.

A *synchronous cooperative interaction*, on the other hand, is one in which two or more users interact on the same task in real-time. Phone calls or text chat are examples of synchronous cooperative interactions. Often, the users in a text chat session take turns in writing (the reader will wait for the writer to complete a thought before responding), but they may also type at the same time. In this case a chat writer is not guaranteed that the other user(s) are focused on the current thread of conversation, but they are working *simultaneously*.

We have further broken up this temporal classification based on how simultaneously the users interact with each other. A *required* simultaneous interaction is one in which the users must manipulate an object at the same time in order to make any progress on the activity. An example of this is two people attempting to lift a heavy piece of furniture, such as a piano. An *encouraged* simultaneous interaction is one in which the users do not have to manipulate objects at the same time, but doing so will allow them to complete the task more easily or in a shorter amount of time. Furthermore, their progress may be hampered (slower) if the interactions are not simultaneous. Moving a heavy object where no lifting is required is an example of this type of interaction.

We can further describe asynchronous interactions as two other forms of "simultaneous" interactions. As with an asynchronous interaction, a *discouraged* simultaneous interaction is one in which the users can make progress on their task independently. However, it also may be much more difficult for the users to make progress on the task if they work on it at the same time. Two people on a playground seesaw are an example of this. If both people on the seesaw attempt to push up at the same time, neither of them will be able to push the other very far. Finally, we can also describe a *required asynchronous* interaction as a *disallowed* simultaneous interaction. Here, the software or system physically prevents the users from making progress on their task if they work together. Two people going through a small doorway is a disallowed simultaneous interaction.

Degree of cooperative control

All objects that are jointly manipulated by two or more users are considered cooperatively controlled. However, there is a wide variability in how these objects are cooperatively controlled. Some objects are controlled through complex constraints that require simultaneous interaction to be activated, but others are simply controlled without constraints through asynchronous interactions (as in a shared database). As a way to specify the differences between the wide variety of cooperatively controlled objects, we introduce a measure called the *degree of cooperative control (DOCC)*.

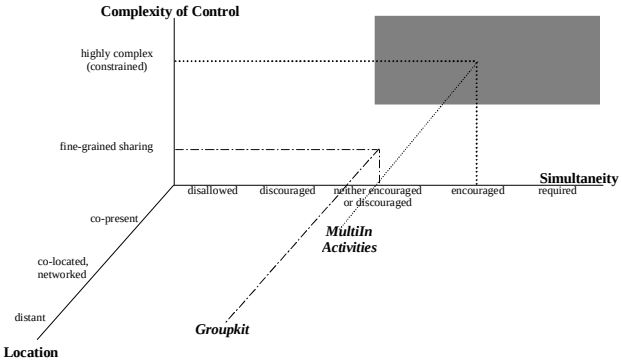

Figure 5. A visualization of three characteristics of systems that use cooperatively controlled objects

The degree to which an object is cooperatively controlled depends on two factors, the simultaneity of interaction and how complex the method is for manipulating the objects. If we graph these two factors on orthogonal axis, such as in Figure 5, one might think of the more cooperatively controlled objects as being in the upper right corner of the graph. This is similar to graphs seen in the CSCW and CSCL literature, such as the one in Ellis, et al [7, p. 41], where groupware is categorized based on the location of the users (same or difference places) and the times or method of synchronization (same or different times). The physical location of the users can be added to this graph as third axis, and although distance does play a role in how tightly synchronized the interaction may be, this will not factor into our definition of a CCO. CCOs are, by definition, able to be used by users who are at a distance as well as co-located.

User Interface Issues for CCOs

A number of interface issues must be taken into consideration when writing software to support two or more people working in a synchronous collaborative situation. If users are to manipulate a single object simultaneously, each user must have access to her or his own device to input their modifications. It is possible, if the users are co-located and using a single display, to use an off-the-shelf computer system with one mouse and one keyboard. However, this scenario typically causes frustration because of either an unintuitive interface or a contention for the mouse or keyboard. Obvious solutions include having the users interact over a network, or to support multiple input devices on a single machine, similar to systems such as MultiIn [4] and MMM [3].

Apart from the hardware, the software should indicate objects each user may access. Also, some indication should be given to all users as to which user currently has possession or is controlling each object in the shared space. Color usually affords a reasonable sign of object ownership, particularly if the color matches that of the owner's cursor.

