COOPERATIVELY CONTROLLED OBJECTS IN SUPPORT OF COLLABORATION

by

Lauren J. Bricker

A dissertation submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

University of Washington

1998

Approved by _____
Chairperson of Supervisory Committee

Program Authorized
to Offer Degree Department of Computer Science and Engineering

Date March 20, 1998 _____

Doctoral Dissertation

In presenting this dissertation in partial fulfillment of the requirements for the Doctoral degree at the University of Washington, I agree that the Library shall make its copies freely available for inspection. I further agree that extensive copying of this dissertation is allowable only for scholarly purposes, consistent with "fair use" as prescribed in the U.S. Copyright Law. Requests for copying or reproduction of this dissertation may be referred to University Microfilms, 1490 Eisenhower Place, P.O. Box 975, Ann Arbor, MI 48106, to whom the author has granted "the right to reproduce and sell (a) copies of the manuscript in microform and/or (b) printed copies of the manuscript made from microform."

Signature _____

Date _____

University of Washington

Abstract

COOPERATIVELY CONTROLLED OBJECTS IN SUPPORT OF COLLABORATION

by Lauren J. Bricker

Chairperson of the Supervisory Committee: Professor Steven L. Tanimoto
Department of Computer Science and Engineering

Developing applications to support collaboration by two or more users involves special challenges. An application designer should anticipate the interaction between the user and the computer, as well as the interactions between the users. Secondly, there is a lack of system support for development of software that accepts simultaneous input from multiple users. The Colt system is designed to address both of these issues. The design of the Colt system is motivated by a model of computer-supported collaboration that measures communication and cooperation between the users during an activity. We present a definition and several examples of "Cooperatively Controlled Objects" that have been designed to support cooperative interactions. A user study is presented that illustrates the use of the model and Colt's analysis tools. The Colt system itself is comprised of three parts, a design methodology, a software toolkit, and visualization and analysis tools. The design methodology aids 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 cooperative objects, and support for input from multiple users. Each cooperative object 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 see if the implementation meets the original design criteria. Colt is unique in its support for the development of co-present simultaneous computer based activities. Finally, we describe five programs developed with the Colt system that incorporate this cooperative control of objects.

TABLE OF CONTENTS

LIST OF FIGURES

Number *Page*

LIST OF TABLES

PREFACE

During high school (from 1980-1982), video game parlors were so popular there was even one in the little strip mall about a half a mile from our house in Ann Arbor, Michigan. I was a video game junkie, spending my allowance the day it was received on Space Invaders™, Asteroids™, Lunar Lander™, Galaga™, Centipede™, and the like.

Recently I remembered a game called Space Duel [47] from Atari that came out during my senior year. Space Duel was a variant of Asteroids, the difference being that two people could work together to fight off the space debris barreling towards them. The game offered a choice between two different synchronous modes of play: attached or unattached. The former, where the players were connected by a "tether" show in Figure 1, was my favorite, although it was difficult to find friends who would also want to play. In the tethered version both players could rotate their individual ships, but since they were connected to each other, they would move based on the direction and relative amounts of thrust given by each player. If the ships were pointed away from each other and the players attempted to move at full thrust, both would stay in the same place. As difficult as it was to move while being connected to your partner, it was advantageous to have a friend protect the rear of your ship from the encroaching asteroids.

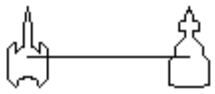

Figure 1. Two players working together in Space Duel.

My love of this game makes me realize all the more that it is only fitting I've chosen to do my thesis on Computer Supported Collaboration.

ACKNOWLEDGMENTS

As I have found over these last few years, completing a Ph.D. is quite a collaborative activity, something I could not have done without the help and support of so many people.

First and foremost, I'd like to thank Dr. Steven Tanimoto for guiding me through this process and teaching me to express my ideas, both formally and informally. I'd also like to thank the other members of my committee, Dr. Alan Borning, Dr. Earl Hunt, and Dr. Linda Shapiro for their help and guidance through this process. I'd also like to thank Marla Baker and Emi Fujioka, for being patient through the many redesigns of the system, and Chris Chapman for helping out with the user study.

I am forever grateful to my family for their constant love and support. My parents, Audrey and Jerry Bricker, encouraged me to attend to graduate school and to actually complete the Ph.D. I found out I was pregnant with my son, Matthew Bricker-Mounts, the day I started in graduate school. He was such an understanding little man during the times I wasn't able to play with him. My sister, Jacqueline Bricker-Critchfield, brought me her unique insights into the world. Her crazy sense of humor kept me sane during these many years. And I must not forget my newest family, Lupe Ortega and the whole LA clan, who welcomed me in even though that meant Ruben stayed up in rainy Seattle.

I must also thank all my friends, especially Ron Critchfield, Erik Selberg, Mary Kaye Rodgers, Karen Ng, Sean Sandys, Elizabeth Walkup, Michelle Malloy and Joe Huber (and the kids), and the Quoven (Qate, Pquatty and Jacqui), for being understanding when I wasn't able to play with them as often as I would have liked.

Y a Ruben Ortega, mi amor y mi vida, sin su amor y apoyo no puedo hacer este. Quieres ser mío por sesenta o setenta años?

DEDICATION

To all the kids, young and old, who love to explore and learn.

CHAPTER 1: INTRODUCTION

1.1 Research problem

Most software developers know that designing and implementing interesting computer applications is difficult, particularly when a user is involved. A software development cycle takes a lot of creativity and planning and, frequently, many iterations of coding and testing the product. Gould and Lewis [37] note that in the design of most applications or systems, the behavior of the algorithms can be rationally analyzed. However, adding an interface that allows a human to interact with a computer, or a *human-computer interaction*, reduces the analytical nature of the system dramatically. They liken a human to a "coprocessor of largely unpredictable behavior" [37, p. 305] that must somehow interact correctly with the system. Myers also points to some reasons why a user interface (UI) is hard to build. Often the code for a UI comprises a large portion (greater than 50%) of the total code, and it requires, among other things, the ability to handle multitasking, real-time programming, and gracefully handle errors that may occur. Furthermore, a UI is rarely considered correct the first time it is designed. Typically a program must be redesigned over many iterations to satisfy the majority of the users [75].

A typical human-computer interaction is a solitary thing – one user manipulates the state of the computer through a set of input devices (such as keyboards, mice, etc). Recently, there has been an interest in supporting *collaborative interactions*, where two or more people work together to contribute a portion of work on a task, on one or more computers [6] (as a regular expression, this could be written as a human(-human)*-computer(-computer)* interaction vs. a human-computer interaction). We will use the term *Computer-Supported Collaboration* (CSC) to mean cooperative or collaborative interactions that are enabled through the use of computer systems and software.

Designing and implementing an application that supports cooperative interactions is even more difficult than a single user application because of the added complexity of the hardware or software involved in maintaining data consistency. It also can be difficult for designers invent good collaborative activities that consist of more than assigning a project to a group of people to do together. In addition, whereas anticipating the interactions between one user and a computer can add a level of complexity and uncertainty to a system, anticipating how a group of distinct people will interact with the machine(s) and each other is next to impossible.

The focus of this research is to aid the design and development of applications for CSC. There are a number of issues to be considered for CSC, including why the users are cooperating, how frequently they are communicating and where they are located when they are working together. 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 the act of working together. However, not every cooperative interaction requires that the people work at the same time or communicate often. Certain situations or objects could potentially encourage or enforce people to work simultaneously together and communicate frequently.

The issue of location of the collaborators has changed dramatically in the last 100 years. Before the advent of technology such as telephones (video or audio), computers and networks, people typically were forced to cooperate in face-to-face situations or through written messages delivered by a courier. Today, collaboration can occur with people located across the hall or across the world, but the quality of the communication and cooperation can be quite different in these two situations. Enhancements in technology may eventually reduce this difference to where the interactions between a group of people will be the same no matter where the individuals are all located.

There are many design considerations for CSC that might be missed during the design phase of an application. What can aid an application designer to determine how the users should and could interact in a simultaneous manner? Once a designer has set out

requirements for a collaborative application, how does a person proceed with the implementation? This dissertation presents two solutions to the problems involved with developing software for CSC. It first presents a general model of communication in CSC. The model can be used in two ways; it can be used, along with the worksheet (shown in Appendix C), as a tool to help focus the developers during the design of a collaborative application. The model can also be used to predict the quantity and type of communication events that occur between users of a collaborative activity. The second solution is a software package to support the implementation of collaborative activities. This package includes a shell application, an application program interface (API) of "Cooperatively Controlled" objects, and support for various multiple input devices. The package supports the development of end-user programs that enforce a tightly synchronized interaction and facilitate communication and cooperation between two or more users.

1.2 Motivation

People cooperate in many ways every day at work, school or home, either to make a task easier or for purely social enjoyment. For example, co-workers on a software design team will meet together often to brainstorm ideas for a new project. The project is generally divided into subprojects, with each team member assigned to work relatively independently on one or more parts. As the project nears conclusion, the team will meet more often to adjust and combine their subprojects. *Computer Supported Cooperative Work* (CSCW) is the study of software and systems designed to support an activity (such as the one above) that is coordinated between two or more people [6, 29, 98]. CSCW is generally focused on how to make a group work more efficiently or productively together. *Groupware* is a term commonly used to describe the software that has been developed to support a cooperative work interaction.

In contrast, the goal of a *Computer Supported Collaborative Learning* (CSCL) application may not be to make the users to work most productively together, but rather it

may increase the amount of problem solving, learning and communication between students. CSCL is based on collaborative learning, a concept that was popular prior to the 1930s and has recently regained popularity in North American schools [65]. Collaborative learning is designed to encourage communication of concepts, discussion of solutions, resolution of 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.

Another reason to support cooperative learning was well stated by Davidson [23] in relation to mathematics. Group learning addresses some of the problems associated with the isolating nature of a typical mathematics curriculum. Many students working in a group become less discouraged and frustrated than students working alone do. The group is not only a source for additional help, but it becomes a support network for members.

Combining computers with group learning is an attractive possibility for education, because computers can empower students to construct and explore objects and worlds. More importantly, computers may offer a better environment for learning than a standard instructional setting. Researchers [62, 63] have noted that computers offer an avenue for students to interactively learn and explore concepts, rather than passively listen to them in a lecture situation, although there is some debate as to the efficiency of this type of learning [60, 62]. There is also evidence that the use of computers helps to motivate students to learn. For example, Monteferrante found that fewer students dropped out of the introductory college calculus series when a computerized learning system was included in the course curriculum [72].

Unfortunately, most schools today only have one or two computers per classroom, which must be shared among the students in that room, or one computer room, which is shared by the entire school. Enabling multiple students to work simultaneously on one computer is an obvious cost saving for school districts, many of which can not afford to provide every child with a computer.

On the whole, CSCL is not only useful in encouraging cooperation and communication (as is non-computer supported collaborative learning), but also may add an enjoyable aspect to tasks, which may help students to stay focused on learning. CSCL, however, is not a panacea in solving the problems of students' not learning in schools. There are many potential problems with integrating computers and collaborative learning into the classroom. Keeping students focused can be difficult in general, but the ability to socialize with classmates, use strange exciting new technology and perform personalized activities can be even more distracting. Lack of teacher experience with the techniques and equipment also adds to the problem of a CSCL environment. Furthermore, some students may not work well together and different discussion styles or argumentative strategies may hinder the students. And finally, there is the problem of how group work should be evaluated. If some members of a group contribute considerably more or less to a project, a single evaluation will not be fair to all or indicative of individual performance.

Although learning is our motivating domain, collaboration is an important issue in entertainment as well. 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 [51], 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 Doom, Quake or Diablo, 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. We are primarily interested in focusing on cooperation where the users are simultaneously working together. This type of interactions may be best suited for learning and entertainment activities, but can be used in a CSCW situation.

1.3 Overview of dissertation

This dissertation presents a software toolkit and design methodology to support the development of CSC activities. Chapter 2 presents a categorization of previous work in CSC. Chapter 3 describes a high level model of CSC applications, followed by details of the program execution model in Chapter 4 and the model of the user's interactions in Chapter 5. Chapter 4 also defines and demonstrates examples of cooperatively controlled objects and includes a discussion of some issues encountered in developing user interfaces for them. Chapter 6 outlines the Colt system, which includes a methodology to help focus the design of a collaborative activity, a toolkit to aid in the development of collaborative activities based on the design, and tools to analyze the data collected from people using the activity. The Colt system is comprised of a hierarchy of cooperatively controlled objects and a framework in which collaborative applications may be developed. Chapter 7 describes activities that have been designed specifically to encourage users to work simultaneously and communicate frequently to solve a problem. Finally, Chapter 8 describes some directions for future work.

CHAPTER 2: PREVIOUS WORK IN COMPUTER-SUPPORTED COLLABORATION

2.1 Taxonomies of computer-supported collaboration

There are many ways to classify techniques and systems for Computer-Supported Collaboration (CSC). One method, outlined by Ellis, et al. [30], includes categories based on the functionality of the application such as: message systems, multi-user editors, group decision support systems (GDSS), electronic meeting rooms, computer conferencing, intelligent agents, and coordination systems. It is, however, difficult to classify many CSC Systems (CSCS) based on a taxonomy of this type since it does not encompass every type of application of interest. Furthermore, many CSCS may fall into more than one distinct category.

Another attribute that is used to categorize a CSCS is the type of activity the system supports. Unfortunately, a spectrum based on whether an application supports collaborative learning or cooperative work is not a straightforward one. There has been a lot of debate in the CSC community as to whether CSCL and CSCW are one in the same, really quite different, or are related along a spectrum. Companies are increasingly interested in using collaborative systems to train employees [54]. For example, the Team Training shell is a generic scheme for developing support for training teams, such as medical trauma teams, to work together [27]. Systems such as these can be classified as both CSCL and CSCW.

Furthermore, even the applications commonly classified as CSCL systems support a wide range of activities. Many may be placed on a spectrum between “discovery” within an information space, and activities that involve object “construction” (see Figure 2).

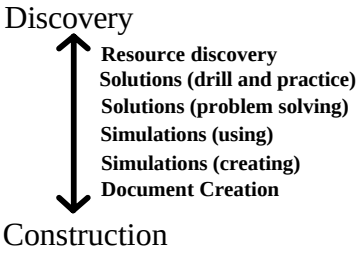

Figure 2. A spectrum of CSCL activities.

A *Resource Discovery* activity is one in which the users are charged with exploring the world, be it through the World Wide Web (WWW) (such as CoVis [34]), or searching through an on-line card catalog of a library [105]. For a *Solution* based activity, the students are given a task of answering questions (drill and practice) or solving a problem given using their knowledge or resources at hand, and perhaps a bit of creativity [100]. A *Simulation* based activity may model social or physical reality. It can be either strictly a discovery activity where the equations necessary to run the simulation are pre-programmed [19] or it could be one in which the users must create a simulation from scratch to explore interactions between elements in a pseudo world [97]. Finally, *Document Creation*, could be described by activities such as writing fiction, writing a paper or drawing a picture. Users of this type of activity must rely on their previous knowledge and previous resource discovery to create and formulate an entirely new entity [11, 81, 82].

As with the taxonomy based on the functionality of the application (described above), there is a wide range of activities between discovery and construction. This classification also has the same problem as the other; aspects of more than one category may be present in some applications.

The two most common methods for categorizing CSC systems are based on the physical locations of the users and the simultaneity of their interactions [30, 84]. These taxonomies are orthogonal and appear to divide systems into the most straightforward categories.

In the case of the physical location of the users, CSC applications may be divided into

two types: distance and co-present collaboration. *Distance collaboration* is one where users work together but are physically separated by barriers (such as walls) or large distances and can not see or hear each other with technological assistance, such as audio or video conferencing. *Co-present collaboration* (also know as face-to-face or co-located collaboration) allows users to interact directly, see each other's expressions and gestures, and therefore communicate more effectively. As video teleconferencing technology improves and gets cheaper, this dichotomy will become less clear. For now, however, the low degree of apparent presence available and the lack of actual contact hamper cooperation at a distance with other users.

For large co-present groups (e.g., a classroom full of students), collaborative applications may run on a set of computers interconnected on a local area network. Alternatively, one or more computers may support the users in a “meeting room” configuration, similar to CaptureLab [29]. Occasionally one computer, or a *Single Display system*, is used by a small group of users sharing it simultaneously. While sharing of a single computer may present some logistical problems to the users, the necessity of sharing can promote communication amongst the users. Physically separating users to work individually on computers tends to discourage communication.

Typically CSC categorizes cooperative interactions in terms of how simultaneous the users work together. An *asynchronous cooperative interaction* is one in which two or more users work on a task at separate times, typically in separate locations [84]. An example of an asynchronous cooperative interaction is delivering information through snail mail or email messages. Participants of this type of cooperative interaction do not expect an immediate response from questions or requests. A *synchronous cooperative interaction*, on the other hand, is one in which two or more users interacting on the same task in real-time. Phone calls or electronic talk (etalk) are examples of a synchronous cooperative interaction.

The following sections will describe activities based on these final categories to help motivate the design of the Colt System.

2.2 Examples of asynchronous distant activities

The most common support for asynchronous distance-based collaborative activities is the user of electronic mail or Internet news. University classes often create mailing lists or news groups as a method to disseminate information or as a forum for discussions. Recently, a World Wide Web (WWW) based news systems, such as NCSA's HyperNews, has been deployed in courses to support group discussions and collaboration on assignments. Many courses in Computer Science and Engineering, as well as other departments, at the University of Washington use this system.

The Computer-Supported Intentional Learning Environment (CSILE) [82] and the Collaboratory Notebook [28], also support asynchronous distance learning. Both systems are based on a knowledge database or *knowledgebase* that is developed by students over time. In a typical activity, the students are given a topic to research and discuss. The users publish a research entry in the database where others can or ask questions about it, comment on it, or agree or disagree with it. Symbols help users to distinguish different types of entries in the database. Both CSILE and the Collaboratory Notebook do encourage the users to discuss a given topic, The nature of these systems, however, do not allow the users to communicate interactively. Users work predominately independently to collect data, and discuss topics, pose problems or solutions, or clarify issues asynchronously through the knowledgebase.

2.3 Examples of synchronous distance activities

A number of applications encourage communication across a network by focusing users on common objects. Most games that can played across a network, such as Warcraft, Civilization II, Ages of Empire, Doom, Sim City 2000 Network Edition, Settlers 2 and Ultima 2, are competitive in nature, although some do allow users to join together to defeat other groups. Still, most users only interact with each other when they are in the same virtual room. Furthermore, dialog between the users, if any at all, occurs when they type messages to each other. Users may attempt non-lexical communication with each

other by changing portions of the activity, such as on screen personas (avatars). For example, in most versions of Quake it is difficult to differentiate between multiple teammates in a virtual room. A user can request by typing in the chat window that one teammate invoke the jump command. When one avatar moves up and down, the user is able to associate that teammate's avatar from the others in that room.

TurtleGraph [57] is a problem-solving environment where users move a cursor with Lisp commands. As with the games listed above, the environment allows typewritten communication between students through a chat window. This application was developed with a cumbersome communication system that, in theory, makes the communication between users more productive.

The systems listed above all use a text-based communication system similar to electronic talk (etalk), Multiuser Dungeons (MUDs, MOOs or MUSHs) or WebChat (an etalk system implemented for the WWW). Other, more recent systems such as Comic Chat [65] or BlackSun Interactive [12] allow users to create their own avatars. The graphical representations are designed to give the users more physical presence during a chat session. While they do allow users to communicate, these systems are not designed to focus users on a specific task and have no support for sharing applications.

In contrast to the applications listed above, GroupWeb [37], and Turbo Turtle[19] not only focus the users on a specific task with specific objects, they also allow the users to communicate with each other through an audio channel. Additionally, these systems use affordances such as telepointers and other "awareness widgets" to indicate to others where a user is working in the system. These applications are considered to be examples of *Strong Sharing* as defined by Tou, et al [104]. Strong Sharing is the ability for a user of a distance synchronous application to witness the changes to shared data made by other users in real-time.

Adding real-time audio/video conferencing may enhance synchronous networked collaboration. Until recently, CSCW systems such as RAVE [34] or Clearboard [53]

were available but required expensive specialized hardware. NetMeeting [65], on the other hand, is a teleconferencing system that is freely available from Microsoft. Users in both RAVE and NetMeeting create conferences in a special window and can work on shared applications simultaneously or by taking turns. Since the visual channel is in a separate application, users must constantly shift their focus between the current task and the video window to see other user's gestures. Collaborators in the Clearboard system, on the other hand, appear behind a user's workspace, allowing him or her to see both simultaneously.

Unfortunately, the hardware required by these system still inhibits direct visual access to partners. Thus these systems may not foster the same level of communication as one finds in face-to-face situations. Additionally, while many of the systems presented in this and the previous section assist collaboration, in many cases the users still work independently on a task, communicating infrequently to join their work to complete the activity. We believe there are some circumstances in which a common focus and close communication are desired. For example, in an entertainment situation, tight cooperation may increase the level of enjoyment for the users. In an educational context, tight cooperation may facilitate the learning process.