Co-present vs. Distant Collaboration Considerations

A popular way to distinguish different kinds of CSCL situations is along an axis that represents the relative distance between the participants (long-distance, intra-meeting-room, or co-present) [7]. While the definition of a CCO is not dependent on the location of the users, different interfaces may be required for each situation.

As stated previously, conflicts often arise over the use of the physical input devices if there is only a single mouse/keyboard pair for all the participants during a co-present activity. One solution to this problem is to provide a separate physical device for each user. Support for multiple input devices must be both at hardware and a software level. Many operating systems today can support multiple joysticks, and some more recent systems support a new standard known as the Universal Serial Bus (USB) [17]. USB is can provide inexpensive multiple serial input, such as mice, keyboards, joysticks, etc. However, most off the shelf operating systems do not support multiple people using separate input devices to control on-screen artifacts at the same time. In order to support synchronous co-present cooperative control, operating systems should, at a minimum, provide multiple cursors to give each user an on-screen presence. This requires that the system accept multiple streams of events (such as mouse move events) from the separate input devices, and that these events include the ID of the device that generated them. Once a multiple cursors are supported, software can be written to allow the users to cooperatively control objects.

In contrast to the lack of support for co-present collaboration, many systems can and do support distance communication by passing messages between each user's computer. However, the latency of this message passing across the network can be problematic for synchronous collaboration. Take as an example, two users attempting to trace the path as shown in Figure 6a. If the user with the white cursor (W) moves before receiving a message that the user with the grey cursor (G) has moved, W might move as shown in Figure 6b. However, if the G had actually moved first and the message had not arrived to update W's system because of network latency, the result might be as shown in Figure 6c. Network latency such as this could easily interfere in how well the users are able to work together.

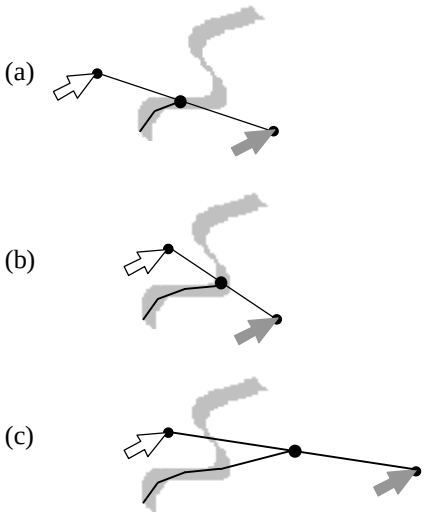

Figure 6. Problems with network latency when two users are trying to tracing a path with the midpoint CCO.

To make up for some of the communication deficiency of long-distance cooperative control, the appearances of shared objects can be adjusted dynamically to reflect not only current ownership or control, but also other emergent properties such as the current degree of contention or intensity of "force," "internal tension," or "pressure" in the object. In the example where a midpoint object is controlled by the endpoints of a line segment, the length of the line segment can be considered as a measure of internal tension, particularly if the line segment is thought of as a rubber band or spring. The color of the half-line segments can be made to indicate the degree of disparity between two users' intent for the controlled point. Additionally, a point of this kind, under high tension, could be displayed in bright red.

THE COLT SOFTWARE TOOLKIT

There are many toolkits designed to support computer-based collaboration. These toolkits facilitate the development of cooperative software by providing abstractions for more difficult concepts in programming groupware applications, such as how to maintain synchronization of the objects or views of objects, and users dynamically joining and leaving sessions. Most of the existing toolkits, such as JAMM [2], Rendevous [9], DistView [14], GroupKit [8], and Habanero [13] support synchronous cooperation over a network.

There are also a number of collaborative systems that encourage communication at a distance through common objects [8], tasks [9] or physical situations [3]. Groupkit [8] and Turbo Turtle [6], in particular, use indicators such as telepointers and other "awareness widgets" to indicate to others (working at distant locations) where a user is working on the screen. These systems permit cooperation, but the activities in which they are used do not require the users to work on a task simultaneously or to communicate frequently. Furthermore, none of these systems support co-present interactions. We believe that there are some situations in which a common focus and closer communication are objectives of the activity.

Since we feel that support for co-present, simultaneous cooperative control of objects may be important in some applications, we have developed the *Collaborative Toolkit*, or *Colt.* Colt is designed to facilitate the rapid prototyping or implementation of such applications. This toolkit includes the Collaborative Object-based Application Program Interface (CO-API) hierarchy of Cooperatively Controlled Objects and the CoImage application shell with support for various multiple-user input solutions. The overall architecture for the Colt system is show in Figure 7.