2.4 Examples of asynchronous co-present activities

Most computer applications are written for a single user on a computer. A group of people interested in working on a task on a standard system cannot work simultaneously without special hardware and software support. They must either agree that one person will control the input resources (mouse, keyboard, etc), or they must agree to take turns. In the former situation, group members who are not in control often feel left out or lose interest in the task. In the latter situation, turn taking will often break down social conflict erupts over the use of the input devices.

Nastasi and Clements [76] studied groups of students programming in Logo on single machines. This research showed that while there was social conflict over the input

devices, the students learned through a process of discovering and resolving cognitive conflicts. The results of their study suggest that the collaborative use of Logo encourages interactions, which supports discussions and coordination of different ideas and higher order thinking.

Inkpen [52] found that pairs of students were able to solve more puzzles when working together on a single computer then when working alone or on separate computers side-by-side. However, the users of this first study were hampered by their contention for the single input resource (the mouse). The software was adapted such that each student in the pair had access to his or her own mouse, although only one mouse could control the cursor at a time. Inkpen then studied the achievement of students using a "give" or "take" protocol for switching who had control over the cursor. Inkpen found that girls using the "give" protocol solved more puzzles than girls sharing one mouse. Moreover, she found that boys seemed to work best using the two-mouse "take" protocol.

Stewart [99] has also studied the topic of software for co-present collaboration and he coined termed the subject "single display groupware." He has modified KidPad to allow multiple users to work simultaneously. His prototype system, Sushi, is currently being developed.

2.5 Examples of synchronous co-present activities

In contrast to the previous section, meeting room systems [29, 98] are specifically designed to allow multiple users, each on their own machines, to work collaboratively and still maintain direct face-to-face communication. Some of these systems include special tabletop monitors that do not obstruct views and/or one large main monitor that is shared by all and serves as a mechanism to synchronize views.

While these systems do encourage communication more than their distributed counterparts do, they do not evoke the excitement and degree of interaction shown by users of co-present video games. Of course, such games are designed to maximize the

players' excitement and suspense. However the lack of excitement may also be due, to some extent, to limitations of the underlying hardware, such as delays due to network bandwidth, or perhaps it is because the users of such systems are not focusing on exactly the same view of the given problem. Even in systems with a large shared view screen users must constantly shift their attention from their own monitor to the front of the room to gain context.

On the other hand, MMM [11] is a multi-user system in which each person has control of his or her own mouse, and all are working on one screen. MMM allows users to focus on the same task without having to shift their attention from their personal displays to the group's screen. This is the same situation found in many interactive home video games developed by Nintendo, Sega and Atari. Unfortunately most of the games developed for these platforms encourage competition or only allow collaboration in the form of turn taking.

Bricker et al [12] presented four co-present CSCL applications designed to promote tight cooperation between users on a specific activity. These applications, *The Color Matcher, Chord Matcher, Curve Fitter and Midpoint Activity* have definite, predefined goals that the users must attain by working closely together. Additionally, the players have access to their own mice, which correspond to colored cursors on the screen. These activities are briefly described below.

2.5.1 The Color Matcher

Figure 3 shows the Color Matcher user interface. The interface contains two color frames; on the left is the target color and on the right is the users' color frame. The users' color is determined by combining the values of its red, green and blue components they are controlled by horizontal scroll bars. Each scroll bar can be modified only by the cursor of the same color. Although each user is limited in what they may control, the users collaborate to reach the single goal of matching the target color. In order to succeed at this activity, users must, at a minimum, communicate about the relative

amounts of color needed.

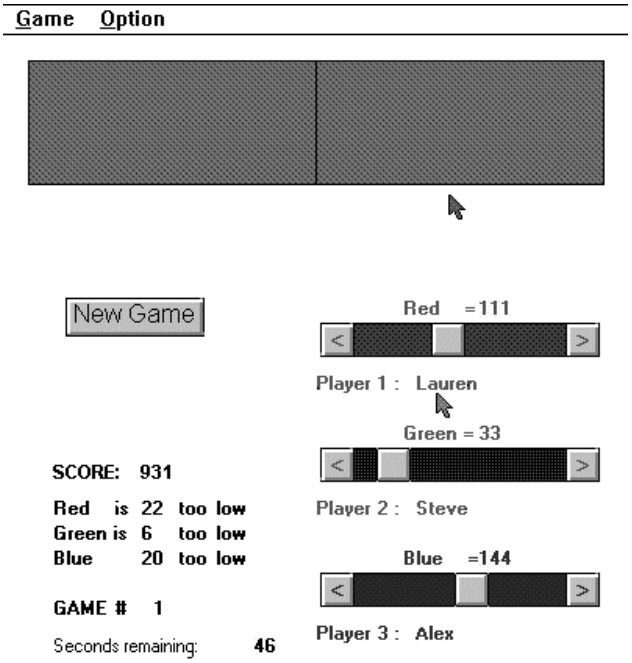

Figure 3. The Color Matcher user interface.

2.5.2 The Chord Matcher

The Chord Matcher application, shown in Figure 4, is very similar to the Color Matcher, except that the users must match musical chords rather than RGB colors. The chords vary from a relatively easy major chord, to more difficult selections of three random notes of a five-octave chromatic scale. Each user selects his or her note from the five-octave range using a color-coded scroll bar. As with the Color Matcher, each user can only operate the scroll bar of the same color as his or her mouse cursor. In order to attain the goal of this activity effectively, students must talk about the pitches in the chord and when to check their answer.

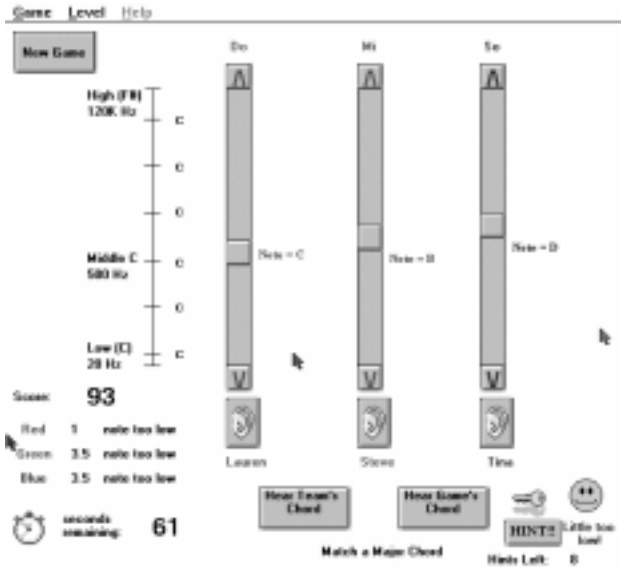

Figure 4. The Chord Matcher user interface.

2.5.3 The Curve Fitter

The Curve Fitter application, shown in Figure 5, allows students to explore how the location of control points defines the shape of a polynomial curve. As with the Color Matcher and Chord Matcher, each user controls a colored cursor on the screen with his or her own mouse.

The users are charged matching a polynomial based on a shape in an image within 120 seconds. The users choose the image, the degree of the polynomial (between 1 and 5) and the number of participants (between 2 to 4) before starting the activity. They are then presented with $n+1$ control points to modify the shape of a degree n polynomial. The active users may only move the control points that are the same color as their cursor. If there are more control points than the number of active participants, some users may have to manipulate more than one control point.

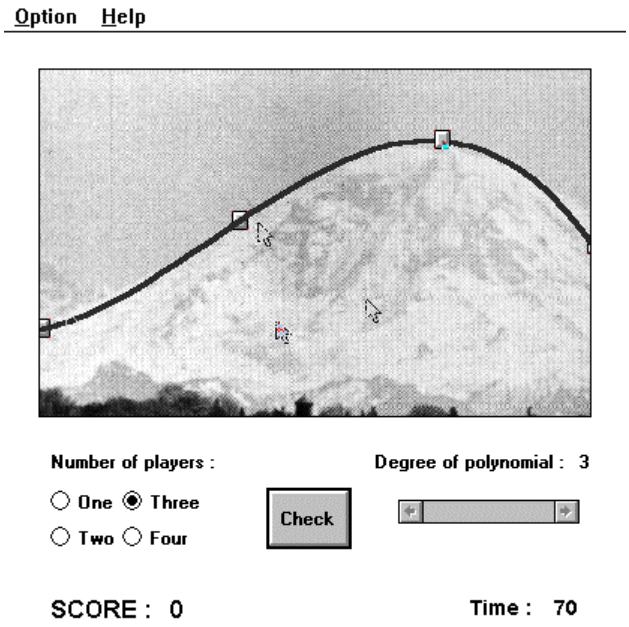

Figure 5. The Curve Fitter user interface

2.5.4 The Midpoint Activity

The Midpoint Activity, shown in Figure 6, may be played in a two-person or three-person mode. In this activity, each user is in charge of the colored point matching his or her mouse cursor. If two users are active, the goal is to match the midpoint of the line segment defined by their points with a predefined point on the screen. When there are three users engaged in this activity, they must match the centroid of the triangle defined by their points with the specified point. But there's a catch - the movement of each of the users' points is limited to areas on the screen defined by bitmapped images. This activity is designed to encourage communication by focusing the users on certain geometrical objects.

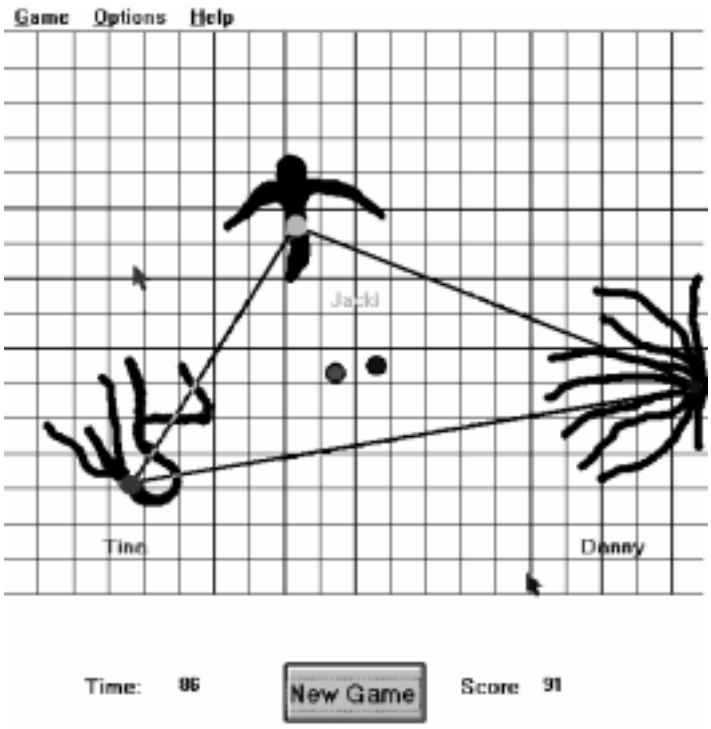

Figure 6. The Midpoint Activity user interface.

2.6 Why support highly synchronous cooperation?

Many common non-computer based activities contain elements of highly synchronized cooperation, such as a three-legged race, recording music or moving large objects. In the case of a three-legged race, a 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 goal of this type of race is not to be able to run faster than as an individual, people do it because it can be entertaining.

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. Although, in theory, one person could do this task, 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 by their entertainment value. 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.

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. As stated in Section 2.4, pairs of students were able to solve more puzzles when working in a cooperative situation. Additionally, the preliminary testing of the four MultiIn-based multiplayer activities described in the previous section [13] provided anecdotal evidence that the students tend to communicate effectively about the problems posed while using highly synchronous co-present applications. These activities contain objects that allow close synchronous interactions among users working closely on a task. The testing of the MultiIn activities also indicated that using multiple-input devices eliminated contention for input resources.

The second reason to support highly synchronous activities is to help focus and motivate students on a learning task. Students are less likely to learn when they are not actively participating in a classroom activity. For some students, working closely with another person to complete a task is a motivation to stay focused on that task. This was informally observed during the testing of the METIP software [64] and Color Matcher

activity [13] in an 8th grade math class in a middle school in Seattle, Washington. When we introduced the METIP software to the students, we gave them specific tasks to accomplish during the testing. During the testing we noticed that some of the students were not focused on the task, instead, they would spend time gossiping and passing notes to each other. The teacher informed us that this was normal for this grade level. Although our initial goal had been to show that the Pixel Calculator and Image Warper software increased the students' interest and learning in Mathematics, by the end of the testing we were pleased we were able to keep them focused on the task. In contrast, when we tested the Color Matcher software with students from the same class, we noticed that they were able to stay focused on the task during the course of the testing. Admittedly, the smaller groups were separated from from the rest of the class during the Color Matcher study, but the amount of focus shown by the users was a promising sign.

2.7 Summary

This chapter has presented a number of methods for categorizing computer supported collaborative applications and focused on one, the classification based on where the users are locating and how they synchronize their activities. It also has presented a number of collaborative systems that can be in this way.

Our research in CSC assumes that effective collaboration requires some amount of communication between the participants. We are particularly interested in exploring whether computer-based objects that encourage or require cooperative control promote an increase in or certain types of communication among the users. In light of our interest, there are four main objectives to this research:

- to develop a model of interactions among users and between the users and collaborative software that will allow (Chapter 3 - Chapter 5),
- to develop methods to measure how effectively the users are cooperating (Section 4.4) and communicating (Section 5.3.3),

- to design and develop a class of objects for use in collaborative applications that encourage or require close synchronous cooperation among the users (Section 4.5.2), and
- to help designers in the process of designing, implementing and testing activities that allow users to cooperatively control objects (Chapter 6).

We feel there are many reasons to support highly synchronous cooperative interactions, when the users are either co-present or at a distance. Although we are most interested in co-present synchronous interactions, we begin by presenting a general model of the human(-human)*-computer(-computer)* interactions. The next chapter will describe a high level overview of this model of CSC.

CHAPTER 3: A MODEL OF COOPERATIVE ACTIVITIES

3.1 Background

A number of researchers [19, 30, 40, 41, 42, 82] have studied factors that contribute to Computer Supported Collaboration (CSC) software not being accepted or used. Grudin [41] points to five factors that contribute to the failure of Computer Supported Cooperative Work (CSCW) applications, including the disparity between who does the work and who benefits from the work and the disparity between the interactions enforced by the software and the current social practices. Furthermore, most applications for CSC do not handle unexpected user interactions, which he terms exception handling or improvisation. In general, the intuitions most designers have for developing software for groups is based on single-user applications and is not correct for multi-user applications. He also notes that it is much more difficult to evaluate and test a group application than one designed for a single user [40, 41]. Applications that can be used by multiple people must be tested on groups or the effects of group dynamics will be missed. One difficulty arises in maintaining the groups long enough to do valid testing. These problems indicate that a generic interface design will not work well for both single and cooperative activities. Furthermore, a cooperative interface should not be designed based solely on experiences with single-user applications.

Characterizing the problems of existing software is the first step towards attaining good, usable collaborative applications. Grudin [42] offers eight challenges of groupware development and use. Based on his work, Cockburn and Jones [19] outlined four principles for good groupware design. While they are somewhat obvious, these principles are recommendations on how to create cooperative software. More formal methods include design languages, user models and system models. Although these methods are more difficult to develop and use, they are specific enough that they can

directly translate into an application implementation or they can predict or analyze specific aspects of human-computer interactions.

An example is the Cooperative Systems Design Language (CSDL) [25] that aids in the design process by supporting the development of cooperative applications from specification to implementation. The language formalizes an architecture for cooperative applications and includes primitives and support for object sharing and control, as well communication control.

User models have a basis in cognitive psychology, human factors engineering, as well as computer science [70]. They are a method for adapting a system to the characteristics, needs and intentions of the user [15]. System models, on the other hand, generally do not include any information about the user. They emphasize system level operations, such as communication, document storage, and tasks, in simpler, functional representation. Models are used in two ways: as a method to analyze a task, which may help in generating design ideas, or as a tool to analyze how an application is used, once it is designed or implemented [43, 59]. Models can be used to evaluate or predict learnability (how fast a user can learn the system), usability (can a user effectively use the system), or efficiency (how fast a user can use the system) of an interface. In large, real-time systems, models can also be used to perform safety analyses to predict whether or not a user is able to process vast and complex amounts of information presented to them.

For example, the Optimal Control Model (OCM) [9, 87] can predict a user performance and system reliability in tasks where data is continuously monitored, such as power plants, air traffic control, flight management, and military systems. The Operator Function Model (OFM) [70, 71] is a hierarchically defined model designed to determine if the users (operators) of a highly complex dynamic systems can internalize the information displayed and can carry out the appropriate control functions. Both OFM and OCM include a model of the user as well as the system, and focus on how the two interact.

The Task Action Grammar (TAG) [85] model is used to predict how easily a person can learn or use a machine command language (such as MS-DOS) [85]. TAG is designed to specifically model the user's competence, or what the user can do, rather than what the user actually does. Olson [79] described a procedure for transforming another task-based object oriented model, the Object-Oriented Analysis Model (OOAM), into an object-oriented software design. This software design can be directly translated into an object-oriented implementation.

GOMS is another hierarchical model that breaks a task down into its Goals, Operators, Methods and Selection Rules [58, 59]. It was intended to predict the performance of a user in the early design and prototyping stages, but it has been mostly used as a tool to analyze existing interfaces and as a tool during the redesign process. John and Kieras [58] compare and contrast the assumptions and predictive power of four variants of the GOMS model: the original Card, Moran, and Newell GOMS (CMN-GOMS), the Keystroke-Level Model (KLM), Natural GOMS Language (NGOMSL), and the Cognitive-Perceptual-Motor GOMS (CPM-GOMS). While all of these variants include some model of the user, the CPM-GOMS model specifically includes primitive operators that are perceptual or cognitive acts on the part of the user.

Other models have attempted to represent the complexities of cooperative software. Items [74] is a system model to aid in the development of complex cooperative multimedia software. The model describes five independent view types including work scenario, scenario flow, document, or workplace scenario and configuration scenario views. Cahour and Salembier [15] argue that user and system models must be included to describe cooperation, and that they are really two sides in the same interaction. They focus on interlocutor modeling (the person taking part in a dialog) and describe three situations where it is crucial. They also point out limitations in current user models for describing certain cooperative interactions.

Finally, Taylor et al, [103] analyze a model of dialog typically used in artificial intelligence (AI) planners. These models take into account the mutual beliefs, intentions,

reasoning, and capabilities of the users, and typically are used to create conversations with simulated (computer-based) agents. They prescribe some assumptions and requirements for their type of model, which they claim “seems to provide the best model of what happens during human conversation.”

In contrast to these efforts, we have chosen to model collaborative activities in order to measure or predict how much communication and cooperation occurs during such an activity. The model, presented in the next subsection, will be used to design a class of “Cooperatively Controlled Objects,” described in Section 4.5.2, which can be employed to encourage more communication between the users.

3.2 A model of computer supported cooperative activities

One of the contributions of this research is a model of cooperative activities, similar to the conceptual model of groupware developed by Ellis and Wainer [31]. Their model contains descriptions of three parts of a cooperative system: the objects and operations users can perform on those objects, dynamic aspects such as control and data flow, and the interface between a user and the system or other users. Our model is similar in that it accounts for the computer, the users and their interactions. However, since communication is fundamental to a cooperative interaction, our model of cooperative activities serves a different purpose. One of the goals of our model is to provide a conceptual structure for the design of new activities. Another goal is to facilitate measuring the amounts of various kinds of communication among the people involved in the activity. This model is also used to aid the design of highly interactive objects that require users to communicate frequently, presented in section 4.5.2.1.

In describing this model we assume that a collaborative activity includes a cooperative program and the external description of a “problem” for the users to solve. Hayes humorously defines a problem this way: "Whenever there is a gap between where you are now and where you want to be, and you don't know how to find a way to cross that gap, you have a problem." [45 p. xii]. Davis gives a more formal definition of a problem: “A

problem is a stimulus situation for which an organism does not have a ready response" [24, p. 12].

We define a *problem description* to consist of an initial state, goal states or criteria the users are expected to reach, any constraints enforced on the users in reaching that goal, and a possible scoring criteria. The goal may be a fully specified state (or set of states) the users must reach or the problem may be an open-ended and exploratory, where there is no specific best solution. An example of a problem with an end state is the Color Matcher Activity in Section 2.5.1. The goal of that activity is to have the users work together to match the red, green and blue value of a specified target color in under sixty seconds. The program starts in a state where the users' color is black, or has an RGB value of (0, 0, 0).

The goal of a game or activity is typically used by the designer of an activity as a way of focusing the users and their communication. This model also assumes that the users are committed to reaching the goal jointly and, as in [103], that there is no malicious intent or deception on the part of the users.

Figure 7 depicts the high level, cyclical view of this model of cooperation, which includes users, a collaborative computer program, and the presentation of the initial problem description. The description may be specified by the software itself, it could be given (say orally or on a sheet of paper) to the users, or it can be inferred from the software itself. Briefly, during the course of the activity the users perceive the state of the program, how the program is used, how the users work together, and the goal of the given task. From these perceptions, each user builds an internal context of the problem. Group discussions and decisions, such as how the problem could be subdivided into smaller sub-problems, are also perceived added to the context. The internal context influences the user's decisions on how to proceed. Based on their decisions, the users act to modify the state of the program. The users then perceive the results of their changes and decide what to do next. We can illustrate this cycle of perception, decision and action using the Color Matcher activity. As stated above, the users are presented with the

goal of matching the target color using red, green and blue scroll bars. They will perceive the difference between their starting state (where the users' color is black) and the target color. The group might discuss possible solutions before moving or the users might decide independently how to move their own scroll bars. As the users move their scroll bar, their color will be updated on the screen. The users will perceive this change, and make new decisions accordingly.

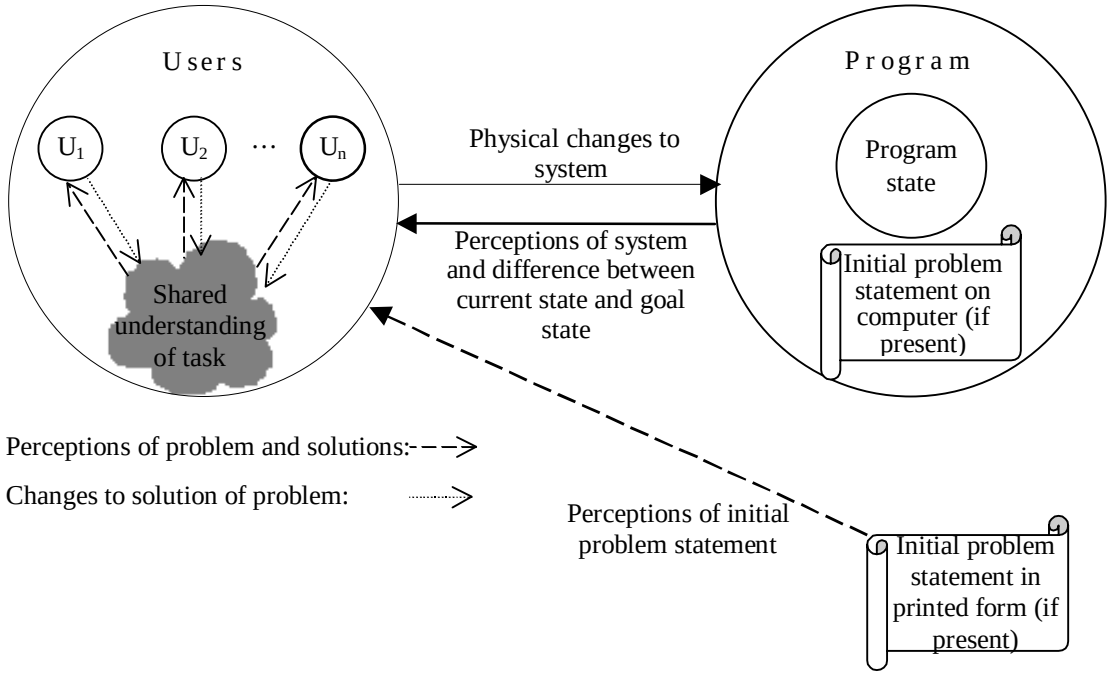

Figure 7. Overall model of users' interaction with a cooperative program.

This model is general enough that it can be used to describe the interactions in any computer-supported collaborative activity. The model does not specify the location of the users, the method of communication (directly or through hardware), or the type of problem description. An alternative example to the Color Matcher would be a open-ended, exploratory activity with a problem description such as "Use CSILE (as described in Section 2.2) to build a knowledgebase about the Western Snowy Plover." Recall that CSILE is an application that is used asynchronously by a group at a distance. As in the Color Matcher example, the users perceive the initial state of the program (perhaps an empty knowledgebase), how the program works, and the task. The users might discuss

their perceptions and decide how to proceed as a group, or they might initially proceed independently. The state of the program changes when a group member adds entries into the knowledgebase. The other users will perceive the changes to the state of the knowledgebase, and those perceptions may influence what they research and their subsequent entries in the knowledgebase.

The next section will give a brief analysis of how the users interact with the computer in "steps." The model of interaction between the multiple users and the computer will be described in Chapter 4 and the interaction between the users themselves will be discussed in Chapter 5. We will continue use the Color Matcher activity (described in Section 2.5.1) in examples to emphasize certain points in these chapters.

3.3 Analyzing interactions in terms of steps

As stated in the previous section, the problem description includes an initial state of the program, a set of goal states or criteria, any constraints enforced on the users, and possible scoring criteria. During the course of the activity, the users manipulate the software and the state of the program will change which, in turn, changes the users' perceptions of the state and their internal context of the problem. Presumably part of this internal context is the users perception of the difference between the current state and the goal state they wish to reach. This context affects how they choose to accomplish the task. Thus, the nature of the task will evolve over time.

It would be difficult to measure the part of this cycle that involves the changes to the user's state of mind. As observers, our only clue as to how the users perceive and internalize the problem is our own interpretation of their external behavior. Many psychologists have tried to understand how humans solve problems and how they can become better problem solvers. Hayes [45] likens a person's search for a problem solution to paths through a maze. Some sequence of available moves will reach the goal, while other may lead down paths that dead end and require backtracking. He lists four basic techniques employed by problem solvers to search through the space of possible

problem solutions, including trial-and-error (blind or systematic), proximity searching (constantly reducing the difference between the current state and the goal), fractionation (or dividing into sub-problems) and knowledge-based (learned solutions and pattern matching) methods. The users may determine their strategy before starting the task or they may decide as they are solving the problem. The users will undoubtedly choose their course of action based on the current state of the software, how they perceive the group is working together, and their own knowledge of the history of the activity to that point, including what may have caused them to backtrack, if at all.

While it is impossible to completely know and describe a user's internal context, the many changes to the state of the program can be described as in Sections 4.2 and 4.3. As the activity progresses, the program will be in many of different states, some of which might hold more significance to the user than others. For example, in the Color Matcher the user will move the thumb of the red scroll bar to different positions corresponding to different values for the red portion of the users' color. The users, however, might focus on the change in the red value from the initial value, corresponding to the location of the thumb and cursor when the left mouse button was pressed, to the final value corresponding to the final location of the thumb, where the mouse button was released. The system states in between resulting from the mouse and thumb moving may not be important to the users.

It may also be impossible to determine how each user perceives the difference between the current state and their understanding of the goal. Each user may have a very subjective measure of how "close" they are to the goal and whether or not they are making progress in the activity. However, the differences between two states of the system could be measured, although the measure used will depend on the activity. We denote this measure as $diff(s_i, s_j)$ where s_i and s_j are two states of the system. Using the Color Matcher as an example, the difference between two states of the red scroll bar could be measured as the absolute value linear distance between the values of the scroll bar states, or $diff(s_i, s_j) = abs(value(s_j) - value(s_i))$. Based on this measure, if the current

state *c* of the scroll bar has value 99 and the goal state *g* has value 103, the *diff(c, g)* is 4. The users probably will not know the actual numerical difference between the two values; instead they will perceive that they are pretty close to the goal.

Since a designer is unable to determine exactly what states the users will deem significant, he or she must infer it from the type of actions possible in the activity. The designer of the Color Matcher might surmise that when the user starts or stops moving the thumb of the scroll bar will seem significant to that user. We define the changes from one significant state to another a *step* the user has taken towards solving the problem. (We note that Hayes also uses the term step in describing problem solving techniques but does not formally define it.) The result of taking step *s* in state *a* takes the system to state *b*. The users' purpose in taking the steps to solve the problem is to end in a goal state *g* defined by the problem description.

We define a step to be *productive* if the difference between the ending state (*b*) and the goal (*g)* is less than the difference between the starting state (*a*) and the goal state. Formally, a productive step is one in which *diff(b, g)* < *diff(a, g)*. An example of a productive step in the Color Matcher would be if a user changes the red value from 0 to 35, if the red value of the target color is 47. Using the same measure described above, the diff(35, 47) = 12 < diff(0, 47) = 47.

A *trivial step* is one in which the user makes a series of modifications to the system, but the starting state and ending state are the same, or if *diff(b, g) = diff(a, g)*. A trivial step in the previous example would be if the user starts modifying the red value at 35 and winds back at 35. The concept of a trivial step will be revisited in Section 4.4.3, when we relate it to the significance of a computer-based event.

Intuitively, an *unproductive step* is one in which the user does not reduce the difference between the current state and the goal, or *diff(b, g)* > *diff(a, g)*. An unproductive step in the previous example would be if the user changes the red value from 35 down to 28. Here diff(35, 47) = 12 > diff(28, 47) = 19.

These three definitions can be rewritten more generally as a *measure of productivity*. If a step *S* is defined as starting in state *b* and ending in state *a* towards reaching a goal state *g*, and there is a difference measure defined between two states, then the measure of productivity can be defined as in Equation 1.

Equation 1. The definition of the measure of productivity of a step starting in state *a* and ending in state *b*.

$$productivity(S, g) = diff(b, g) - diff(a, g) = P \text{ is } \begin{cases} productive, \text{ if } P > 0 \\ trivial, \text{ if } P = 0 \\ unproductive, \text{ if } P < 0 \end{cases}$$

Determining the productivity of an individual step may not be enough to determine if progress is being made in an activity. Take flying from Washington State to Oslo, Sweden as an example. Travelling west from Seattle (say to Hawaii) would be an unproductive step because the distance from Hawaii to Oslo is greater than the distance from Seattle to Oslo. Flying west from Yakima to Seattle is also an unproductive step by the same measure. However, when combined with the productive step of flying via the circumpolar route from Seattle to Oslo, we find that this unproductive step is a better start then first flying east, say to New York City, and then across the Atlantic ocean.

Another example is a maze where the users may chose to move down one path, only to find that it is a dead end and they must backtrack. Although the series of steps that brought them closer to the goal appear productive, they did not get them to the goal. Thus, it is more meaningful to combine the productivity measures of a series of *n* successive steps that have brought the system to state s_n to determine if a group is making progress in the activity. We define the *measure of progress* of *n* successive steps $\{S_0, .. S_n\}$ in Equation 2. If the total magnitude of the productive steps is greater than the total magnitude of the unproductive steps, $progress(\{S_0, .. S_n\}, g)$ will be positive, or we can say the group is making progress towards the goal.

Equation 2. The definition of the productivity of n steps to state s_n.

$$progress(\{S_0, .. S_n\}, g) = \sum_{i=0}^{n-1} diff(s_i, g) - diff(s_{i+1}, g)$$

The definitions of steps, measures of productivity and progress are useful to determine if users are making progress on the activity. Admittedly these definitions are estimates because we, as observers, cannot really tell what is significant in the users' minds. However, they may give an observer insight as to how well a user is working on an activity. The hope is that as users become more skilled at the activity and at working with each other, the relative weight of the productive steps will increase and the weight of unproductive and trivial steps will decrease. The next chapter, in Section 4.4 particularly, will revisit this concept in a slightly different way to measure how the users work together.

CHAPTER 4: MODELS FOR HUMAN-COMPUTER INTERACTIONS IN COMPUTER SUPPORTED COLLABORATION

4.1 Prologue

There are many methods that can be used to describe the design of a computer program. In the previous chapter we discussed models that focus on one element of a computer program, such as the human-computer interface design and our general model for collaboration using a computer. Since our interests include understanding and measuring how multiple users interact with a computer program, we are primarily interested in modeling programs that takes input as events from multiple users. In this chapter we present a model that captures the essential properties of a cooperative computer program (the *program* in Figure 7) and defines measures of the users collaboration based on their joint interactions with the computer.

In the next section, we will explore two general methods that can be employed to describe the design of a computer program, finite automata and Statecharts. This section will also introduce the notation that is used in the rest of the chapter. Section 4.3 presents a simplification to focus on the input from multiple users required in collaborative programs. Section 4.4 defines our method for measuring the collaboration between users and Section 4.5 presents methods aimed helping designers to focus users on a collaborative task. The Color Matcher activity, which was described in Section 2.5.1, will be used throughout this chapter as an example to illustrate the points raised.

4.2 General program models

4.2.1 Notation

A typical application can be described as containing a set of objects. One method used to describe changes to the state of a system is to track the state variables that can be modified during the execution of the program. (We could also track the unchangeable properties, but that would be uninteresting.) We denote the set of modifiable objects in the system $O = \{O_1, O_2, \ldots O_m\}$. Each object O_i has some set of n_i properties, $P_i = \{p_{i1}, p_{i2}, \ldots, p_{in_i}\}$, that can be modified. We note that each object may have a different number of properties, and that not all of the properties on an object will be modifiable during the course of the activity. The set of all modifiable properties in the system $P = \{p_{11}, p_{12}, \ldots, p_{1n_1}, p_{21}, p_{22}, \ldots p_{2n_2}, \ldots, p_{m1}, p_{m2}, \ldots p_{mn_m}\}$, where p_{ij} is the j^{th} property of the i^{th} object.

In the physical world some properties, like the location of a point on a computer monitor, may have an infinite number of possible values. We will assume that the computer representation is simplified with a finite set of discrete values, V_{ij}. Nevertheless, this set may be large, for example, the set of possible pixel locations for a point on a screen is large (i.e. 1024 x 768 = 786,432). Thus the finite set of possible states of the system, consists of all possible vectors of values $(v_{11}, v_{12}, \ldots, v_{ij}, \ldots, v_{mn_m})$ where $v_{ij} \in V_{ij}$ for property p_{ij}. This finite set of possible states is defined as Q in finite automata, which wil be presented in the next section.

As an example, let us use this model to describe the Color Matcher from Section 2.5.1. Table 1 specifies the objects, properties and possible values used to describe a system state for this activity. Each object does contain more properties than listed, but we are focusing only on certain properties of interest to simplify the presentation of the model for this activity.

Table 1. A partial view of objects, properties and possible values for the Color Matcher activity

Object (O)	**Property (P)**	**Possible Values (V)**
Red scrollbar	Relative X position	0-255
	Selected	Yes/no
Blue scrollbar	Relative X position	0-255
	Selected	Yes/no
Green scrollbar	Relative X position	0-255
	Selected	Yes/no
User's color	Red Value	0-255
	Blue Value	0-255
	Green Value	0-255
	Scored	Yes/no
Target Color	Red Value	0-255
	Blue Value	0-255
	Green Value	0-255
Check button	Depressed	Yes/no
	Released	Yes/no

As stated above, Q is the set of all possible vectors of values for each property for objects in the application, or $Q = \{(v_{11}, v_{12}, \ldots, v_{ij}, \ldots, v_{mn\ m}) \mid v_{ij} \in V_{ij}$ for property p_{ij} of $O_i\}$. An example of a state q is the vector of values shown in column V of Table 2 and the initial state (q_0) is shown in Table 2, right most column. The set of 768 possible final states include states where each of the red, green and blue scroll bar has one of 256 values, and the user's "scored" property is set to *yes*. Since the values for the red, green and blue component of the users' color directly corresponds to the values of the corresponding scroll bars; they don't increase the number of final states.

Table 2. A partial view of an example state of the Color Matcher Activity.

Object (O)	Property (P)	Possible Values (V)	Initial Value (q_0)
Red scrollbar	Relative X position	57	0
	Selected	No	No
Blue scrollbar	Relative X position	90	0
	Selected	No	No
Green scrollbar	Relative X position	198	0
	Selected	No	No
User's color	Red Value	57	0
	Blue Value	90	0
	Green Value	198	0
	Scored	No	No
Target Color	Red Value	55	55
	Blue Value	128	128
	Green Value	226	226
Check button	Depressed	No	No
	Released	No	No

A cooperative application allows a set $U = \{U_1, U_2, \ldots, U_p\}$ of one or more users to manipulate the objects in the system. Each user manipulates the objects through devices $D = \{d_1, d_2, ..., d_p\}$, which in turn generate a finite set of input events I. Note that there is a direct mapping between each U_j and d_j. The set $I = I_1 \cup I_2 \ldots \cup I_p$, and each I_j is the set of events allowed by each user and contains a null event (λ) for the case when a user is not currently submitting input to the system. Input events are system dependent, but commonly include: left mouse button depressed (LMD), left mouse button released (LMU), right mouse button depressed (RMD), right mouse button released (RMU), mouse move (MM), keyboard character entry [ASCII value 0-255] (C#, where # is the ASCII value), message send (MS), message received (MR).[1] The input alphabet, defined

[1] Multiple keyboard character input by a single user (such as the keys "jkl" all depressed simultaneously) is not handled in this model since this is not found in most applications. An exception to this is key modifiers such as shift, control, alt, escape, etc. Should the user press multiple keys, such as "jk," at the same time, we will assume that the system serializes the input to be "j" then "k." For "control-k", on the other hand, the control key is considered a modifier of the "k" key, not a separate keystroke.

as Σ in finite automata, consists of the set of vectors $(i_1, i_2, \ldots i_p)$ where each $i_j \in I_j$.

Although most of the recent cooperative applications do not support multiple users simultaneously sharing a keyboard, it can be represented in this model by allowing the character inputs from each user to only include keys from "their" part of the keyboard. Take for example a game where one user operates keys in the set {s, d, f, e} and another uses {j, k, l, i} to move a screen object in any of the directions in the set {left, down, right, up}. Here we have $U = \{u_1, u_2\}$, $I_1 = \{\lambda, s, d, f, e\}$, and $I_2 = \{\lambda, j, k, l, i\}$. Then Σ = {(s, λ), (s, j), (s, k), (s, l), (s, i), (d, λ), (d, j), (d, k), (d, l), (d, i), (f, λ), (f, j), (f, k), (f, l), (f, i), (e, λ), (e, j), (e, k), (e, l), (e, i), (λ, λ), (λ, j), (λ, k), (λ, l), (λ, i)}.

The input alphabet for the Color Matcher is described as $\Sigma = \{(i_1, i_2, i_3) \mid i_j \in I_j\}$. For this example we assume the possible inputs from each user are the same, that is $I_j = I$ = {λ, LMD, MM, LMU}, for j = 1, 2, 3.

4.2.2 Finite Automata

The finite automaton is a well known, general mathematical model of a system represented using a quintuple $(Q, \Sigma, \delta, q_0, F)$, where

- Q is the finite set of possible states of the system,
- Σ is a finite input alphabet
- δ is the transition function which maps Q x Σ into Q (often described pictorially),
- q_0 is the initial state, and
- F is the set of possible final states.

As stated in the previous section, in order to benefit from the structure of finite automata, we have chosen to limit the possible values of the object properties to a finite set. The set Q consists of all possible vectors of values $(v_{11}, v_{12}, \ldots, v_{ij}, \ldots, v_{mn\,m})$ where $v_{ij} \in V_{ij}$ for property p_{ij}. We note that $q_0 \in Q$, and $F \subseteq Q$. The input alphabet Σ consists of vectors of events $(i_1, i_2, \ldots i_p)$ where each i_j is a possible input event from user j. Defining Σ in this ways allows us to express the ability of the users to modify an object simultaneously.

Timing is an implicit aspect of the model since each transition from one state to the next is assumed to happen over some period of time. The transition function for most programs appears very complex when depicted as a Finite Automaton. For example, Figure 8 shows a pictorial description of part of the transition function δ for part of the Color Matcher activity. This diagram depicts δ when a user manipulates the red scroll bar. The figure only shows the possible numeric scroll bar values and whether or not the red scroll bar is selected. Other state variables such as the location of the thumb position or static variables such as the size, color and location of the scroll bar have been omitted to simplify the diagram. Furthermore, the states of other objects, such as the blue and green scroll bars and the user's color, are not listed due to how many possible states exist.

The user begins by pressing the left mouse button down on the thumb of the scroll bar to select it. Once the scroll bar is selected, the user can set a value for the scroll bar by moving the mouse to a different location. The user ends the interaction by releasing the left mouse button when the thumb has been positioned in the desired location.