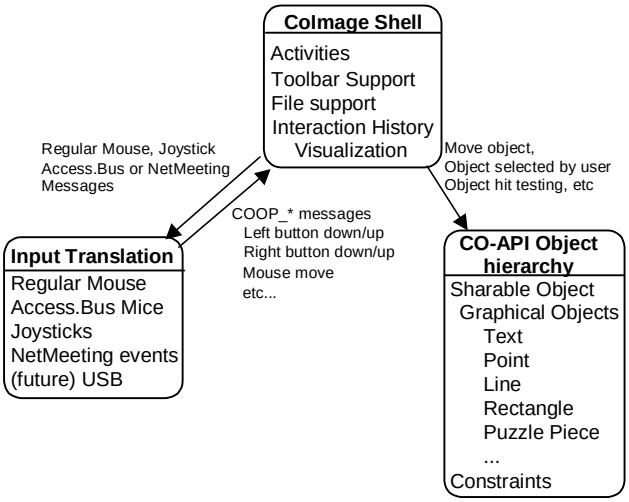

Figure 7. The Colt toolkit system architecture.

As input is delivered to the CoImage shell window, it is diverted to the Input Translation subsystem (ITS). This subsystem translates the input-device dependent event messages into input-device independent Colt event messages. Colt messages include information about position, button state, and the user associated with the message. The activities implemented in the CoImage shell respond to the device independent messages. The movement and location of each device is mapped to a separate color-coded cursor on the screen. Based on the location and the owner of a cursor, the activity software can determine which objects to manipulate and if that user may manipulate that object, respectively. In particular, if a user attempts to manipulate an object he does not own the object will not allow that user to control the object, and the attempt to control the object is ignored. The CCOs in the CO-API do not respond directly to the input events, they only respond to changes to object properties.

The CO-API, CoImage Shell and Input Translation subsystem will be discussed in the next sections.

The system requirements for the toolkit are Pentium based

PC platforms running Microsoft Windows 3.1 or Windows 95. For the co-present, single display situation, we currently support the Access.Bus multiple input system from Computer Access Technology Corporation for Windows 3.1 and Microsoft Sidewinder™ Game Pads for Windows 95. The CoImage Shell supports distance collaboration with Microsoft NetMeeting™ conferencing technology. NetMeeting supports audio, video, file and data transfer, as well as a rudimentary sharing (turn-taking) mechanism for single-user applications that are not designed to be manipulated simultaneously.

The CO-API

The CO-API is comprised of a number parts: a hierarchy of cooperatively controlled objects (including object necessary for specifying access and recording a history of events) event, a constraint system, and tools for viewing and transforming bitmapped images.

Cooperatively controlled objects are implemented in our software environment as reusable C++ classes, and they thereby can inherit state and behavior. CCOs derive the ability for sharing from access control lists similar to [16]. Other properties of a CCO include location, size, orientation, coordinate system, display parameters, and color. An object will ignore attempts to change a property by users who do not have the proper privileges (as dictated by access control lists).

In order to track when the users cooperatively control objects, each object in the CO-API can also store information regarding the users' changes to an object, or *actions*, in what we call a *history of actions*. The information stored for each action will vary depending on how the object is manipulated, but typically includes the changed property and the new value, which user effected the change, when the change occurred and how long it took. The history of actions may be saved and analyzed after the activity is completed. This information may be restored at a later time for evaluation purposes. The interface for an activity can be designed to include visualizations that aid the analysis of measured quantities such as the number of interactions on an object, the total time of each interaction, how the interactions overlapped, or the order in which the users interacted.

In addition to the object hierarchy, the Colt system includes a *Constraint Manager* for use in the implementation of more complex interactions between the users and the objects. The Constraint Manager keeps a list of constraints currently used in the Colt system. Each constraint contains a list of input objects, or objects that may affect other objects in the system, and a list of output objects, or objects which are affected by changes to other objects. As a user manipulates an object in the CO-API hierarchy, the object informs the Constraint Manager that it is being modified. The Constraint Manager checks its list to see if any constraint includes that object as an input. The constraint manager will then "fire off" any changes to output objects of constraints using this object as an input object. The constraint can be viewed as a function taking states of input objects to new states of the output objects.