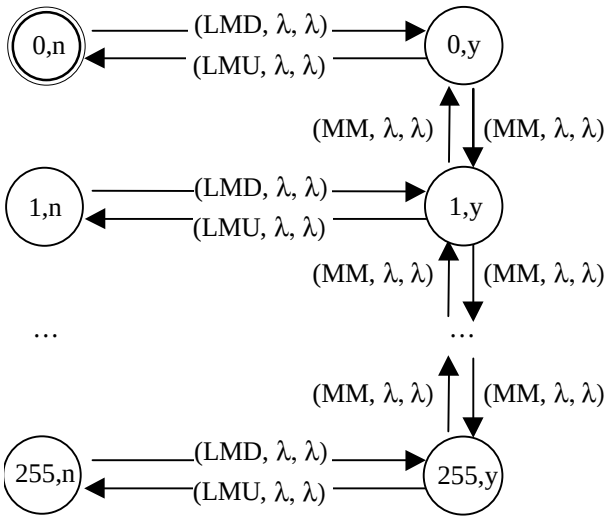

Figure 8. Pictorial representation of the portion of the transition function δ when a user manipulates a scroll bar. The pair inside each circle represents a state vector of the scroll bar value and whether or not the scroll bar is selected (with y for yes or n for no).

4.2.3 Statecharts

As evidenced in the previous section, finite automata can be awkward in describing complex systems because the number of possible states can get very large [47]. This is due to a combinatorial explosion of possible states, if there are **m** state variables and each variable has a possibility of **n** values, the total number of possible states for the system is $\mathbf{m}^{\mathbf{n}}$. For example, the state of a point in a square region could include the x and y coordinates of its location, as well as the color described by the red, green and blue values of the point (thus there are 5 state variables. If there are 256 possible values for each color component and, for simplicity's sake, the square region is 256 pixels on each side, the total number of states is 5^{256}, or greater than 8.6×10^{178}! A state transition diagram for a program that required states such as these would be, as Harel states, "unstructured, unrealistic and chaotic" [47], and hence unreadable and useless.

One method to reduce this problem of exponential growth of the state space is to extend the definition finite state machines and diagrams to include hierarchy, concurrency and communication. Statecharts [47] are a visual formalism designed to capture these extensions and can be used to specify complex discrete-event systems, including human-computer interactions.

In contrast to a transition diagram for a finite automaton, we are able to more succinctly depict the event model for all three scroll bars and the users color is shown in a Statechart diagram, shown in Figure 9. Statecharts use a more complex syntax than Finite Automata to depict a state transition diagram when the model contains a very large number of possible states. As with the finite automata for the red scroll bar, the Statechart description for each scroll bar shows that the starting state is unselected with value 0, and visually describes how the value of a scroll bar is changed when the scroll bar is selected and then moved. The Statechart syntax simplifies the description of the transition diagram by having a text description of how the value of the scroll bar changes when a specified user moves their mouse when the scroll bar is in its selected state, rather than specifying all 256 distinct values. For example, instead of listing all of the states of the

selected scroll bar on the right in Figure 8, Figure 9 shows “on Red MM: change red scroll bar position.” The Statechart also specifies that when a user moves their mouse in the selected scroll bar, the group’s color will change to reflect the new position.

Statecharts also allow hierarchical nesting of states. There are two levels of nesting in this example. On the top level the red, green and blue scroll bars exist in the fine-grained sharing version of the Color Matcher activity. This nesting may not seem necessary for the original version of the Color Matcher activity described in Section 2.5.1, but it will be used to describe a new version of the Color Matcher that will be used in Chapter 5 and described in detail in Section 7.6. (The new version allows the users to choose their color by one of two methods, depending on the state of a program variable.) Inside the box for each scroll bar are two sub-states for whether or not the scroll bar is selected. Two additional notations used in Statecharts to simplify the transition diagram are the *default state* and the *history*. The default state is shown as the solid ball with the arrow pointing to the scroll bar unselected state. The history is shown by the H-entry (the circled H) and specifies that the system will enter the last state of the object.

Another software modeling method, the Unified Modeling Language (UML) includes Statecharts for visualizing the transitions between the states of the objects in the system. UML was designed based on other object-oriented modeling languages such as the Booch, Object Modeling Technique (OMT), and Jacobson methods [89]. This model allows a designer to specify objects, properties, and methods and the relationships among them in a software program. Rational Rose is a graphical application for developing UML models of a software design [89]. We have used Rational Rose to present the design of the Colt system in Chapter 6.

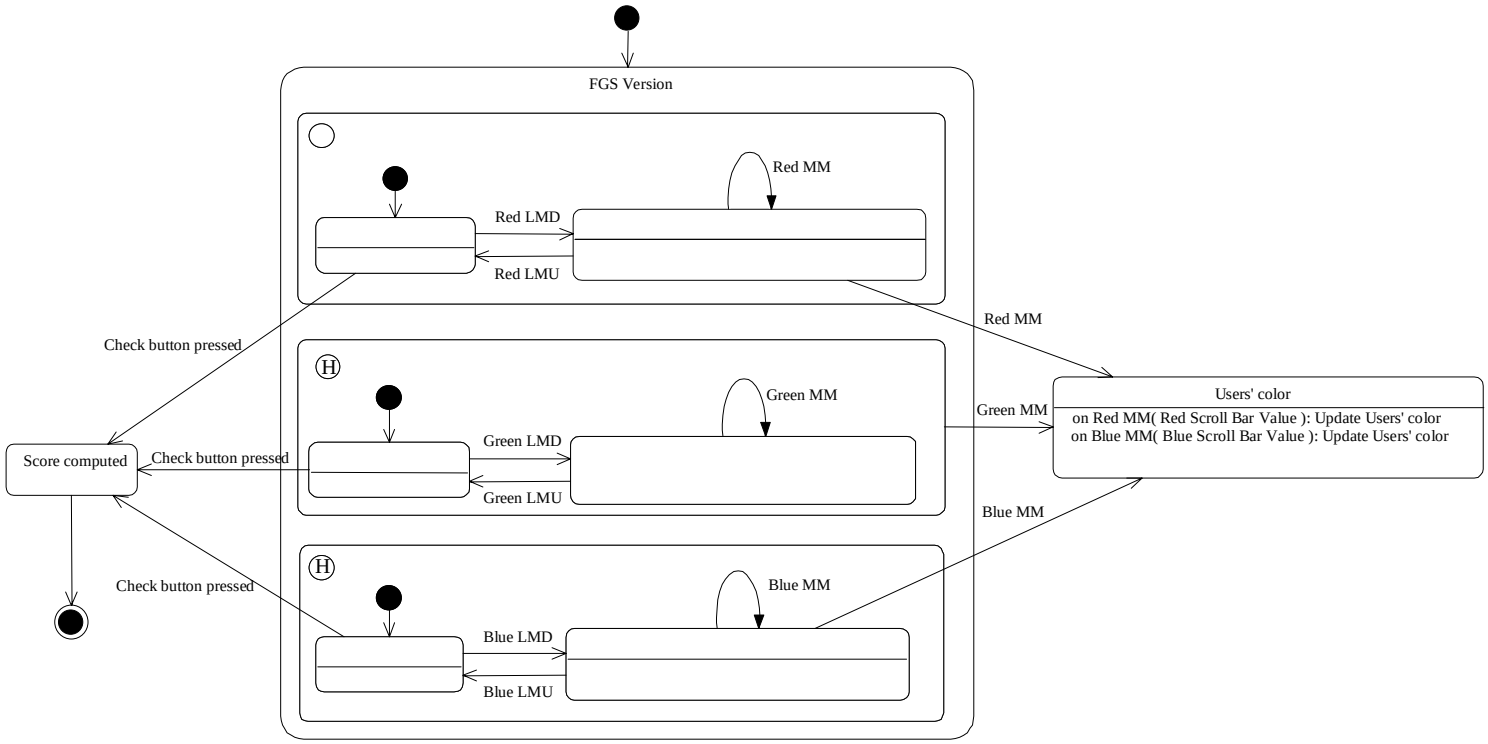

Figure 9. Statechart model for the Color Matcher Activity

4.3 Simplified program execution model using combined input events

The previous section discussed using Statecharts to simplify a complex state diagram that includes cooperative input from multiple users. Another method to simplify a complex state diagram is to modify the domain of the alphabet Σ to contain fewer symbols, but which have more complex meanings. In order to accomplish this type of simplification we must make some assumptions about the type of system being used. In particular, we will assume that each user of the activity being modeled has a separate an input device, and each input device generates distinct input events.

We may choose to represent how the simple events from the input devices combine to create a more complex event. For this example, we will assume that each input device generates messages such as: *left mouse button depressed* (LMD), *left mouse button released* (LMU), *right mouse button depressed* (RMD), *right mouse button released* (RMU), *mouse move* (MM). We also assume a keyboard generates messages consisting of keyboard character entry [ASCII value 0-255] (C#, where # is the ASCII value). Regular expressions, or variants such as flow expressions, can be used to describe the combination of device events. Regular expressions (RE) are powerful enough to describe how an individual user's device inputs combine to form a program event, but they will not describe the case where concurrent multiple input by multiple users is required for state transitions. *Flow expressions* (FE) have been used to model concurrent execution of programs [92]. FEs allow additional operators, such as the shuttle operator, to extend regular expressions, and they are a more expressive method for describing concurrent input by multiple users. For example, if a user's cursor is placed over an object, a sequence of *(LMD MM* LMU)* mouse events may be translated into a *Move* event on that object. However, if an object cannot move until two users are actively moving it, we might indicate this by the flow expression $JointMove = ((LMD_1\ MM_1{}^*\ LMU_1)\ @\ (LMD_2\ MM_2{}^*\ LMU_2))$. This indicates that each subsequence of events *(LMD MM* LMU)* must occur in order, but that the two users must do their move operations in parallel.

Table 3. Simple events from input devices combined using regular or flow expressions to form complex program events.

Events from input device	Program Event	How event may be used
LMD LMU	Select	If over an object, select that object
$(LMD_1\ LMU_1)$ ◎ $(LMD_2\ LMU_2)$	JointSelect	Two users over an object, select that object
LMD MM* LMU	Move	If over an object, move that object
LMD MM* LMU (or LMD LMU)	Create	If not over an object, create an object
$(LMD_1\ MM_1{}^*\ LMU_1)$ ◎ $(LMD_2\ MM_2{}^*\ LMU_2)$	JointMove	Two users over an object, moving that object simultaneously
RMD RMU	Rotate	If over an object, rotate that object 90 degrees[2]
C[1-255]*	Type	Creating a message or editing an existing message

Table 3 lists examples of complex program events that can be formed from simple device events described using regular expressions. We note that program events typically used in a graphical system include mirroring horizontally or vertically, zooming in orout, deleting or creating an object, hiding or revealing an object, moving an object back or front, and changing properties of the object. Combining the device events to create program event can simplify the state transition function by reducing the number of possible states. Figure 10 shows an example of how the number of states in Figure 9 can be reduced by removing the unselected and selected red, green, and blue scroll-bar states and combining the device events *LMD*, *MM*, and *LMU* to form the program event *Move*.

[2] Although not commonly used in interface design, the complex event Rotate = RMD RMU will be used in the model of the puzzle activity, described in Section 7.4.

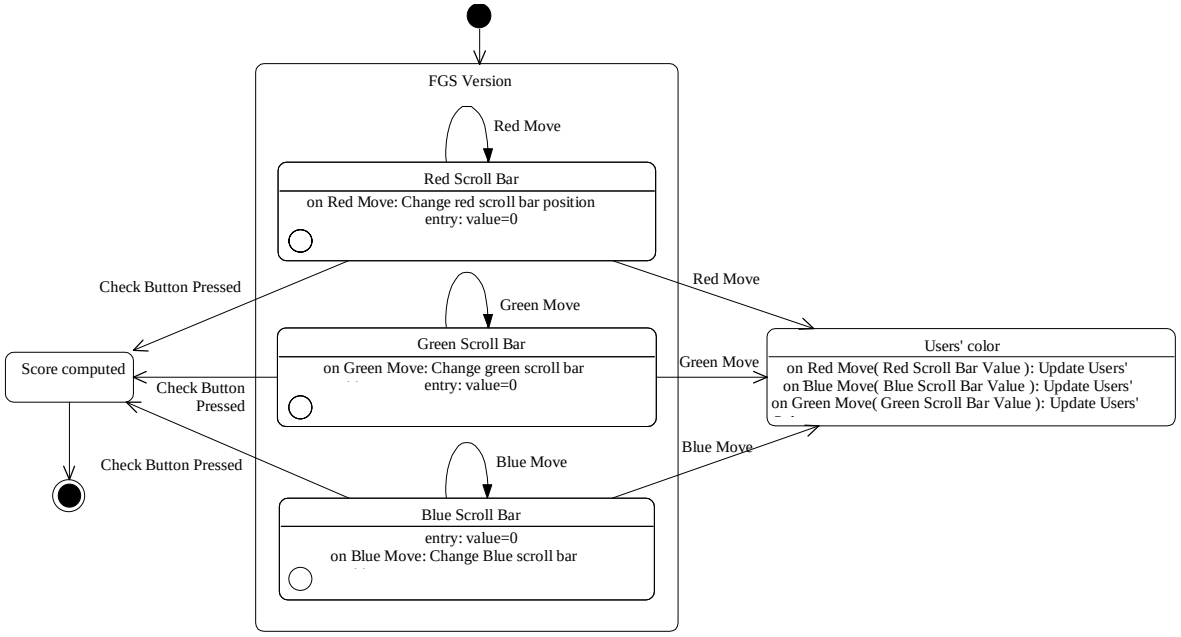

Figure 10. Modified statechart using program events rather than device events.

4.4 Measuring user participation

As stated in Section 2.6, one reason to support highly synchronous collaboration is that it may help users stay focused on a task. One possible measure of a user's involvement is to measure how much each user physically participated in the activity. This type of information could 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 form of feedback could also be helpful to a teacher who needs to determine how much each member of a collaborative group contributed to the solution of a problem.

We will begin by defining an *Event Span*, or the period of time over which an event occurs, and relate them to simple and program events as described in the previous section. Using event spans we will define two metrics of physical (as opposed to verbal) participation in a cooperative activity. The first, a *measure of joint activity*, will summarize the amount of individual and simultaneous group action during a given period of time in the activity. The second, a *measure of significance*, will describe the quality of each user's participation in the activity.

4.4.1 Program event spans

Most program events (as defined in Section 4.3) take some non-trivial amount of time to complete. If we assume that each simple event *i* takes some amount of time t_i to occur, the total time *t* to complete a complex event is given by Equation 3.

> Equation 3. Total time *t* for a program event as a sum of the times t_i for simple events.

$$t = \sum_{i=1}^{n} t_i$$

For example, as stated in the previous section, a *Move* event can be represented as *LMD MM* LMU*. If we assume that a simple *LMD* event takes t_d seconds, one *LMU* requires t_u

seconds and each *MM* event takes t_m seconds, then a *Move* event will take $t_d + t_u + n * t_m$ seconds to complete, where *n* is the number of MM events. Column three of Table 4 lists formulas for calculating the time required to complete various kinds of program events based on the time for each simple event.

Table 4. Time to complete complex program events based on simple events from input devices.

Events from input device	**Program Event**	**Time based on the**	
		sum of the times of the constituent simple events	**difference between start and end event times**
LMD LMU	Select	$t_d + t_u$	$T_{LMU} - T_{LMD}$
LMD MM* LMU	Move	$t_d + t_u + n * t_m$, where n is the number of MM events	$T_{LMU} - T_{LMD}$, proportional to number of MM events
LMD MM* LMU	Create	$t_d + t_u + n * t_m$, where n is the number of MM events	$T_{LMU} - T_{LMD}$, proportional to number of MM events
RMD RMU	Rotate	$t_d + t_u$	$T_{RMU} – T_{RMD}$
C[1-255]*	Type	$n * t_c$, where where n is the number of keys pressed, and t_c is the time for one key press	$T_{C[n]} – T_{C[0]}$, where n is the number of keys pressed

Alternatively, we can assume that *LMD* and *LMU* events are nearly instantaneous and occur at clock times T_d and T_u respectively, and that the series of *MM* events occur between those two times. The *Move* program event will occur over the span of time $[T_d, T_u]$. The *Move* program event will take $T_u - T_d$ seconds to complete. Examples of determining program event times are shown in column 4 of Table 4. More generally, each program event *i* starts at time T_{is}. The time the event completes, or ends is denoted T_{ie} (note that $T_{is} \leq T_{ie}$). We may denote the starting and ending time of event *i* as an *Event Span*, denoted $ES_i = [T_{is}, T_{ie}]$. Thus, we can define the amount of time an event *i* takes to

execute, in seconds, using Equation 4.

Equation 4. Calculation of number of seconds from an event span

$$t_i = Seconds([T_{is}, T_{ie}]) = T_{ie} - T_{is}$$

We note that on some computer systems, event *i* can appear to have $t_i = 0$ seconds, if $T_{is} = T_{ie}$. This will occur if the interval for the computer's internal clock is larger than the actual time the event takes if measured by a "wall clock." For example, if the computer clock ticks every 1 second, an event that, by a wall clock, takes 0.5 seconds will appear instantaneous to the system. We call these events *essentially instantaneous.*

Since event spans are really intervals of time when an event occurs, they can be joined through union and intersection operations. We define these two operators for two event spans $ES_1 = [T_{1s}, T_{1e}]$ and $ES_2 = [T_{2s}, T_{2e}]$ where $T_{1s} \le T_{1e}$, $T_{2s} \le T_{2e}$ (by definition) and $T_{1s} \le T_{2s}$, in Equation 5 and Equation 6, respectively.

Equation 5. Definition for the intersection operator for event spans assuming $T_{1s} \le T_{2s}$.

$$[T_{1s}, T_{1e}] \cap [T_{2s}, T_{2e}] = \begin{cases} [T_{2s}, T_{2e}] \text{ if } T_{2e} \le T_{1e}, \\ [T_{2s}, T_{1e}] \text{ if } T_{2s} \le T_{1e} \text{ and } T_{1e} \le T_{2e}, \text{ and} \\ \varnothing \text{ otherwise} \end{cases}$$

Equation 6. Definition for the union operator for event spans, assuming $T_{1s} \le T_{2s}$.

$$[T_{1s}, T_{1e}] \cup [T_{2s}, T_{2e}] = \begin{cases} [T_{1s}, T_{1e}], [T_{2s}, T_{2e}] \text{ if } T_{1e} \le T_{2s}, \\ [T_{1s}, T_{2e}] \text{ if } T_{2s} \le T_{1e}, \text{ and} \\ [T_{1s}, T_{1e}] \text{ if } T_{1s} \le T_{2s} \text{ and } T_{2e} \le T_{1e} \end{cases}$$

A non-empty intersection of event spans by two different users on the same object indicates that they have been actively manipulating the object simultaneously for some

period of time. Two event spans on an object are *disjoint* if their intersection is empty.

We will use the following example to illustrate points made in this and the next two subsections. Imagine an activity where a group G of three users, $\{U_1, U_2, U_3\}$, are working together to manipulate an object *O*. Each user generates the following event spans on the object:

U_1: [10:50:29, 10:50:45], [10:50:48, 10:50:50], [10:50:51, 10:51:30]
U_2: [10:50:28, 10:50:30], [10:50:32, 10:51:34]
U_3: [10:50:28, 10:50:32], [10:50:33, 10:50:55], [10:51:00, 10:51:25]

A somewhat easier to read, graphical representation of this information is shown in Figure 11.

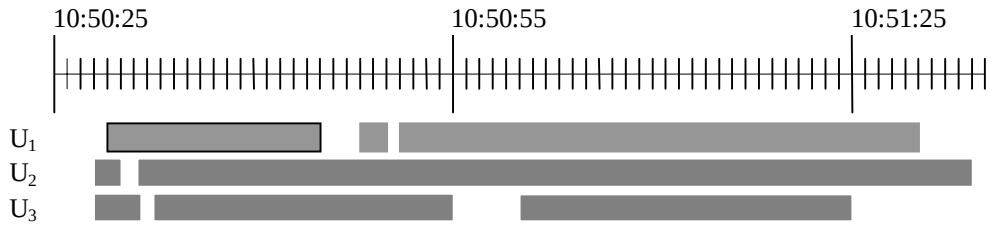

Figure 11. A graphical representation of the event spans generated by three users on one object.

The intersection of the first event spans generated by each user is:

[10:50:29, 10:50:45] ∩ [10:50:28, 10:50:30] ∩ [10:50:28, 10:50:32] =
[10:50:29, 10:50:30].

The union of the same three event spans is:

[10:50:29, 10:50:45] ∪ [10:50:28, 10:50:30] ∪ [10:50:28, 10:50:32] =
[10:50:28, 10:50:45].

Finally, we note that the calculation of the number of seconds in the union of a disjoint set of event spans is the sum of the number of seconds in each event span, as shown in Equation 7.

Equation 7. Calculation of the number of seconds of the union of disjoint event spans.

$$Seconds\left(\bigcup_{i=1}^{n} ES_i\right) = Seconds(ES_1) + Seconds(ES_2) + \ldots + Seconds(ES_n)$$

For example, the number of seconds in the following union of disjoint event spans would be:

Seconds([10:50:28, 10:50:30]∪[10:50:33, 10:50:55]) =
Seconds([10:50:28, 10:50:30]) + Seconds([10:50:33, 10:50:55]) =
2 + 22 = 24.

4.4.2 A measure of joint activity

As stated in Section 4.4, measuring how much the users either independently or jointly manipulate an object may be of interest to a designer or a teacher. For any object in an application, the *measure of joint activity* summarizes this information by weighting the relative amount of time the users work in subgroups. The time the users spend working simultaneously all together weighs heavier than the time spent working in subgroups, and the time the users spend working in subgroups weighs heavier than the time the users spend working individually.

In order to present the measure of joint activity, we must first discuss the "total activity" of a group over a period of time. We begin by stating that there is *activity* whenever any user is actively manipulating an object, that is, when there is a program event on the object generated by a user. The *Total Activity* on an object during the time period $[T_s, T_e]$ by the group of users $G = \{U_1, \ldots U_n\}$ is the union of all the periods of activity starting at T_s and ending at T_e. This is shown more formally in Equation 8.

Equation 8. Total Activity on *O* by all users in the group *G*.

$$TA(O, G, [T_s, T_e]) = \left(\bigcup ES_i\right) \bigcap [T_s, T_e], \forall\ ES_i \text{ generated on } O \text{ by } U_j \in G$$

Using the example,

TA(O, {U_1,U_2,U_3}, [10:50:30, 10:51:30]) =
[10:50:28, 10:51:34] ∩ [10:50:30, 10:51:30] =
[10:50:30, 10:51:30].

This is shown graphically as the interval in Figure 12. Essentially this determined that at least one user was active during the time period [10:50:30, 10:51:30]. Using Equation 7, we can calculate the number of seconds of total activity. This figure will be used later to determine the percentage of total activity that the users worked as individuals, pairs, groups of 3, and so on.

Seconds(TA(O, {U_1,U_2,U_3}, [10:50:30, 10:51:30]) =
Seconds([10:50:30, 10:51:30]) =
60.

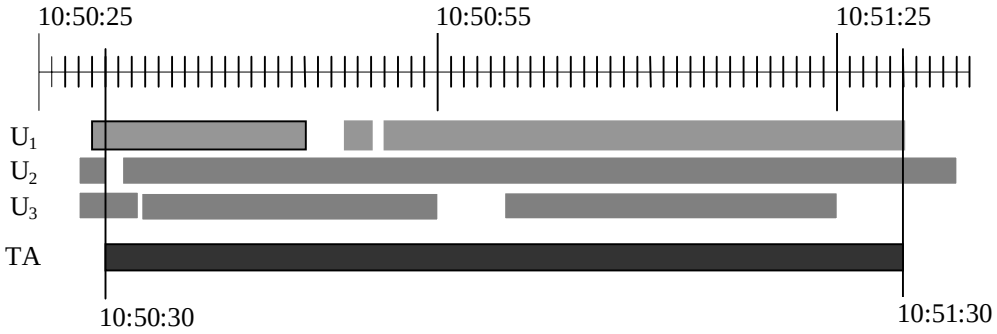

Figure 12. A graphical representation of the total activity (TA) on an object by three users during [10:50:30, 10:51:30].

Next we present the equations to calculate the amount of time the users worked as individuals, pairs, groups of three, etc. Depending on the design of the activity and number of users that can potentially work together, some amount of time during [T_s, T_e] will be spent manipulating an object as individuals or as groups of two or more users. Recall that for any group of *n* users, $G = \{U_1, U_2, U_3, \ldots U_n\}$, the number of subgroups of size *m* can be determined by Equation 9. We will use |G| to denote the number of members of group G.

Equation 9. The definition of the combination function. This is used to determine the number of subgroups of size *m* in group *G*, where $|G| = n$.

$$combination(n, m) = \binom{n}{m} = \frac{n!}{(n-m)!\,m!}$$

We will define *choose(G, m)* indicate the set of subgroups with *m* members in them. In other words, $choose(G, m) = choose(\{U_1, U_2, U_3, \ldots U_n\}, m) = \{SG_1, SG_2, \ldots SG_{combination(n, m)}\} = \{SG_i, \forall |SG| = m\}$. For example, $combination(3, 2) = 3$, and $choose(\{U_1, U_2, U_3\}, 2) = \{\{U_1, U_2\}, \{U_1, U_3\}, \{U_2, U_3\}\} = \{SG_1, SG_2, SG_3\}$.

An equation similar to the one in Equation 8 can be used to determine the ***T**otal **A**ctivity by a **S**ubgroup SG* on an object *O* during $[T_s, T_e]$, or $TAS(O, SG, [T_s, T_e])$. The *TAS* is shown in Equation 10 and uses a function called *Indiv(ES, U)*. The function *Indiv(ES, U)* determines a new set of intervals that are the period(s) of time during an event span *ES* when only *U* is manipulating object *O*. Using the example,

Indiv([10:50:32, 10:51:34], U_2) = {[10:51:30, 10:51:34]}
Indiv([10:50:29, 10:50:45], U_1) = ∅.
Indiv([10:50:29, 10:50:45], U_3) = ∅.

Basically this states that for all of the event spans shown in Figure 11, U_2 worked alone during [10:51:30, 10:51:34], but U_1 and U_3 never worked individually.

Equation 10. Total activity by a subgroup *SG* for an object *O* during a time period $[T_s, T_e]$.

$$TAS(O, \{U_j\}, [T_s, T_e]) = \left(\bigcup Indiv(ES_i, U_j)\right) \bigcap [T_s, T_e], \forall \text{ portions of } ES_i \text{ generated by } U_j \text{ where only } U_j \text{ is actively manipulating } O, \text{ and}$$

$$TAS(O, SG, [T_s, T_e]) = \left(\bigcup\left(\bigcap ES_{i,j}\right)\right) \bigcap [T_s, T_e], \forall \; ES_i \text{ generated on } O \text{ by } U_j \in SG, \text{ where } SG \text{ contains more than 2 users.}$$

Using the example, the time *O* was manipulated during [10:50:30, 10:51:30] by individuals:

$TAS(O, \{U_1\}, [10:50:30, 10:51:30]) = \varnothing$
$TAS(O, \{U_2\}, [10:50:30, 10:51:30]) = \varnothing$
$TAS(O, \{U_3\}, [10:50:30, 10:51:30]) = \varnothing$

by pairs of users:

$TAS(O, \{U_1, U_2\}, [10:50:30, 10:51:30])$ = {[10:50:32, 10:51:33],
[10:50:55, 10:51:00],
[10:51:25, 10:51:30]}
$TAS(O, \{U_2, U_3\}, [10:50:30, 10:51:30])$ = {[10:50:45, 10:50:48],
[10:50:50, 10:50:51]}
$TAS(O, \{U_1, U_3\}, [10:50:30, 10:51:30])$ = {[10:50:30, 10:50:32]}

and by all three users simultaneously:

$TAS(O, \{U_1, U_2, U_3\}, [10:50:30, 10:51:30])$ = {[10:50:33, 10:50:45],
[10:50:48, 10:50:50],
[10:50:51, 10:50:55],
[10:51:00, 10:51:25]}

Again, this can be depicted graphically, as in Figure 13.

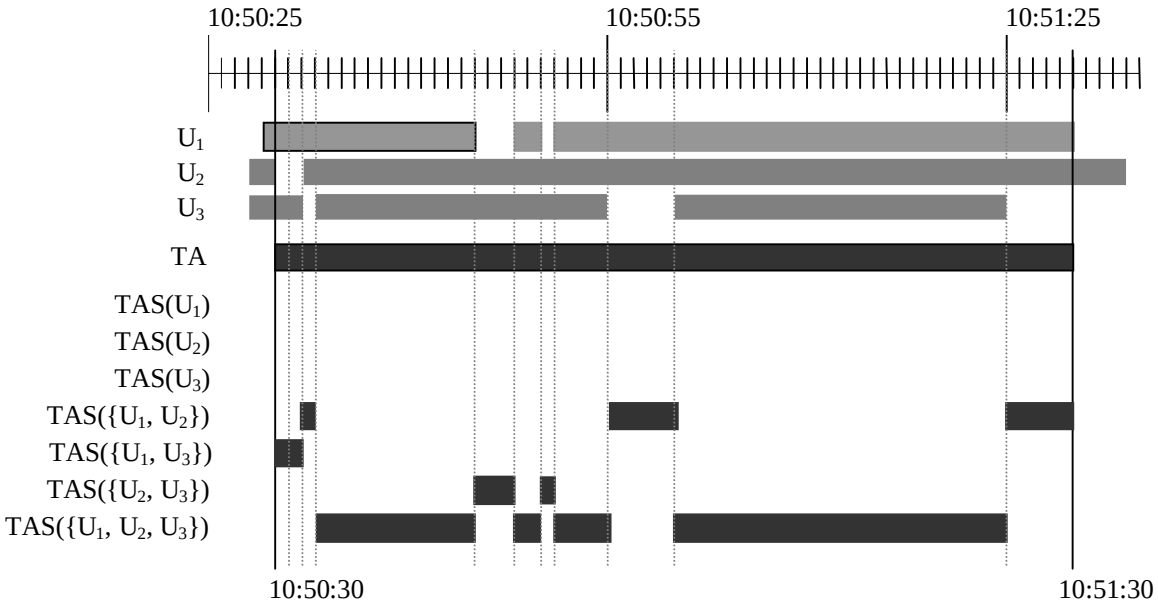

Figure 13. A graphical representation of the total activity for each subgroup (TAS(SG)) on an object by three users during the time period [10:50:30, 10:51:30].

Using the equation to determine the total activity for each subgroup of size *m* (TAS), we can determine the ***T**otal **A**ctivity for **A**ll **S**ubgroups (TAAS)* of size *m* spent manipulating the object during $[T_s, T_e]$. This is shown in Equation 11.

> Equation 11. Total activity by all subgroups *SG* of group *G* during the time period $[T_s, T_e]$ for an object *O*.

$$TAAS(O, choose(G, m), [T_s, T_e]) = \bigcup TAS(O, SG_i, [T_s, T_e]), \forall\ SG_i \in choose(G, m)$$

Using the example, we can compute the total time the users worked individually as:

$$\begin{aligned} TAAS(O, choose(G,1), [10{:}50{:}30, 10{:}51{:}30]) = {} & TAS(O, \{U_1\}, [10{:}50{:}30, 10{:}51{:}30]) \cup \\ & TAS(O, \{U_2\}, [10{:}50{:}30, 10{:}51{:}30]) \cup \\ & TAS(O, \{U_3\}, [10{:}50{:}30, 10{:}51{:}30]) \\ = {} & \varnothing \cup \varnothing \cup \varnothing \\ = {} & \varnothing. \end{aligned}$$

For brevity, the solutions for the TAAS for subgroups of size two and three are not worked here, but can be shown to be:

$$\begin{aligned} TAAS(O, choose(G,2), [10{:}50{:}30, 10{:}51{:}30]) = {} & \{[10{:}50{:}30, 10{:}50{:}33], \\ & [10{:}50{:}45, 10{:}50{:}48], \\ & [10{:}50{:}50, 10{:}50{:}51], \\ & [10{:}50{:}55, 10{:}51{:}00], \\ & [10{:}51{:}25, 10{:}51{:}30]\} \\ TAAS(O, choose(G,3), [10{:}50{:}30, 10{:}51{:}30]) = {} & \{[10{:}50{:}33, 10{:}50{:}45], \\ & [10{:}50{:}48, 10{:}50{:}50], \\ & [10{:}50{:}51, 10{:}50{:}55], \\ & [10{:}51{:}00, 10{:}51{:}25]\} \end{aligned}$$

Graphically, we can show this as in Figure 14.

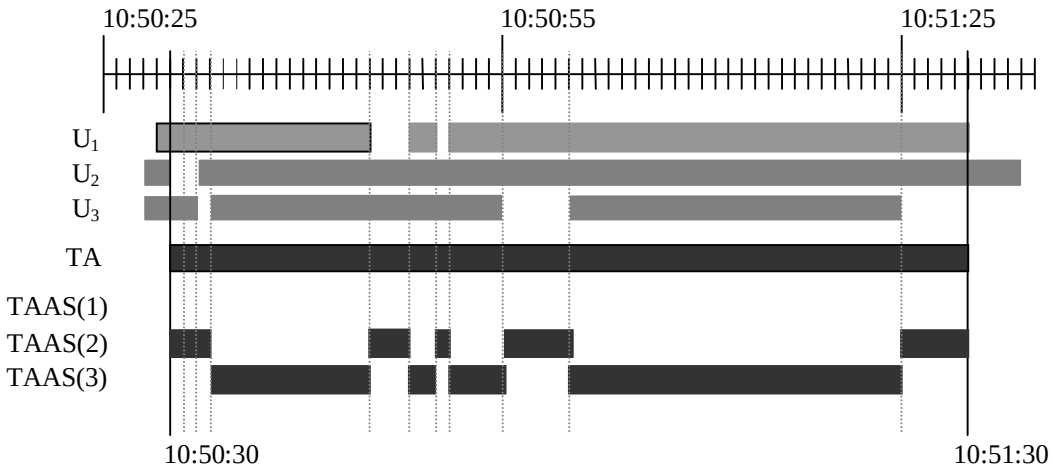

Figure 14. A graphical representation of the total activity for all subgroups of size *m* (TAAS(m)) on an object by three users during the time period [10:50:30, 10:51:30].

To compare the relative amounts of total activity time for the subgroups of size *m*, we can compute the TAAS as a percentage of the total activity, as shown in Equation 12. Note that for $n = |G|$, $P_1 + P_2 + \ldots P_n = 100\%$.

Equation 12. Percentage of time manipulating *O* as groups of size *m* during the time period $[T_s, T_e]$.

$$P_m(O, G, [T_s, T_e]) = \frac{seconds(TAAS(O, choose(G, m), [T_s, T_e]))}{seconds(TA(O, G, [T_s, T_e]))} * 100\%$$

Using the example, the percent of time the users were working independently is:

$P_1(O, \{U_1,U_2,U_3\}, [10:50:30, 10:51:30]) = 0 / 60 * 100\% = 0\%$.

The percent of time the users were working in pairs is:

$P_2(O, \{U_1,U_2,U_3\}, [10:50:30, 10:51:30]) = 17 / 60 * 100\% = 28.3\%$.

The percent of time the users were working all together in a group of three is:

$P_3(O, \{U_1,U_2,U_3\}, [10:50:30, 10:51:30]) = 43 / 60 * 100\% = 71.7\%$.

Finally, the measure of joint activity can be determined using Equation 13. If each user moves independently (i.e. there is no simultaneous manipulation of the object), *JA(O, G)* will be equal to 1 indicating that only one person was moving at any time. If all of the *n* of the users in the group are moving simultaneously during the total active time, *JA(O, G)* will equal *n*. Values for *JA(O, G)* in between *1* and *n* will indicate that the users were manipulating the object somewhere in between, perhaps they worked independently for some time, then later worked together, or not all of them could coordinate and work at precisely the same time.

Equation 13. Equation to calculate the measure of joint activity on an object *O* for a group *G*.

$$JA(O, G) = (1*P_1 + 2*P_2 + 3*P_3 + \ldots + n*P_n)/100\%$$

Using the example,

$$JA(O, \{U_1,U_2,U_3\}) = (1*0 + 2*28.3 + 3*71.7) / 100\% = 2.717.$$

This can be interpreted as the three users were working more simultaneously than independently to manipulate the object. In essence, this value states that, on average, approximately 2.7 users were active at any given time.

4.4.3 Determining the quality of program events

During the course of an activity, some users in a group might contribute a lot to the solution of the problem while others may only contribute a small portion. The expectation is that the more actively a user is participating, the more likely he or she will be focused on the given task. The value of the measure of joint activity (JA) for a task may not be expressive enough to determine how well the users worked together on the task. Thus we wish to define a method to measure the contribution of each user as a way of estimating how focused they were on the task.

One possible method to measure the amount a user *U* contributes would be to determine

the total number of seconds that the user was active on a object during [T_s, T_e]. However, depending on the activity, a user might have a high total activity, but in reality their contribution to the solution is very small. In the Color Matcher activity, it would be like a user pressing and holding the left mouse button (to start the move event) and not moving the mouse before releasing the mouse button. While the computer might detect that this is a mouse move, the user is really not contributing to the solution of the task. Therefore, it may be necessary to also measure the "quality" or *significance* of the users' participation in the activity. Admittedly, this measure might be subjective and is very dependent on how the user can manipulate the objects presented by the application.

This is best illustrated with an example. Assume the user is working with an application where LMD MM* LMU combines as a *Move* or a *Create* program event depending on the location of the LMD simple event. If the user presses the mouse button down while over an object, the LMD MM* LMU combination moves the object to the new location where the mouse button was released (where the LMU occurred). This is shown in Figure 15(a). If the user generates the LMD over a location where there is no object, the LMD MM* LMU combination will create a free-formed line following the locations of the cursor when a MM event occurs that will end at the location of the simple LMU event. This is shown in Figure 15(b). An application designer might decide that the situation in Figure 15(a) is less productive (defined in Section 3.3 as one that reduces the difference between the current state and the desired goal state) than Figure 15(b). To this designer, the significance of that user's participation in the activity while generating that Move program event is lower than the significance of the user who created the blob with the same sweep of the mouse.

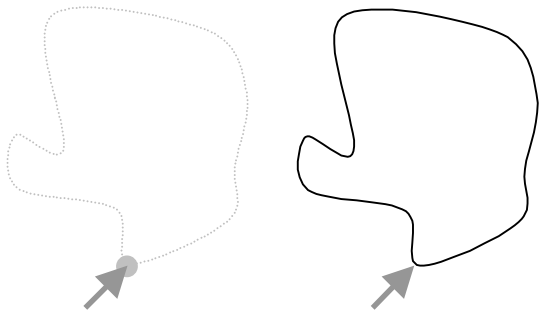

Figure 15. An example of the effects of a LMD MM* LMU combination when the location of the LMD and LMU are the same. (a) This combination generates a Move program event, and the object winds up in the same location on the screen. (b) This combination generates a Create program event and draws an object.

Of course, a user who presses their mouse down and never moves the cursor before releasing the mouse would do very little towards moving or creating an object! A designer is likely to think that the significance of this type of participation in the activity is low in either case.

We define a second metric, a *measure of significance*, as a way for a designer to describe the quality of the program events generated by a user during his or her participation in an activity. If the user generates a *significant* program event, then their participation in the activity may be high for that period of time, whereas a user who generates an insignificant program event may not be contributing much to the solution of the task. Depending on how a designer defines the measure of significance for a particular class of program event, a user who generates a trivial event may be taking a trivial step (as defined in Section 3.3) towards the solution of a given problem. Again, the measure of significance and what constitutes significant events are dependent on the design of activity. It is best to describe these concepts with the following example.

A *Move* program event, as previously defined, is made up from a sequence of LMD MM* LMU simple events. The LMD and LMU are very significant in generating the Move program event, through them the user is informing the system where and when to

start and stop the move event. Depending on the system, some number of MM events are generated at potentially uneven intervals by the computer during the time between the LMD and LMU events. Figure 16 shows an example of how a user moved a small circle from point P to Q. The user generated a LMD over the circle at point P, moved his or her mouse, which generated a series of MM events, then generated a LMU event at point Q.

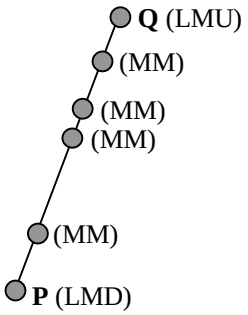

Figure 16. A Move program event comprised of a LMD, a series of MM events, then a LMU.

Some MM events may contribute significantly to the overall move program event, while others may not. What an application designer considers a significant contribution to a program event will be dependent on the type of activity being developed. A designer may decide if a program event is comprised of a high number of significant simple events for a given amount of time spent on the event, then the more *significant* the program event is. In other words, the insignificant simple events are ignored because the combination of the other events still produces the same result. Consider the example in Figure 17 of a MM event, shown as a darker line segment, that only displaces the point slightly relative to the overall displacement of the object. If this small MM displacement is a large percentage of the total positioning of the object (as shown in Figure 17(a)), then the MM is a significant simple event. If the MM of the object is a small percentage of the total movement, as shown in Figure 17(b) or (c), then it is considered to be an insignificant simple event.

The designer of this activity may chose to define that a MM event is insignificant in the context of a *Move* program event when the movement in the x and y direction is less than 2% of the total movement to that point. Thus, in the examples in Figure 17, the first MM

event, i_{a1}, i_{a2}, and i_{a3}, are significant simple events since they contribute a large portion (100%) of the movement thus far. In case (c), however, the MM event i_{c12} is small in comparison to the movement thus far. The designer might consider this an insignificant simple event.