Currently the constraints in the CO-API hierarchy are implemented as one-way constraints. At the time of the original implementation we did not see the need to include multi-way constraints or to gracefully handle cyclical constraint dependencies in this system, but the constraint manger is designed to handle the eventual use of more complex constraint hierarchies

The CoImage Shell

We support development of collaborative activities using CCOs by providing an application shell that handles the operations, such as handling input from toolbars, and saving and restoring files, which are shared by all activities built in the environment.

As in Bricker, et al [4], the activities supported by Colt allow each user to own an input device and corresponding colored cursor on the screen. The color of the cursor identifies its user. Objects are displayed in a user's color if only they are permitted to manipulate that object. For example, a tool on the toolbar is outlined with the user's color if they are permitted to use that tool. If two users may select a tool, it is outlined with both of the users' colors. The design and implementation of an activity defines how the tools in the toolbar are used. Some tools simply set the value of a property for an object, while others may change a mode for the whole activity. Depending on the design of the activity, a user's cursor may change shape to indicate that a user just selected an instance of a particular tool.

The CoImage shell also supports saving and restoring data in a number of formats. The shell includes methods that save the state of an activity as a binary data file so that a user can continue at a later date. The shell also includes separate methods for saving a history file in a tab delimited text format. This format permits an experimenter to visually read the data or analyze it using another program such as a spreadsheet. A copy of the data is also written in a binary CoImage data file format whenever the text version is written to disk. The CoImage shell can also save image data in the standard Microsoft Windows bitmap format, and in a format that can be read by a World Wide Web (WWW) browser.

The CoImage shell also includes methods that read and write a text-based problem description file. A problem description includes an initial state, goal states or criterion the users are expected to reach, any constraints enforced on the users in reaching that goal, and a possible scoring criteria. The current implementation of the shell contains methods that can be overridden by an activity developer to read and write problem description files. Most activities implemented with Colt to date support a problem file that contains the number of users, the initial state of the

program, a text description of the goal state(s) that can be presented to the user, and the scoring criterion.

The Input Translation Subsystem

The input translation subsystem (ITS) was developed to abstract away the details of any specific input device. Device dependent input events delivered to the CoImage shell are diverted to the ITS, where they are translated into input-device independent messages. These Colt messages include information about the mouse position, the state of the mouse buttons, and the user associated with the message. Thus, the application developer only needs to respond to the one set of messages

ANALYSIS TOOLS

The Colt system also provides tools to aid in the analysis of the history of actions stored by each CO-API object used in the activity. We will discuss two tools here; the visualization and the measure of joint activity.

Visualization of the users' interactions

The visualization of the history of actions provided in the Colt system is intended to aid in analyzing how users work together on an activity. This visualization shows the time interval each action occurred as a bar color-coded to match which user changed the object. The bars are marked with a character to denote what type of action occurred (*s* when the object was selected, *m* when it was moved, *r* when it was rotated, etc). The colors of the bars denote which users produced the events.

Figure 8 shows an example of a visualization from the "drawing" activity described in the next section. In this figure, the drawing point (ptTool) is cooperatively controlled by two methods. In the first thirteen seconds (denoted by the tick marks on the time line at the top), each user is moving the point by changing the x- or y-coordinate with a scroll bar. At first these movements are sequential, then they occur simultaneously, shown when the movement intervals overlap. In the second half of this figure, the two users are changing the location of the point by moving a line together.

We note that this visualization is easily understood if the context of the interface and how the objects relate to each other are known. We feel it is valuable in assessing how users work together, when they communicate and what their communication may entail.

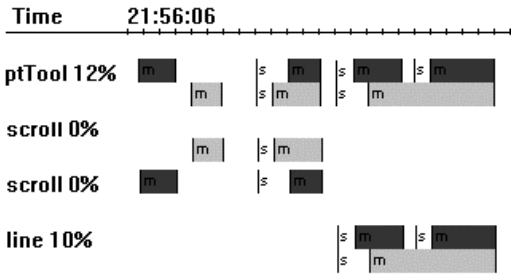

Figure 8. A visualization of the *history of actions* in the drawing activity. A bar with a *s* indicates the user selected an object, a bar with a *m* shows the user moved that object.

Measures of the users' interactions

One reason to support highly synchronous collaboration is that it may help users stay focused on a task. During the course of an activity, however, some users in a group may contribute a lot more to the solution of the problem than others. The expectation is that the more actively a user is participating, the more likely he or she will be focused on the given task. Such a measure of a user's involvement might be helpful to a designer during an iterative development process to identify when the users were able to cooperate and when they had problems. This measure could also be used as feedback to a teacher who needs to determine the contributions of each member of a collaborative group to the solution of a problem.