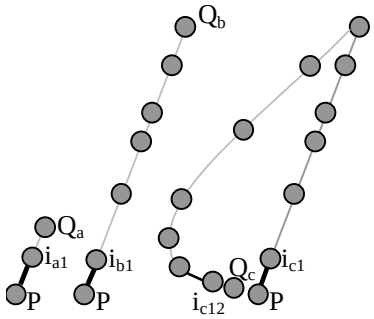

Figure 17. Three example *Move* program events from point P to Q. (a) The MM from P to i_{a1} is actually a large portion of the total distance, but the total distance is small. (b) The MM from P to i_{b1} is actually a small portion of the total distance, but the total distance is significant. (c) A similar situation to (b), but the starting and ending locations are very close.

In this case, a formal description of the significance of a program event *E* could be the average time per significant simple event. This is determined by a function of the number of seconds in the event span $[T_s, T_e]$ (or $seconds(ES) = seconds([T_s, T_e])$) and a count of the number of significant simple events *N*, shown in Equation 14. This equation will give a higher value the more significant the event is.

Equation 14. Function to calculate the significance of a program event as the average time per significant simple event.

$$significance(ES) = \frac{Seconds(ES)}{N}$$

Note that essentially instantaneous program events (describe in Section 4.4 as those with event spans of length 0) do not, by definition, contain any insignificant simple events. Selecting an object is an example of an essentially instantaneous program event that is comprised of a LMD LMU pair, each of which contributes significantly to the program

event. Thus N will never be 0.

An alternate definition of a significant *Move* program event might be one in which the total movement is a large distance from the starting point. In this case, the designer may wish to define the significance as in Equation 15.

Equation 15. Function to calculate the significance of a program event based on the location of the LMU simple event.

$$significance(ES) = \begin{cases} 0, \text{ if location of the LMU event is within 10 pixels of the LMD event} \\ 1, \text{ otherwise} \end{cases}$$

Similarly, it could be related back to the concept of the productivity of a problem-solving step, as described in Section 3.3, using the function shown in Equation 16.

Equation 16. Function to calculate the significance of a program event based on the productivity measure.

$$significance(ES) = \begin{cases} 1, \text{ if productivity} > 0 \\ 0, \text{ if productivity} = 0 \\ -1, \text{ otherwise} \end{cases}$$

As stated earlier in this section, the measure of joint activity (JA) may not be expressive enough to determine the quality of the interaction between the users. The significance computes this quality by estimating how focused each user is while interacting with their portion of the task. The significance can be used as a weighting factor in the calculation of the JA. The next section will present this new weighted JA (WJA) in greater detail. However, we will mention here that it is convenient to normalize the significance to the values between 0 and 1 for use in the WJA.

Significance values can also be displayed graphically, as in Figure 18, by using brightness to indicate the significance of an event. In this figure, the brighter the event

span representation, the more significant it is, the darker the event span, the less significant it is. In order to illustrate the example, we will assign each of the event spans listed in Section 4.4.1 a normalized significance rating as listed below using the modified notation.

User 1: significance([10:50:29, 10:50:45]) = 0.00
significance([10:50:48, 10:50:50]) = 1.00
significance([10:50:51, 10:51:30]) = 0.75
User 2: significance([10:50:28, 10:50:30]) = 0.25
significance([10:50:32, 10:51:34]) = 0.50
User 3: significance([10:50:28, 10:50:32]) = 0.25
significance([10:50:33, 10:50:55]) = 1.00
significance([10:51:00, 10:51:25]) = 0.50

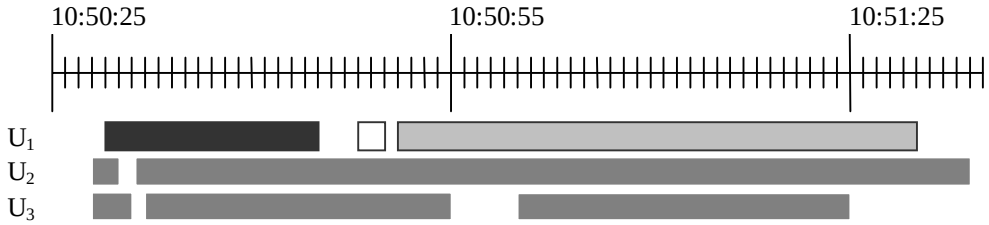

Figure 18. A graphical representation of the event spans and their significance. The lighter the event span, the more significant it is.

Since multiple users might be manipulating an object during a given time period, we will modify our notation slightly to distinguish the significance for each user's interaction. We define *significance($[T_a, T_b]$, U_i)* to be the measure of significance for the event span generated by user U_i that contains the interval $[T_a, T_b]$. For simplicity, we will assume that the measure of significance is constant during an event span ES = $[T_s, T_e]$. Thus, for any interval $[T_a, T_b] \subseteq ES$, *significance($[T_a, T_b]$) = significance(ES)*. In the above example:

significance([10:51:00, 10:51:25], U_1) = 0.75
significance([10:51:00, 10:51:25], U_2) = 0.50
significance([10:51:00, 10:51:25], U_3) = 0.50

4.4.4 A measure of joint activity, revisited

The measure of joint activity (JA) is a way for a designer or an experimenter to determine

the extent to which the users were able to work together. However, as stated in the previous section, although the users may appear to be working simultaneously together, the relative quality or significance of each user's influence on the program objects may be different. In particular, depending on the application, some of the users may do proportionally more or less work towards reaching the goal of the activity. Thus we wish to weight the original JA by the significance of each user's interaction.

Recall that the JA is computed as a weighted sum of the percent of total active time that the users worked independently, in pairs, in groups of three, etc. Formally this was defined in Equation 13 as $JA(O, G) = (1*P_1 + 2*P_2 + 3*P_3 + \ldots + n*P_n) / 100\%$. Each P_m is defined in Equation 12 as the percentage of total activity (TA(O, [T_s, T_e])) that the users in all subgroups of size *m* manipulated the object (TAAS(O, choose(G, m), [T_s, T_e])). However, during each interval $[T_a, T_b] \in TAAS(O, choose(G, m), [T_s, T_e])$, the significance of some users interactions may be more than others. Our goal is to weight the amount of time in each interval of a TAAS to adjust JA for the differences in significance. We will call this the *weighted TAAS*, or the *ws-TAAS*.

Essentially, we want to weigh each interval [T_a, T_b] by some combination of the significance values for each user's interaction with the object. A straightforward function to calculate the significance weight for an interval [T_a, T_b] is to average the significance values for each user during that interval. This is shown in Equation 17.

Equation 17. Calculating the significance weight of an interval.

$$weight([T_a, T_b], SG) = \frac{\sum \ significance\ ([T_a, T_b], U_i)}{m} \ , \forall\ U_i \in SG \text{ and } m = |SG|$$

We can now define the *weighted-seconds* of an interval $[T_a, T_b]$ as number of seconds in that interval, multiplied by the weight calculated from the significance values, as shown in Equation 18.

Equation 18. The weighted-seconds function to calculate the number of seconds in an interval, weighted based on the average significance value.

$$weighted\text{-}seconds([T_a, T_b], SG) = weight([T_a, T_b], SG) * seconds([T_a, T_b])$$

To illustrate this next point, we will reexamine TAAS intervals shown in Figure 14 using the significance values for each event span given in the previous section. Figure 19 shows the original event spans labeled by their significance values, and the TAAS(O, choose(G, m), $[T_s, T_e]$) intervals for m = 1, 2, and 3. We will focus on the intervals marked by A, B, C, and D.

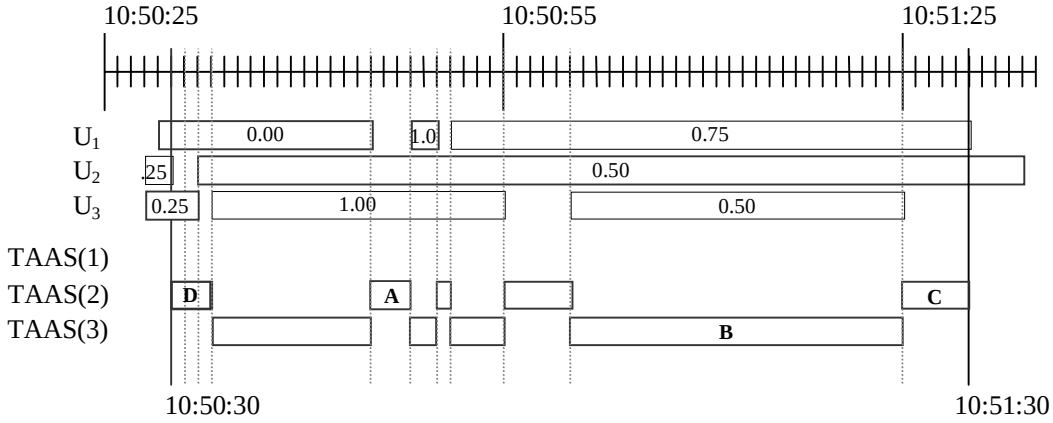

Figure 19. A graphical representation event spans marked with their significance values, the TAAS intervals for subgroups of size m = 1, 2, and 3 (TAAS(m)). The intervals A, B, C, and D will be examined in the text.

Interval A corresponds to [10:50:45, 10:50:48]. As shown in the figure, the significance values for this interval for U_2 and U_3 are 0.50 and 1.00, respectively. Thus,

weight([10:50:45, 10:50:48], {U_2, U_3}) =
(significance([10:50:45, 10:50:48], U_2) + significance([10:50:45, 10:50:48], U_3)) / 2 =
0.50 + 1.00 / 2 =
0.75.

weighted-seconds([10:50:45, 10:50:48], {U_2, U_3}) =
weight([10:50:45, 10:50:48], {U_2, U_3}) * seconds([10:50:45, 10:50:48]) =
0.75 * 3 =
2.25.

Similarly for interval B, which corresponds to [10:51:00, 10:51:25], the significance values for U_1, U_2 and U_3 are 0.75, 0.50 and 0.50, respectively. Thus,

weight([10:51:00, 10:51:25], {U_1, U_2, U_3}) =
(significance([10:51:00, 10:51:25], U_1) +
  significance([10:51:00, 10:51:25], U_2) +
  significance([10:51:00, 10:51:25], U_3)) / 3 =
0.75 + 0.50 + 0.50 / 3 =
0.583.

weighted-seconds([10:51:00, 10:51:25], {U_1, U_2, U_3}) =
weight([10:51:00, 10:51:25], {U_1, U_2, U_3}) * seconds([10:51:00, 10:51:25]) =
0.583* 25 =
15.575.

Finally for interval C, which corresponds to [10:51:25, 10:51:30], the significance values for U_1 and U_2 are 0.75 and 0.50, respectively. Thus,

weight([10:51:25, 10:51:30], {U_2, U_3}) =
(significance([10:51:25, 10:51:30], U_1) + significance([10:51:25, 10:51:30], U_2)) / 2 =
0.75 + 0.50 / 2 =
0.625.

weighted-seconds([10:51:25, 10:51:30], {U_2, U_3}) =
weight([10:51:25, 10:51:30], {U_2, U_3}) * seconds([10:51:25, 10:51:30]) =
0.625 * 5 =
3.125.

However, this method will not work for interval D, which corresponds to [10:50:30, 10:50:33]. Recall that interval D is actually comprised the two sub intervals, [10:50:30, 10:50:32] and [10:50:32, 10:50:33], with different weights:

[10:50:30, 10:50:32] ∈ TAS(O, {U_1, U_3}, [10:50:30, 10:51:30])
significance([10:50:30, 10:50:32], U_1) = 0.00
significance([10:50:30, 10:50:32], U_3) = 0.25
weight([10:50:30, 10:50:32], {U_1, U_3}) = 0.00 + 0.25 / 2 = **0.125**
weighted-seconds([10:50:30, 10:50:32], {U_1, U_3}) = 0.125 * 2 = 0.25.

[10:50:32, 10:50:33] ∈ TAS(O, {U_1, U_2}, [10:50:30, 10:51:30])
significance([10:50:32, 10:50:33], U_1) = 0.00
significance([10:50:32, 10:50:33], U_2) = 0.50
weight([10:50:32, 10:50:33], {U_1, U_2}) = 0.00 + 0.50 / 2 = **0.25**
weighted-seconds([10:50:32, 10:50:33], {U_1, U_2}) = 0.25 * 1 = 0.25.

It becomes apparent from this example that the weights may not be the same if an interval $[T_a, T_b] \in TAAS(O, choose(G, m), [T_s, T_e])$ is actually comprised of two or more subintervals. Thus we must define the *weighted TAAS*, or *ws-TAAS*, based on the intervals in the total activities by subgroups of size *m* (the TAS as defined in Equation 11) and not the TAAS, as in Equation 19. We also note that, as with determining the number of seconds in a union of disjoint intervals (Equation 7), the *weighted-seconds* of a union of disjoint intervals is the sum of the weighted-seconds of each interval, as shown in Equation 20.

Equation 19. Definition of weighted seconds of total activity by all subgroups *SG* of group *G* during $[T_s, T_e]$ for an object *O*.

$$ws\text{-}TAAS(O, choose(G, m), [T_s, T_e]) = \sum weighted\text{-}seconds(TAS(O, SG_i, [T_s, T_e])), \quad \forall\ SG_i \in choose(G, m)$$

Equation 20. Calculation of the weighted number of seconds of the union of disjoint intervals.

$$weighted\text{-}seconds\left(\bigcup_{i=1}^{n} I_i\right) = \sum_{i=1}^{n} weighted\text{-}seconds(I_i)$$

In the given example we can compute the weighted seconds of each interval in each *TAS(O, SG, [10:50:30, 10:51:30])*. For individual users this is trivial since there we no

intervals of time when just one user was manipulating the object. For pair and all three users, the calculations are as follows:

ws-TAAS(O, choose(G, 2), [10:50:30, 10:51:30]) =
weighted-seconds(TAS(O, {U_1, U_2}, [10:50:30, 10:51:30])) +
weighted-seconds(TAS(O, {U_2, U_3}, [10:50:30, 10:51:30])) +
weighted-seconds(TAS(O, {U_1, U_3}, [10:50:30, 10:51:30])) =
weighted-seconds([10:50:32, 10:51:33] ∪
[10:50:55, 10:51:00] ∪
[10:51:25, 10:51:30]) +
weighted-seconds([10:50:45, 10:50:48] ∪ [10:50:50, 10:50:51]) +
weighted-seconds([10:50:30, 10:50:32]) =
(0.25 * 1 + 0.625 * 5 + 0.625 * 5) + (0.75 * 3 + 0.75 * 1) + 0.125 * 2 =
6.5 + 3.0 + 0.25 =
9.75.

and by all three users simultaneously:

ws-TAAS(O, choose(G, 3), [10:50:30, 10:51:30]) =
weighted-seconds(TAS(O, {U_1, U_2, U_3}, [10:50:30, 10:51:30]) =
weighted-seconds([10:50:33, 10:50:45] ∪
[10:50:48, 10:50:50] ∪
[10:50:51, 10:50:55] ∪
[10:51:00, 10:51:25]) =
0.5 * 12 + 0.83 * 2 + 0.75 * 4 + 0.583 * 25 = 25.435

We can simply define the weighted percentage of time manipulating an object *O* by groups of size *m*, WP_m, as shown in Equation 21. Note that if the weights for computing the weighted percentages are between 0 and 1, for $n = |G|$, $P_1 + P_2 + \ldots P_n \leq 100\%$.

Equation 21. Weighted percentage (WP) of time manipulating *O* as groups of size *m* during $[T_s, T_e]$.

$$WP_m(O, G, [T_s, T_e]) = \frac{ws\text{-}TAAS(O, choose(G, m), [T_s, T_e]))}{seconds(TA(O, G, [T_s, T_e]))} * 100\%$$

Thus, the weighted percentage of time the users worked individually, in pairs or in groups of three are as follows:

$WP_1(O, \{U_1,U_2,U_3\}, [10:50:30, 10:51:30]) = 0 / 60 * 100\% = 0\%.$
$WP_2(O, \{U_1,U_2,U_3\}, [10:50:30, 10:51:30]) = 9.75 / 60 * 100\% = 16.3\%.$
$WP_3(O, \{U_1,U_2,U_3\}, [10:50:30, 10:51:30]) = 25.435 / 60 * 100\% = 42.4\%.$

Finally, the new weighted measure of joint activity, *WJA(0, G)*, can be written as in Equation 22.

Equation 22. Function to calculate the weighted joint activity, WJA.

$$WJA(O, G) = (1*WP_1 + 2*WP_2 + 3*WP_3 + \ldots + n*WP_n) / 100\%.$$

Using the example, we find that their interaction was really much more like the users were working independently than in a group of three users simultaneously:

$$WJA(O, G) = (1 * 0 + 2 * 16.3 + 3 * 25.435) / 100\% = 1.089$$

4.5 Focusing the users on a specific task

As stated in Section 2.1, a common method for classifying cooperative software is based on the physical location of the users and how synchronously they can work together. There are certain instances when having tight synchronous interactions focused on one object may be preferable, either to keep users focused on a task (in the case of middle high school students) or for entertainment purposes. We have chosen two methods that may aid in keeping the users focused. The first involves having the program enforce how simultaneous the users interact, which will be discussed in the next subsection. The second method, discussed in Section 4.5.2, is to allow the users to manipulate object through a more complicated control system using objects we call Cooperatively Controlled Objects.

4.5.1 Specifying the types of interactions

Developers may 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.

Recall that 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 do not expect an immediate response from questions or requests. A *required asynchronous cooperative interaction* or a *serial interaction* is one in which each user must have access to the other user's completed message before proceeding. An example of this type of interaction is a sequential train of thought developed by a group of two users, U_1 and U_2, through email or a knowledgebase. The events generated by a user include research (the time when the user is doing discovering information to include in the message, which may or may not include computer time), compose (the time spent writing the message), send, receive, and read. In the example that follows, $ES(E, [T_{is}, T_{ie}], U_j)$ is an event E that occurs during the span of time from T_{is} to T_{ie} by user U_j.

ES(research, $[T_{1s}, T_{1e}]$, U_1): U_1 does research for the message.
ES(research, $[T_{2s}, T_{2e}]$, U_1): U_1 types in the message.
ES(send, $[T_{3s}, T_{3e}]$, U_1): U_1 sends the message to U_2
ES(receive, $[T_{4s}, T_{4e}]$, U_2): U_2 receives the message
ES(read, $[T_{5s}, T_{5e}]$, U_2): U_2 reads the message
ES(research, $[T_{6s}, T_{6e}]$, U_2): U_2 researches a response to the message
ES(compose, $[T_{7s}, T_{7e}]$, U_2): U_2 types the response
ES(send, $[T_{8s}, T_{8e}]$, U_2): U_2 sends the response
ES(receive, $[T_{9s}, T_{9e}]$, U_1): U_1 receives the response
ES(receive, $[T_{10s}, T_{10e}]$, U_1): U_1 reads the response.

An alternative definition for a require asynchronous interaction is that each event span must not overlap with one another, or $\cap ES(E, [T_{is}, T_{ie}], U_j) = \phi$, $\forall$ event spans generated by each user j. (Note that for this definition we assume that any one user does not generate overlapping event spans.)

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 etalk are examples of a

synchronous cooperative interaction. Often, the users in an etalk 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. An etalk writer is not guaranteed that the other user(s) are focused on the current thread of conversation. Many users will switch between applications or tasks while a writer is completing a thought.

We have further broken up this temporal classification based on how simultaneously the users may interact with each other. We define a *simultaneous cooperative interaction* or *tightly coupled interaction* between two or more users (U_1, U_2…, U_n) as an interaction where the intersection of event spans generated on an object *O* those users is non-empty. In the terminology defined in Section 0, $TAAS(O, choose(\{U_1, U_2 \ldots, U_n\}, m), [T_s, T_e]) \neq \varnothing$, where $m > 1$.

A *required* simultaneous interaction, or a *coactive* interaction, is one in which the users must manipulate an object at the same time in order to make any progress on the activity. This type of interaction could be used in a multi-player first person perspective game such as Doom, Quake or Diablo. In an immersive world, U_1 approaches a door that contains a large vault-style handle, such as in Figure 20. The round latch mechanism permits the handle on the door to move. But the latch is extremely heavy and U_1 cannot move it alone. U_2 arrives and helps, and they can open the door cooperatively.

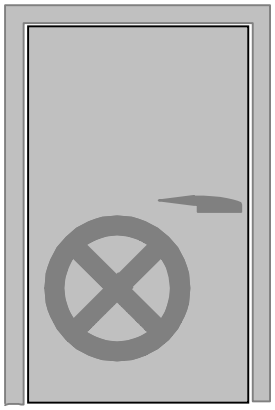

Figure 20. A vault style door lock.

An *encouraged* simultaneous interaction is one in which the users do not have to be

manipulating 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. In the previous example, the latch on the door is very heavy but U_1 can start to open it alone. When U_2 arrives and helps, the latch becomes much "easier" to move and they open the door quickly.

We further describe asynchronous interactions as two other forms of "simultaneous" interactions. As with a 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 the event spans on the object overlap. In our previous example, if U_2 begins to help U_1 open the door, the latch will become harder, but not impossible, to move. Finally, we can also describe a required *asynchronous* interaction as a *disallowed* simultaneous interaction. Here, the software or system physically prevents the users' event spans from overlapping. The software literally prevents the users from making progress on their task if they work together. In our example, if U_1 is already moving the door latch when U_2 arrives, then any of U_2's attempts to assist will have no effect. The latch on the door is an exclusive object, only U_1 or U_2 can manipulate the latch at any given time.

4.5.2 Cooperatively Controlled Objects (CCOs)

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.

4.5.2.1 CCOs defined

In object-oriented terminology, an object contains state and has behavior. We define a *controlled object* to be an object containing methods which allow a one or more users to manipulate properties of that object through input devices such as mice, joysticks,

keyboards, etc. A *cooperatively controlled object* (CCO) is a controlled object 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." 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 with inputs of the current values of the property (and perhaps other properties of other objects) and the input-device states corresponding to the multiple users simultaneously manipulating the object. The output of the function determines the value of the property. By providing one such function for each of the CCO's modifiable properties, a complete control mechanism for the CCO can be specified.

Fine-grained sharing is one method for implementing a CCO that has been previously studied [86, 93]. An object is expressed in terms of its parts (or properties) where a different user controls each property. 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 simply check in and check out small pieces of a "shared" document do not necessarily coordinate in real time. They can work asynchronously and they are not forced to synchronize their activities in the same way 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-grain sharing.

Figure 21 depicts an example in which a point is controlled by decomposing it into its x and y coordinates and each coordinate is modified by separate users. A cooperatively controlled point, on the other hand, may involve a more general functional relationship between user inputs and controlled properties. For example, the controlled point could be

determined by the intersection of two lines and controlled by four users, each user directly manipulating the location of one of two points on a line with a mouse.

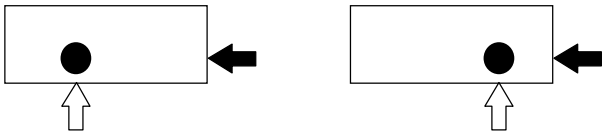

Figure 21. Controlling the location of a point using a fine-grained sharing technique.

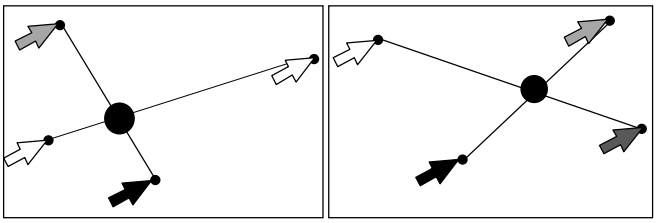

Figure 22. Cooperatively controlling the location of a point.

Table 5 lists some examples of cooperatively controlled objects. Geometrical objects, such as points, lines, and polygons, can of course be controlled using fine-grained sharing at the level of coordinate values, vertices, etc. They can also be controlled cooperatively via geometric or other relationships between input values and the controlled properties. More complicated spatial objects, such as fractals and bitmapped images, can also be made into CCOs by allowing users to jointly manipulate their properties. More abstract mathematical objects, such as transformations on the plane or transformations of images, may be cooperatively controlled. Yet another example of cooperative control can be found in the Curve Fitter program described in [12], 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, and their characteristics relevant to cooperation, communication and effective control must be examined.

Table 5. A list of cooperatively controlled objects.

Objects	**Example**
Point	x, y r, theta Midpoint of a line Intersection of two lines Intersection of the bisectors of the angles of a triangle The centroid of a triangle
Line	x1, y1, x2, y2 Endpoints of line Endpoint and midpoint of line Intersection of two three dimensional planes
Triangle right triangle equilateral triangle	Two angles totaling less than 180 degrees and one side Three points Three intersecting lines
Quadrilateral rhombus square rectangle	Four points Four angles totaling less than 360 degrees and one side Four intersecting lines …
Fractal curves and plants	An axiom, production rules and the number of cycles[8] Points on line segments to control length and branching factor
Color	Red, green and blue color values Hue, saturation, value parameters
Curve	Coefficients of the polynomial describing the curve n+1 control points, where n = degree of polynomial of curve
Musical chord	Three notes of a chord
Image warping transformation	Number of image warping control lines and locations and lines Weights of image warping control lines

4.5.2.2 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 knowlegebase). 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)*.

The degree to which an object is cooperatively controlled depends on two factors, the simultaneity of interaction and the how complex is the method for manipulating the

objects. If we graph these two factors on orthogonal axis, such as in Figure 23, 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 [30, 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.

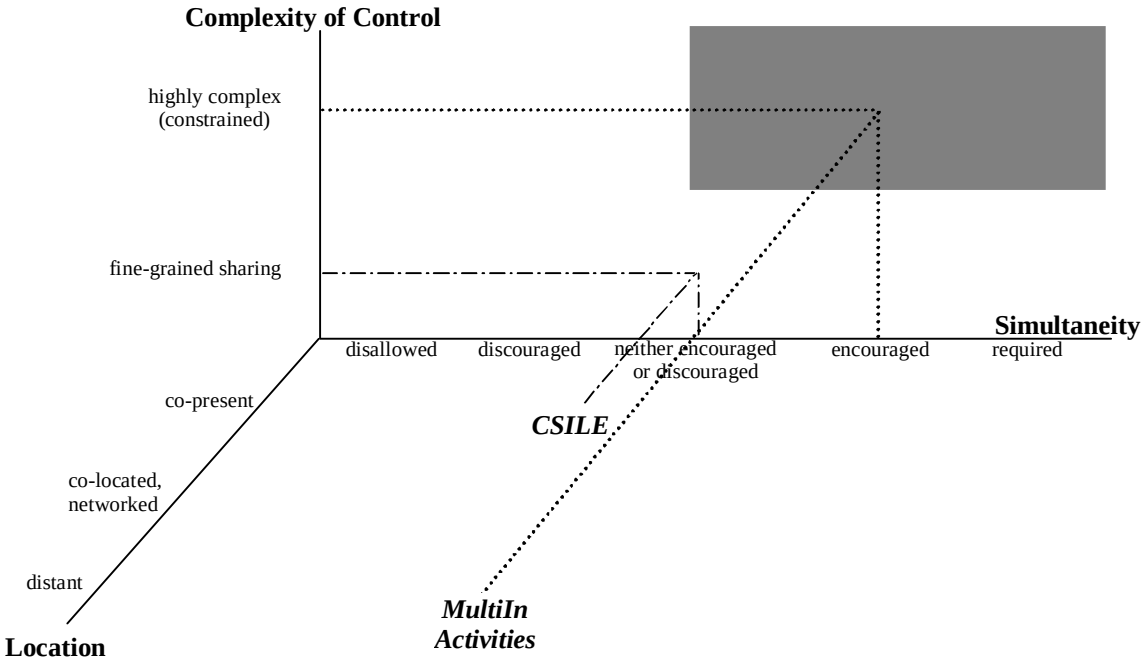

Figure 23. A visualization of three characteristics of systems that use cooperatively controlled objects.

We may, however, wish to describe the DOCC in a more formal or numeric way, such as a pair *(S, COC)* where *S* is a *description of the simultaneity* and the *COC* is the *complexity of control*. The values for *S* and *COC* correspond to the locations on the axes of the graph above. Thus, an analysis of an existing application may find that the value of *S* corresponds to a certain location of the graph, helping to classify the application. For

example, if CSILE were to be analyzed, we would expect that the value of *S* component of the DOCC would translate into the category of "Neither encouraged or discouraged." This analysis could also be used during the design phase of a cooperative activity. If a designer specifically wanted the interactions between the users to be highly simultaneous and complex, he or she might want the objects in the application to fall in the upper right hand quadrant of the graph. These objects could be designed similar to other controls that had estimated values of *S* and *COC* that fall in the same quadrant. Now, admittedly, what is designed and how people use it in reality may be two very different things. The designer could, however, design the application in an interative manner that includes usability testing that allows the designer to evaluate how true his estimates are.

The *description of simultaneity, S*, is essentially a normalized estimate of the measure of joint activity, *JA*, discussed in Section 0. Recall that the value of *JA* for an interaction indicates how simultaneous users manipulated an object during an activity. For a group *G* of *n* users $\{U_1, U_2, \ldots U_n\}$ charged with a task, the measure produces a value from *1* to *n* where a lower value indicates more independent control and a higher value indicates more simultaneous control. If we normalize the value of the expected or average *JA* to be between 0 and 1 using the formula shown in Equation 23, we could imagine this corresponding to the types of interactions discussed in Section 4.5.1. This relationship is shown in Table 6.

Equation 23. Function to normalize the measure of joint activity.

$$Normalize(JA(O, \{U_1, U_2, \ldots U_n\})) = \begin{cases} \frac{JA\,(O, \{U_1, U_2, \ldots U_n\}) - 1}{n - 1,\ for\ n > 1} & ,\ if\ \ n > 1 \\ 0\ otherwise \end{cases}$$

Table 6. Values for the description of simultaneity, *S*, and how they relate to the type of simultaneous interaction enforced by the activity.

Value of *S*	**Type of simultaneous interaction**
0	Disallowed
.25	Discourage
.50	Neither encouraged or discouraged
.75	Encouraged
1.00	Required

The value of *Complexity of Control (COC)* is determined based on the number of constraints used to control the object. A *Constraint* is a specified relation that should be maintained by the system. Constraints are often used as tool for building user interfaces [35]. For example, A = B + C is a constraint. A change to the value of A can change the values of B and/or C. Similarly if the value of B or C changes, the value of A will change. If we wish to specify that the is a one-way constraint, that is, the value of A will change if the value of B or C changes, but not vice-versa, we can denote this as A := B + C or A ← B + C. One-way constraints are used in systems such as most spreadsheets, EVAL/Vite [49], and Rendevous [86], and have the benefits that they are easy to understand and implement, and can be solved very quickly. Constraint graphs also can be used to depict relationships, as shown in Figure 24.

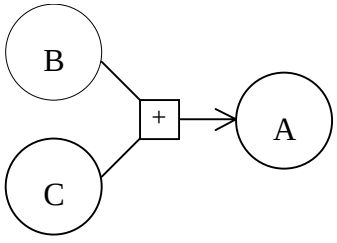

Figure 24. Graphical depiction of a constraint.

Recall (from Section 4.2.1) that an object *O* contains a set of changeable properties P = $\{p_1, p_2, p_3, \ldots p_n\}$, and may also contain a set of methods or functions, where the output of a method may set one or more of the properties of the object. These functions (or a composition of these functions) may control one or more properties of the object. We

will define a CCO as an object where at least one of the properties is cooperatively controlled.

The functions that cooperatively controlled a property may be manipulated through a series of constraints. In most mouse-based interactions, the user manipulates the location and states of the buttons on the device, which in turn manipulates the properties of cursor on the screen, which in turn manipulates an object. In a highly complex interaction, the changes to the properties of one object change the value of a property of another object, and so on.

For a user's cursor *UC* and objects O_1, O_2, … O_m that each contain some number of properties p_a, p_b, p_c, … p_m, a general relation that could hold might be as follows:

$$O_n.p_n \ldots O_2.p_b := O_1.p_b := UC.p_a$$

Simply directly manipulating the location of an object to follow the location of the user's cursor would be written as:

$$point.location := U_1C.location$$

In the previous case the location is taken to be one property, an (x, y) pair, rather than listing it out as separate entities. This is not true in our second case of fine-grained sharing an object is a form of cooperative with a constraint sequence of two steps. In the example of two users manipulating the *x* and *y* properties of a point by fine-grain sharing it as shown in Figure 21, the constraint sequence for this interaction is:

$$point.x := U_1C.x$$
$$point.y := U_2C.y$$

However, if a point were defined as the midpoint of a line, then the constraint sequence would be:

midpoint.location := (line.endpoint$_1$.location + line.endpoint$_2$.location) / 2, and
line.endpoint$_1$.location := U_1C.location
line.endpoint$_2$.location := U_2C.location

If a point were defined as the intersection of two lines (as in Figure 22), the constraint sequence would be:

point.y := point.x * line$_1$.slope + line$_1$.y-intersect
point.x := (line$_2$.y-itersect - line$_1$.y-intersect) / (line$_1$.slope - line$_2$.slope),
if line$_1$.slope != line$_2$.slope, or undefined if line$_1$.slope = line$_2$.slope
line$_1$.y-intersect := line$_1$.endpoint$_1$.y – line$_1$.slope * line$_1$.endpoint$_1$.x
line$_2$.y-intersect := line$_2$.endpoint$_1$.y - line$_2$.slope * line$_2$.endpoint$_1$.x
line$_1$.slope := (line$_1$.endpoint$_1$.y - line$_1$.endpoint$_2$.y) /
(line$_1$.endpoint$_1$.x - line$_1$.endpoint$_2$.x),
if line$_1$.endpoint$_1$.x != line$_1$.endpoint$_2$.x
line$_2$.slope := (line$_2$.endpoint$_1$.y – line$_2$.endpoint$_2$.y) /
(line$_2$.endpoint$_1$.x – line$_2$.endpoint$_2$.x),
if line$_2$.endpoint$_1$.x != line$_2$.endpoint$_2$.x
line$_1$.endpoint$_1$.location := U_1C.location
line$_1$.endpoint$_2$.location := U_2C.location
line$_2$.endpoint$_1$.location := U_3C.location
line$_2$.endpoint$_2$.location := U_4C.location

These examples have been presented in increasing order of complexity of the constraints. We define the measure of the *COC* to be the number of constraint non-circular relations required to manipulate the property of object. This measure is relative to a particular set of primitive relationships. A directly manipulated object would have a *COC* of 1, a fine-grained shared object would have a *COC* of 2, and a more complex constraint would have a higher *COC* still!

4.6 Summary

This chapter presented a model of multiple users' interactions with a program. After discussing why general models of computer programming are not sufficient to represent complex collaborative applications, we presented an alternate model designed to measure the cooperation between a group of users. We presented three metrics designed to

measure the users' participation in an activity. The first metric, the measure of joint activity (JA), summarizes the relative amount of time the users work individually, as subgroups, or as the whole group simultaneously. The second metric, the significance, is a method to determine the quality of each user's contribution to the task. Noting that the measure of joint activity does not take into account the significance of the users' interactions, the third metric, the weighted joint activity (WJA), was presented as by combining the first two measures.

This chapter also presented two techniques that can be employed in a computer system that are designed to focus the users on a task. The first technique is to specify how simultaneously the users are forced to work together on an object. The second technique is to allow the users to manipulate the program objects through more complicated control system. Finally, we presented and defined objects we call Cooperatively Controlled Objects, or CCOs. This definition included a way to measure the degree to which an object is cooperatively controlled in a program.

The next chapter will present a model of how the users' interact with each other. This model of the human-human interaction will allow us to measure or predict the amount of communication that is occurring between the users.

CHAPTER 5: MODELS OF HUMAN-HUMAN INTERACTIONS

5.1 Models of user activity

In his book *User Centered System Design* [77], Donald Norman describes a model of seven stages of user activity, shown in Figure 25. These stages include: establishing the goal, forming the intention, specifying the action sequence, executing the action, perceiving the system state, interpreting the state, and evaluating the system state with respect to the Goals and Intentions. (p. 41) Here, he uses the term *Goals* to describe the "state the user wishes to achieve" and *Intentions* as "the decision to act so as to achieve the state." (p. 37). This model has been used as a basis for the development of other models, such as the GOMS[1].

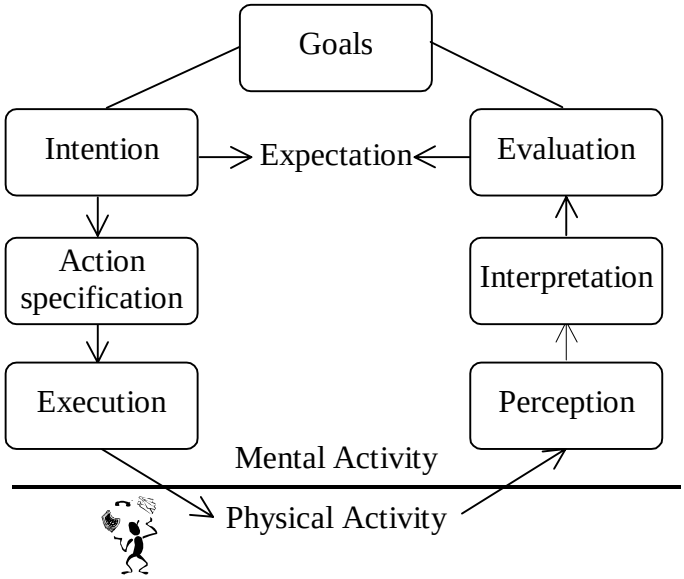

Figure 25. Donald Norman's *Seven stages of user activities*.

Norman's model does not implicitly include what we term the user's *internal context*. Cahour and Salembier define *context* as "the set of transitory representations built from the characteristics of the situation and permanent knowledge for the purpose of

interpreting communicative act (sic) and anticipating the understanding of others."[15, p. 286] Our definition of internal context is a state of the user's mind based on perceptions of the world around him or her, and includes what Norman calls the user's *goals* and *intentions*. As with Cahour and Salembier, it also encompasses a "memory" of the past and current states of the system or world, thoughts or opinions about other users, their abilities or roles in an activity and how to communicate with them, and their own views on the state of the activity. Thus, we have modified Norman's pictorial description of his model to include the user's internal context, shown in Figure 26. Some of his stages, such as perception and establishing the goal, modify a user's internal context, while other stages, such as evaluation, intention and action specification, use the internal context to form the next step. Interpretation both modifies and depends on the internal context; it requires a context to form an interpretation and the interpretation will perhaps change the user's view of the external world.

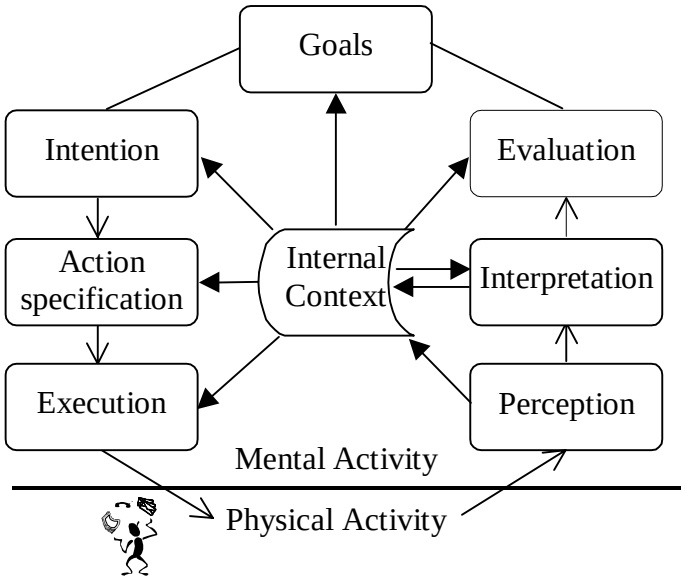

Figure 26. Norman's model modified to include the influences on and from the user's internal context.

We are most concerned with two parts of this model: perception and execution. In particular we are interested in what happens when a user executes an action that modifies

the computer system, communicates with another user, or does something to change to the how the users wish to solve the given problem. A user will modify the computer system through attached input devices and, depending on the application, these input devices would create events to modify the state of the application. Similarly the users will perceive changes to the system and incorporate them into their own internal context.

The Belief, Desire and Intention (BDI) model is a similar model commonly used in the design of intelligent agents [1]. Here an agent's cognitive representation of the state of the world, or "beliefs," are used to form plans on how to accomplish a task. These beliefs, coupled with the agent's desires and commitment to accomplish something, create an intention to do an action. The changes that are perceived as a result of the action become part of the agent's beliefs. We explore this model in more detail in the next section with respect to a model of conversation.

5.2 Models of communication

Interactions between the users, in particular communication and discourse are more difficult to model than a single user. Humans are by nature complex and unpredictable, unlike computers, which follow sets of predefined instructions. The models above are qualitative models of how a single person or agent might react to input. Communication between two or more people is that much more difficult to model because of the added complexities of the interactions between them. As Clark and Shaefer point out, "Contributing to discourse…appears to require more than just uttering the right words at the right time. It seems to consist of collective acts performed by the participants working together" [17, p. 259].

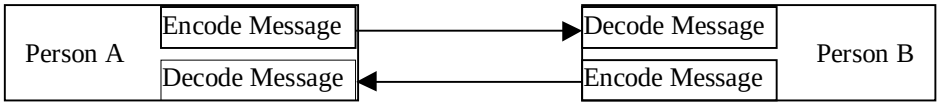

Figure 27. Model of person-to-person discourse.