One measure of the users' involvement is to determine how much the users either independently or jointly manipulate an object. For any object in an application, the *measure of joint activity* or *JA* summarizes this information by weighting the relative amount of time the users work in subgroups of various sizes. Thus, the time the users spend working simultaneously all together weighs more heavily than the time spent working in subgroups, and the time the users spend working in subgroups weighs more heavily than the time the users spend working individually. The JA can be thought of as a measure of the average number of users who are actively manipulating an object at any given time. The Colt system includes an analysis tool for calculating the JA that may be used while viewing the event spans for an activity. The tool has a very simple interface that takes an input file containing the periods of time that are to be analyzed, an output file name and the object to analyze in the activity. The tool produces an output file containing the analysis in a tab-delimited text form.

ACTIVITIES THAT EMPLOY COOPERATIVE CONTROL

We have developed a number of activities that employ CCOs using the Colt system. As in Bricker et al [4], the activities described below allow each user to own an input device and corresponding colored cursor on the screen. The color of the cursor identifies its user. Objects are displayed in a user's color if only they are permitted to manipulate that object.

A Collaborative "Drawing" Activity

The collaborative Etch-a-Sketch™ is one of the activities implemented using Colt, based on the toy developed by Ohio Arts. This activity allows the users to draw with one of three different cooperatively controlled pens. In the simplest form of the activity, users can manipulate the pens to draw what they'd like, or they can be given a specific task such as "draw a house." The activity can also be used to solve more complex problems, such as mazes,

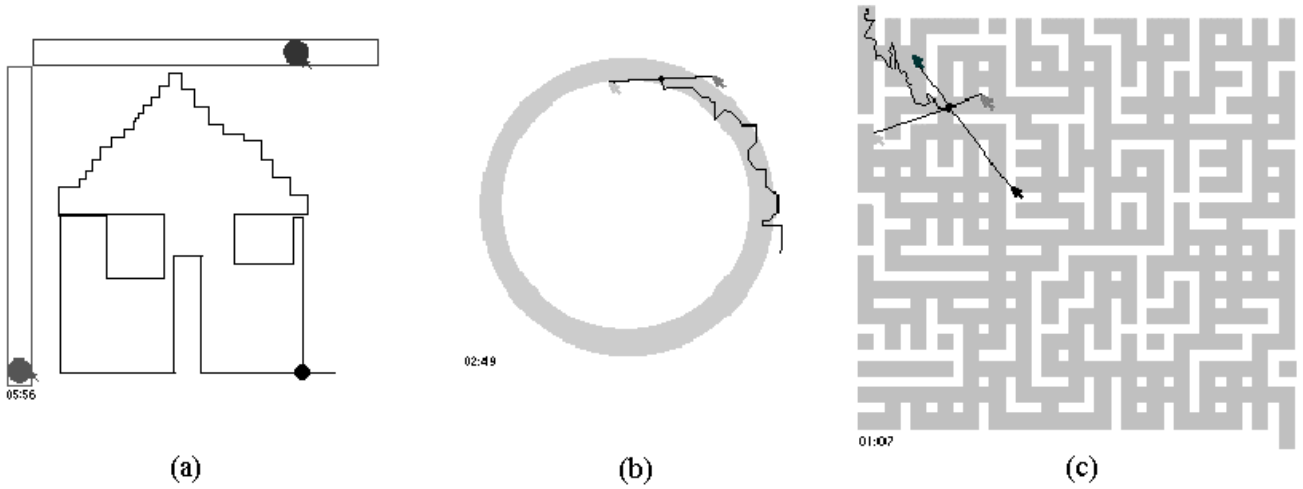

Figure 9. (a) Fine-grained sharing etch-a-sketch used to draw a house, (b) Two users manipulating a cooperatively controlled pen, (c) Four users manipulating a cooperatively controlled pen to solve a maze.

by including a background picture that the users must cooperatively trace in the least amount of time. Once the timer begins, the users must stay within the boundaries of a background object or time is added to their total in increments larger than 1 second.

The simplest cooperatively controlled pen allows each user to manipulate a slider. The location of the "thumb" in the slider corresponds to the *x* or *y* coordinate of the pen. An example of drawing a house using the "fine-grained shared" pen is shown in Figure 5a.

Figure 5b shows users tracing a circle with another CCO. This control constrains the location of the drawing pen to be the midpoint of a line segment. Two users adjust the location and rotation of the line by manipulating endpoints of the segment. To make this interaction more difficult, each user may only manipulate the endpoint he "owns," which is colored to match his cursor. A user may not manipulate the other user's endpoint. This cooperatively controlled pen could be further restricted so that the line cannot move until both users are actively manipulating their endpoints.

The most difficult control, shown being used to solve a maze in Figure 5c, requires four users to manipulate the pen. The location of the drawing nib is defined as the intersection of two lines. Each line is controlled by its own pair of points, and each user may only manipulate the point colored to match her or his cursor. Even without this restriction, this pen is very difficult to control without help of other users. By making it difficult to manipulate the pen alone, this control encourages the users to focus together on the task and communicate about the problem posed by the activity.

The Collaborative Puzzle Activity

The collaborative puzzle is an activity implemented by a student using Colt [1]. The users may choose to solve the puzzle in parallel, or by using the CCO version of this activity. In the *parallel* version, shown in Figure 10 the pieces of the puzzle are moved separately, although the puzzle as a whole can be worked on simultaneously by more than one user. This is identical to the interaction of multiple people working on a physical jigsaw puzzle.

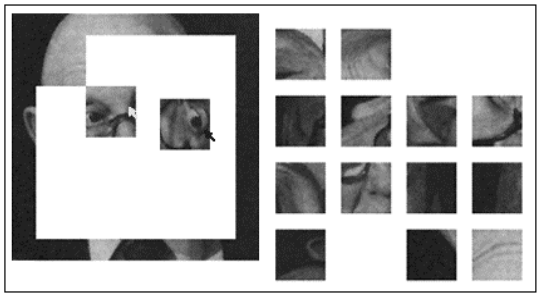

Figure 10. Two users manipulating pieces simultaneously in the Parallel Puzzle Activity.

In the *CCO* version of the activity, shown in Figure 11, the positioning of each puzzle piece is controlled using a line segment. The geometric center of the puzzle piece is constrained to the midpoint of an "attached" line segment. Two users adjust the location and rotation of each piece in an integrated manner by manipulating endpoints of the segment.

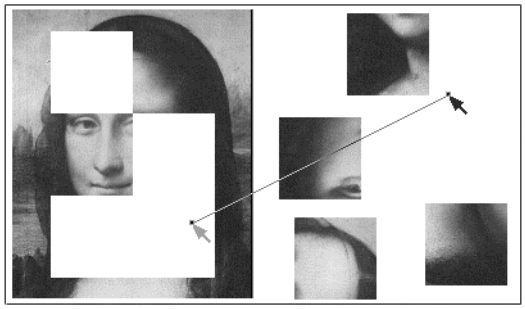

Figure 11. Two users manipulating one piece using a line segment in the Constraint Puzzle Activity.

Both the parallel and CCO versions of the jigsaw puzzle

activity have another interesting feature: they utilize the history of actions for each object to maintain a record of the selections, rotations and translations performed by each user. This information can be saved and analyzed by researchers after the puzzle has been solved. The interface includes added visualizations to help analyze measured quantities such as the number of actions on a puzzle piece (by each user or by all users), the time of each action, and the order in which the users interacted with each piece. These added visualizations are described in more detail in [1].

An Exercise in Coordination: The Chopstick Activity

The Chopstick activity was another activity developed by a student testing the Colt System. This activity is based on a game where two people each use a chopstick to pick up a "bean." The users must coordinate tightly in order to be successful at the game.

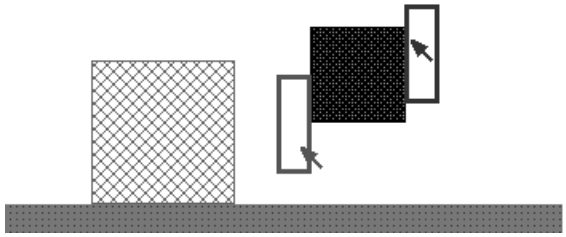

Figure 12. Two users manipulating the large square "bean" in the Chopstick activity.

There are a number of options in this activity, some of which are designed to make the coordination much more difficult. There are two sizes of bean to pick up, large and small. The bean can be square, which has more surface area on the sides to work with, or round. In more difficult rounds the basket "floats" above the ground, much like a basketball basket. In the most difficult case, the basket slowly moves across the screen. Figure 12 shows the simplest version of this activity in action.

The Color Matcher: Colt style

The Color Matcher activity, as described in [4], was originally implemented in Microsoft's Visual Basic with the MultiIn system to support the Access.Bus multiple input devices. MultiIn can only be used under Windows 3.1, and we wished to conduct a user study on a version of this activity under Windows 95 (for more information on the user study, see [5]). The original version of the Color Matcher activity took an undergraduate student approximately ten weeks to implement. The re-implementation of the activity using Colt took the author only a few days. This version included an interface that mirrored the original version as well as a new CCO method for selecting the users' color, shown in Figure 13.

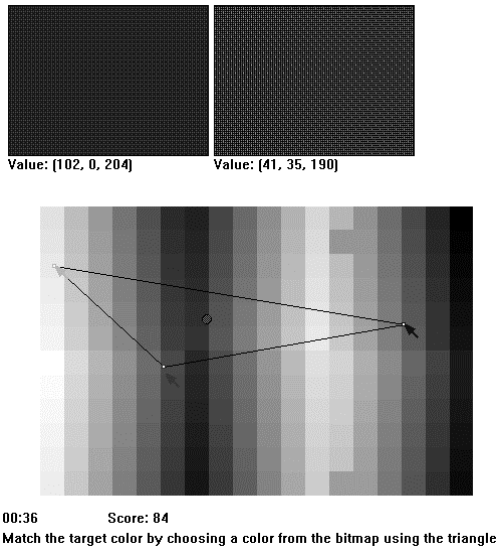

Figure 13. The Color Matcher activity user interface.

In this activity, the users' color is set according to the color of a selected pixel in a bitmapped image (e.g. a palette). The pixel location is set by the location of the centroid of a triangle. Each user controls the vertex of the triangle that is colored to correspond to the color of their cursor. Although it would be impractical to have a bitmapped image of a full 24 bit color palette, the Windows 256 color palette affords a reasonable approximation for the purposes of the activity. The users' color in the Colt/CCO version of the activity is scored in the same way as the original application – based on the distance from the target color in RGB color space.

TESTING THE TOOLKIT WITH DEVELOPERS

The graduate student who authored the Puzzle activity (Marla Baker) was the first user of the toolkit. She felt that the toolkit made her job a lot easier by hiding many of the details that are necessary in a multiple user system. These details included displaying multiple cursors, mapping users to the cursors, and translating device dependent input messages into a device independent form that contains user information. Marla noted that her primary difficulties included learning the toolkit, Microsoft Visual C++ and the Microsoft Foundation class hierarchy. Her only negative comment about using the toolkit was that it was constantly evolving at the time she was developing her activity, and that the documentation was somewhat scarce and out-of-date [1]. The latter problem was later rectified.

The Chopstick game was designed and developed by an undergraduate at the University of Washington (Emi Fujioka) as a way to test the design methodology as well as the toolkit. The design and development of Emi's Chopstick activity was begun after Colt had become more stable than for Marla's project and a user manual had been written. Emi found both the documentation and the example code to be useful, but did make some suggestions

on how to improve them. These suggestions will be incorporated into the next version of the documentation. Emi also was able to develop her activity relatively easily, and also had to learn the compiler environment as well as the Colt system. Part of her design would benefit from the implementation of multi-way constraints, but since the system is not yet supporting them, she used other methods to simulate this. She did notice that it is currently impossible to get the information for both of the sticks simultaneously and that the device events arrive at the application serially. This was another area in which she had to work around the existing toolkit to permit users to move the ball with the sticks.

FUTURE WORK

The Colt system was tested in the development of the drawing, jigsaw, and chopstick activities. Further testing needs to be done with developers to improve the system.

There are many enhancements we'd like to add to the software toolkit, including adding support for other types of multiple input devices, such as the USB, and support for multi-way constraints. As the USB standard gains acceptance and is supported, we would like to modify the Input Translation subsystem of the CoImage Shell to use it. The CCOs designed to date can also be ported to other platforms, such as Java, support distance collaboration in WWW based activities.

We also wish to continue to expand the CCOs in CO-API to support other types of collaborative interactions and activities. Colt could be used to implement collaborative versions of popular games such as: Tetris™, where one person controls side-to-side motion while the other controls rotation. Additionally, CCOs could be used in 3D virtual worlds, or first-person perspective games such as Doom™. An example of a CCO in such a game is a cooperatively controlled latch on a door where users may be required, encouraged, discouraged or forbidden from manipulating the latch at the same time.

CONCLUSIONS

This paper has given a description of the toolkit and analysis tools in the Colt System, as well as collaborative activities that were built using them. The software toolkit includes Cooperatively Controlled Objects, objects generalize fine-grained shared objects in being manipulated by more complicated, higher degree-of-freedom, or less intuitive methods. Additionally, CCOs enforce a tighter degree of simultaneity than their fine-grained shared counterparts. CCOs and the applications that use them can help encourage close collaboration and communication among a group of users.

ACKNOWLEDGMENTS

We gratefully acknowledge the partial support of the Washington Technology Center, Ark Interface II, a Packard Bell Company, and the National Science Foundation under Grant Number RED-9155709.

REFERENCES

1. Baker, M.J. Bricker, L.J., Tanimoto, S.L. *Cooperative interaction techniques in a computer-supported collaborative learning environment*. University of Washington Technical Report UW-CSE-97-04-03. April, 1997.

2. Begole, J., Struble, C., Shaffer, C., and Smith, R. Transparent Sharing of Java Applets: A Replicated Approach, in *Proceedings of UIST'97*, (Banff, Canada, October 14-17, 1997), ACM Press, N.Y., pp 55- 64.

3. Bier, E. and Freeman, S. MMM: A User Interface Architecture for Shared Editors on a Single Screen, in *Proceedings of UIST'91*, (Hilton Head, November 11-13, 1991), ACM Press, N.Y., pp 79-86.

4. Bricker, L., Tanimoto, S., Rothenberg, A., Hutama, D., Wong, T. Multiplayer Activities Which Develop Mathematical Coordination, in *Proceedings of CSCL'95* (Bloomington, October 17-20, 1995), ACM Press, N.Y., pp 32-39.

5. Bricker, L. *Cooperatively Controlled Objects in Support of Collaboration*. Ph.D. Thesis, University of Washington, Department of Computer Science and Engineering, Seattle, 1998.

6. Cockburn, A. and Greenberg, S. Children's Collaboration Styles in a Newtonian MicroWorld, in *Proceedings of CHI'96* (April 13-18, Vancouver, BC), ACM/SIGCHI, N.Y., 1996, pp. 181-182.

7. Ellis. C. A., Gibbs, S. J., and Rein, G. L. Groupware: Some issues and experiences. *Communications of the ACM*, 34(1):38-58, 1991.

8. Gutwin, C, Stark, G., and Greenberg, S. Support for Workspace Awareness in Educational Groupware, in *Proceedings of CSCL'95* (Bloomington, October 17-20, 1995), ACM Press, N.Y., pp 147-156.

9. Hill, R., Brinck, T., Patterson, J. Rohall, S., and Wilner, W. The Rendezvous language and architecture. *Communications of the ACM*, 36(1):62-67, 1993.

10. Inkpen, K., Booth, K.S., Klawe, M., and Upitis, R. Playing together beats playing apart, especially for girls, in *Proceedings of CSCL '95* (Bloomington, October 17-20, 1995), pp. 177-181.

11. Microsoft. Microsoft NetMeeting. Available as http://www.microsoft.com/netmeeting/ (Accessed September 3, 1998).

12. Munson, J. and Dewan, P. A concurrency control framework for collaborative systems, in *Proceedings of CSCW '96* (Boston, November 16-20, 1996), pp. 278-287.

13. National Center for Supercomputing Applications. NCSA Habanero. Available as http://www.ncsa.uiuc.edu/SDG/Software/Habanero/ (Accessed September 2, 1998).

14. Prakash, A. and Shim, H.S. DistView: support for building efficient collaborative applications using replicated objects, in *Proceedings of CSCW '94* (Chapel Hill, October 22-26, 1994), pp. 153-164.

15. Ross-Flannigan, N., "The Virtues (and Vices) of Virtual Colleagues," in *Technology Review*, Cambridge, MA: MIT, March/April, 1998.

16. Shen, H. and Dewan, P. Access Control for Collaborative Environments, *in Proceedings of CSCW'92* (Toronto, October 31-November 4, 1992), ACM Press, N.Y., pp 51-58.

17. Universal Serial Bus. Welcome to USB. Available as http://www.usb.org (Accessed September 2, 1998).