Communication has been studied from a number of perspectives, such as linguistics and

cognitive psychology. There are many models of communication in the psychology literature, such as the Berlo Model, Information Theory Model, Barlund Model, Westly and MacLean Model, and Wenburg and Wilmot Model [107, p. 49-56]. A common theme in many of these contemporary models of discourse, similar to the Normative Code Model [56], is that a person encodes a message and sends it out to a person who receives and decodes it, as shown in Figure 27. Some models take into account internal or external behavioral cues, or added message (signal) noise to account for things like misunderstandings. The Wenburg and Wilmot model further takes into account the fact that, in theory, conversation can proceed for an infinite amount of time. In particular, they model conversation as an infinite cycle of each person constructing, saying and interpreting messages.

Wenburg and Wilmot state that small group communication (groups of 3 or more people) is more complex than interpersonal communication. However, they note that "it resembles dialog, all members have an equal opportunity to participate, feedback is immediate, correction of messages is still possible, and meaningful personal relationships still can be maintained." If a group gets too large, however, smaller cliques may form. For the purposes of our model, we will assume that the groups are small enough that the discourse is similar to interpersonal communication.

Corman models the communication perceived by an individual participating in the discourse based on a number of factors, including the physical distance and boundaries between them. He conducted a study of a public works organization and a large computer research and development center in an attempt to validate this model. The distance and boundaries between employees who communicated in the organizations were physically measured then compared to the data collected in interviews about their perceptions. Although we will not include the perception of communication in our model, it is interesting to note that he found evidence that boundaries (physical restrictions) that prevent direct communication decreases the perceived communication [22]. The results of Corman's study, however, may give insight as to why people might resist using a

distance (networked) collaborative environment. If the communication between the users is fraught with barriers, the users may become frustrated by their perceived decrease in communication and they may not wish to use that CSC system.

Models of discourse have also been used in artificial intelligence (AI) for designing conversational intelligent agents such as Shared Planning [37] and in the Collagen toolkit [90]. Here the discourse is described in terms of how the participants (human or computer) communicate their beliefs, desires and intentions. We also wish to describe the types of communication that occurs during a collaborative problem-solving task. However instead of attempting to create a computer-based agent, our goal is to measure or predict how the design of a computer program can influence the communication between the users of a multi-user program.

5.3 Our model of the user's interactions

We are interested in what types of and how much communication can and will occur between the people in specific situations. We feel that with appropriate software design, the intensity of user collaboration and communication may be raised. For example, a program that requires a pair of users to focus their attention on different parts of one object or on pairs of closely connected objects can make it necessary for the users to negotiate over control of components. We note that the goal of collaboration in some situations may not be to make the task easier, but may be to bring the users together in certain ways. Some tasks may be more difficult for users in a forced collaborative environment than they would be for a single user in a traditional single-user environment, but these types of tasks may require certain types of social, problem-solving or instructive communication in order for the users to succeed. An instructive analogy involves the winning of a running race; an individual runner could easily beat a pair of runners who each have one leg bound to one of the other's, but the "three-legged race" is an activity that has social, team-building, and entertainment value of its own. In general, we wish to discover what it is about collaborative activities that brings about communication and

aids users in discovering important aspects of a problem.

This model of collaborative communication measures two aspects of a cooperative interaction. The first is the extent to which if different cooperative interaction styles affect how well a user stays on a task. The second is the extent to which the different interaction styles effect the amount of certain types of verbal communication. The next two sections describe the types of interactions and our classification of communication types. Section 5.3.3 will describe the results of a user study conducted to validate these two aspects.

5.3.1 Types of interactions

As stated in the introduction, a cooperative interaction is one in which two or more users work together, each contributing a portion of the work on a task or problem. In Section 2.1, cooperative interactions were divided into categories based on the physical locations and simultaneity of the users' manipulations. Interactions were further subdivided in Section 4.5 based on whether the activity requires, encourages, discourages or disallows users to interact simultaneously. Intuitively, one might expect that users behave differently depending on how the activity defines the ways in which the users may interact. In particular, we expect users to stay more focused and communicate more frequently the more they are required to interact simultaneously in order to complete the task.

For example, in the case of an encouraged (but not required) synchronous cooperative interaction (such as in etalk), oftentimes the users 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. An etalk writer is not guaranteed that the other user is focused on the conversation the entire time; users often switch between etalk and other tasks while a thought is being completed. It would be more difficult, but not impossible, for a user to become unfocused when all users are required to simultaneously interact in order for the group to make progress on the task. Consider the following example where a line

segment cannot be moved unless two users each move their endpoint at the same time. If the activity is not carefully designed, one user could "pretend" to move their endpoint by pressing their mouse button down over the point, then never actually move the mouse. In this case, their finger is engaged but their focus may be elsewhere. In contrast, in the case of an encouraged simultaneous interaction, a user can make some progress on a task even if no other user is involved. The fact that the progress may be hampered or slower if the interactions are not simultaneous will more than likely encourage the users to work together at the same time.

One might expect that if the software discouraged simultaneous interactions, (perhaps by making it more difficult to do the task together) the users might try to work together. If the difficulty of this interaction frustrates the users, they would likely revert to some sort of turn-taking mechanism, and their focus would be more apt to drift off waiting for their turn to do some work on the task. This is even more likely to be the case if the software disallows a simultaneous interaction (the users cannot make progress on their task if they work together) and they must take turns.

5.3.2 Classifying types of communication

A communication act can occur through one of three channels: verbal (auditory), physical (gestural), or graphical (written, symbolic). Gestural communication might include pointing with a finger or with a cursor, or actively manipulating an object on the screen. Graphical communication includes textual representations of verbal communication, but might also include manipulating an object or drawing on the screen. The focus of this model is on measuring the amount and types of task-directed verbal communication, since this will be the primary form of communication between users of a highly interactive system that uses CCOs. We are particularly interested in measuring social, instructive, and problem-solving communication that may be desirable in cooperative learning or entertainment situations.

Tang [102] did an example of such a study with small groups of three to four people

working on a design activity. He videotaped and analyzed their interactions based on whether they were listing items, drawing, or gesturing around a shared work area. His findings indicate a need for collaborators to be able to view each other's gestures in order to communicate effectively on a cooperative task. Another, longer-term study was done by Smith, Smoll and Hunt. They observed and coded twelve categories of behaviors of little league coaches over several years. They classified these behaviors as reactive or immediate responses to player behaviors and spontaneous behaviors, which are not reactive [96]. We have chosen to classify verbal communication based on the types of behaviors listed above. In particular, we have chosen to classify communication based on the types of verbal behaviors that may be beneficial in learning situations and might also add entertainment value to an activity. For example, in a cooperative learning situation we would expect hear problem solving, instructive, and social interchange. One might expect to hear more of the problem solving and social communication in an entertainment situation. Each category contains a number of types of communication, as listed below.

Gesture (either with hand or mouse)
Social

- Taking control of the group or communication
  - Getting attention, such as "hey", "look at this", "ok"
  - Quieting the group
  - Personal communication, such as thinking out loud, or getting situated
- Synchronizing tasks
  - Asking who should go next
  - Saying "ready," "on the count of three", "1, 2, 3", etc
  - Accepting or acknowledging readiness
- Social skills
  - Apologizing
  - Teaching social skills, such as "you shouldn't interrupt," "it was my turn"
- Support
  - Discouraging, lack of support (like "We suck")
  - Encouraging, comments of reassurance
- Laughter and enjoyment, positive comments on the program, joking around about the program

- Frustration, Sighs, negative comments about the program, comments of frustration

Instructional or Teaching and Learning- Informing others of facts about the state of the world that may help in solving the problem, or participating in gaining that knowledge.

- Explanations (descriptions, narration, expositions, arguments - from Maybury) to help another user understand a concept. Also includes clarifications of concepts or ideas not understood by other users.
- Positive/negative reinforcement for understanding or not understanding a concept
- Interrogative (one user asking another for help or clarification of a concept being instructed)

Problem Solving (strategizing) accepting, rejecting, the formulation of and contribution towards the formation of a method to solve the problem

- Identifying and restating the problem (elaborating issues, discussing the context of the problem)
  - Discussing the current state
  - Discussing how close the current state of the activity is to the solution, like "that's pretty close"
  - Comments on score
- Formulating a strategy
  - Brainstorming/idea generation
  - Formulating methods
  - Stopping noises
  - Accepting the current state
- Evaluating ideas and alternative/estimating/weighing benefits and risks
  - Idea acceptance/rejection
- Task allocation
  - specifically telling one person to do something, like "Blue, you go up" or "Blue you do it now", or "you take this part, I'll take the other"

We had four additional categories to facilitate easy coding of the videotapes. An *off-task* communication event was one where the communication had nothing to do with the activity at hand, such as "do you want to go out."[3] An indecipherable event was one in that we just determine what was said from the audio of the videotape. An unclassifiable event was one that could be used by the person classifying the communication if he or

[3] Yes, this did really happen, although it was not captured on videotape. Before one of the trials, a male subject invited the two female subjects in his group to a fraternity party being held that same night.

she was uncertain as to the intent of the statement. Finally, an experimenter event was communication either to or from the experimenter. Typically this occurred when the subjects needed clarification on the program.

5.3.3 Studying types of communication

> *…to ask whether a model is right or wrong it not the proper question. The purpose of testing the validity of a model is to decide how and under what conditions its usefulness can be improved, not whether it is correct.*
>
> -- Pew [87, p. 664]

As stated in Section 2.1, common methods for classifying cooperative software is based on the physical location of the users and how simultaneously they can work together. For the physical location, users may be co-present or at a distance using a network. Many studies of collaboration have been done using different types of cooperative work systems and different channels of communication ([53] lists a number of references). For example, McDaniel et al [66] found that there were more unique threads of conversations when people communicated in a computer mediated communication (CMC) situation than in a face-to-face (FTF) situation. Isaacs and Tang [53] compared a desktop video conferencing system with FTF communication and telephone conversations in a case study. Among other things, they monitored gestures and measured how many times per minute a group took turns at speaking. Their findings lead to the development of some guidelines on what a desktop video conferencing system should and should not do. Heath and Luff [45] studied gestures in video mediated interactions as opposed to face-to-face interactions. They claim that while a distance situation with video is similar to co-presence, it limits the view of other users to small areas on screen, which limits how a user is able to perceive others' gazes and gestures, and thus how the users can and do interact. Finally, Sellen compared speech patterns in three different situations: co-present, a "picture-in-picture" (PIP) video system, and the Hydra system that consisted of a monitor, camera and speaker in lieu of where a person would normally be in a meeting. In this way, the when a user turned to face a monitor, they would also face the camera associated with that monitor, thus conveying gaze and gesture between the participants.

The hypothesis was that communication between the users in the Hydra system would closely resemble a face-to-face interaction in terms of the length of a user's turn, the number of turns taken, and the number of interruptions. They found that there was no statistically significant difference in objective measures of the two video systems. They did, however, find that users noticed the differences between the different conditions, and that they had a preferences on which to use.

We have chosen to focus on another typical classification of CSCW/CSCL applications. This classification is based on how asynchronously or synchronously users work together. An asynchronous cooperative interaction is one in which two or more users work on a task at separate times, such as delivering information through snail mail or email messages. Participants of this type of cooperative interaction do not expect an immediate response from questions or requests. 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 etalk are examples of a synchronous cooperative interaction. We have further broken up this "temporal" classification based on how simultaneously the users may interact with each other.

Recall that in Section 4.5.1 we introduced a categorization for simultaneous cooperative interactions. A *required* simultaneous interaction is one in which the users must manipulate objects at the same time in order to make any progress on the activity. An *encouraged* simultaneous interaction is one in which the users do not have to be manipulating objects at the same time, but doing so will allow them to complete the task more easily or in a shorter amount of time. A discouraged simultaneous interaction is one in which, although a simultaneous interaction may be possible, it may be more difficult to work at the same time. This may revert into an asynchronous interaction. A *disallowed* simultaneous interaction is when the users are unable to work together at the same time.

The next sections present a user study that was performed to gauge how different types of interaction styles influence a group's communication during a task. We wished to focus

on three different styles of simultaneous interactions, *required, encouraged,* and *disallowed*.

5.3.3.1 Method

We performed a user study to measure how different types of user interaction influence whether the users stay on-task and how it affects the ways in which they communicate. Sixty-six undergraduate students from the University of Washington psychology subject pool participated in groups of three. Each group was given a unique identifier, and each user was identified by the color of their cursor, but their actual identities were kept anonymous. Ten groups (thirty students) were used as a pilot study to verify our testing and analysis methods. The results of the twelve groups (36 students) we used for the actual study will be presented in Sections 5.3.3.3, 5.3.3.2 and 5.3.3.3.

The complete procedure for the study took less than an hour and is listed in Appendix A. Briefly, for each of the groups, a video camera was pointed at the monitor and started at the beginning of the activity. In this way, the action on the screen was filmed and the voices recorded without compromising the anonymity. In order to discern which voice belonged to which subject, each user was asked to say, for example, "I'm in group *5* and I am color *blue*." Each user was also instructed to wear a colored pipe cleaner on each of their hands so we could later distinguish who initiated pointing gestures aimed at the screen. Each user was given a Microsoft Sidewinder Gamepad [69] joystick as an input device and instructed in their use.[4]

During the course of their hour time slot, each group of three students interacted with the cooperatively controlled triangle in the three different conditions, listed in Table 7. The conditions were renamed from our technical terms (encourage, required, disallowed) during the experiment to avoid influencing the users. The conditions were presented to

---

[4] The users were told to move the triangle they must be "active" on the vertex that corresponds to their cursor color. Pressing the button labeled "A" while the cursor is over the vertex activates the vertex. Once active, the vertex could be moved using the directional button on the left side of the joystick.

the subjects in a mixed order to attempt to eliminate order effects. For each condition the group was given a description of the condition and how the joysticks must be used in that condition, then given one color on which to practice, then tested with three colors. The colors for each condition were held constant over all of the groups.

When the users completed a condition, the files resulting from the testing were saved for later comparison. The subjects also filled out a post-test questionnaire (shown in Appendix B).

Table 7. User testing conditions for the Color Matcher software.

Simultaneous interaction style	**Condition name**	**Description**
Disallowed (Individual)	Serial	Each user must move his or her part of the object independently.
Encouraged	Conjoined	Each user can move his or her part of the object at any time, or the users can move together.
Required	Coactive	Each user must move his or her part of the object in order for the object to move.

The software used for this study was a version of the Color Matcher Activity modified to include a Cooperatively Controlled Object (CCO) to set the users' color. The new version of the Color Matcher, described in Section 7.6, was developed with the Colt System, which is described in Chapter 6. The CCO in this version allows each user to control a vertex of a triangle. As the vertices of the triangle move, the location the centroid of the triangle changes. The location of the centroid of the triangle determines the users' color by choosing the color below this point from a color bitmap. As with the original version of the software, the users are given sixty seconds to match the target color, although they are allowed to end a trial at any time by pressing the check button to reveal their score. As the users worked with the new version of the activity, the system collected event span data for the triangle, to record how each user was manipulating the objects on the screen.

After the user testing was complete, we analyzed the videotapes, data collected by the software and post-test questionnaires. The system data was analyzed using a view of the recorded event spans and the measure of joint activity (discussed in 4.4) that are a part of the Colt system. Data from the questionnaires was analyzed to determine if the users enjoyed themselves, whether or not they were frustrated by the activity, and what were their perceptions of their communication.

The analysis of the videotapes was time consuming and problematic for two reasons. The first was that because we were not able to videotape the users' faces, it was often impossible to discern between the voices of two users of the same sex. The second problem was that since the testing was conducted with the computer present in an ordinary office, the noise level on the videotapes was very high. The distractions on the tape included noise from other people in the office, airplanes flying overhead, and even geese. To solve this problem we digitized the audio from the videotapes and used Cool Edit 96 sound editing software [101] to reduce the worse periods of noise on the tapes. The verbal communication between the users was transcribed from the digitized version of the audio.

The videotapes of the sessions were replayed once to transcribe the audio conversation, and a second time with the text scripts to determine the communication events. The communication between the users was coded using a checklist based on the classification of communication listed in Section 5.3. After communication was classified, the events in each category were tallied. The videotapes were analyzed in two phases. In the first phase, the videotapes were analyzed from the pilot study groups in order to verify the classification and make any corrections and additions. Also during this phase, the results of two groups from the primary rater were compared to the results from a secondary, untrained rater to determine the inter-rater reliability of the classification. The difference between the two raters was an average of 16% for social communication and 11% for

problem solving communication.[5] We expect that had the secondary rater been trained, the results would be even closer. The same method for classifying the communication was used during the second phase of analysis, except that no changes were made to the checklist.

We had a number of hypotheses that we wished to examine during this testing. Some of the results were collected using the event history and analysis mechanisms, which will be described in Chapter 6. Data relating to the communication was collected from the videotapes, as described above. Here are three of the hypotheses, expressed as expectations:

1. The users will work simultaneously in the conjoined condition, even though they are not required to do so. (If they do work together, what percentage of time do they work simultaneously together?)

2. The types of communication will change depending on the types of interaction enforced by the software. For instance, if the users in the disallowed or required simultaneous (serial or coactive) interaction condition, we will find that there are more "coordination" or "social" communication events than in the other conditions. For the encouraged (conjoined) condition, we expect to find less coordination than problem solving.

3. The users will enjoy the tasks and not be frustrated by them. If there is a difference between the conditions, the users will enjoy and be least frustrated by the conjoined condition more than the serial and coactive conditions. Intuitively, this would be because the users will react negatively when the software does restrict how they interact.

The first hypothesis was tested using the checklist analysis of the videotapes. The second hypothesis was tested using the event span analysis and visualizations. A question that is related to the third hypothesis involves the significance of the interaction. We would have

[5] The comparison of the instructional communication is omitted because of the low frequency of occurrence of those communication events. This will be discussed further in Section 5.3.3.3.

liked to determine, in the coactive condition how much the users "spoof" the computer by selecting an object so that it appears they are actively manipulating an object, but are really manipulating the object very little. Unfortunately we were not able to answer this question because the software at the time of testing did not support tracking this kind of data. The last hypothesis was measured by analyzing the responses to the questionnaire. The results and discussion for each of these hypotheses will be presented in the next three sections.

5.3.3.2 Hypothesis 1 results and discussion

In order to verify the first hypothesis, we must first quantify how much the users work together in each condition. The measure of joint activity (JA, from Section 4.4.2) was calculated on the event spans generated by the users, producing results listed in Table 8.

The first thing to notice is that for all of the groups, JA = 1 for the serial condition, which is to be expected. A visualization of a typical trial in the serial condition is shown in Figure 28. This visualization, as well as the others shown, depicts the events for the centroid of the triangle when shown in the span view (described in Section 6.4). Because these figures are black and white (not color) images, the blue user is shown in black, the red user is shown in dark gray and the green user is shown in light gray.

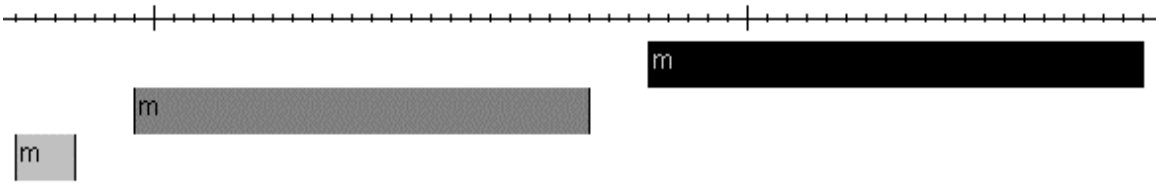

Figure 28. A visualization of a typical serial interaction by Group 8.

One might expect that since all the users must be active in order to move the triangle, JA = 3 should hold for each group in the coactive condition. Although this is not the case, the values are fairly close ($\overline{JA}$ = 2.78, s = 0.22). This can be explained by examining the visualization of the event spans. Figure 29 shows a visualization of an optimal synchronization in a coactive condition. The total active time in this trial was spent

working as a group of three, or JA = 3. In most cases, however, the synchronization appeared to be off slightly, as in Figure 30. One case in which a group apparently had a lot of difficulties synchronizing their activity is shown in Figure 31.

Table 8. Results of the measure of joint activity (JA).

Group	**Measure of joint activity (JA)**			**$JA_c < JA_a$?**
	Serial (JA_s)	**Conjoined (JA_c)**	**Coactive (JA_a)**	
1	1	2.32	2.59	yes
2	1	2.45	2.84	yes
3	1	2.83	2.94	yes
4	1	2.44	2.70	yes
5	1	1.48	2.51	yes
6	1	2.44	2.97	yes
7	1	2.69	2.85	yes
8	1	1.91	2.89	yes
9	1	2.36	2.31	no
10	1	2.89	2.86	no
11	1	2.05	2.76	yes
12	1	2.75	2.95	yes
average	1	2.38	2.78	
stdev	0	0.45	0.22	

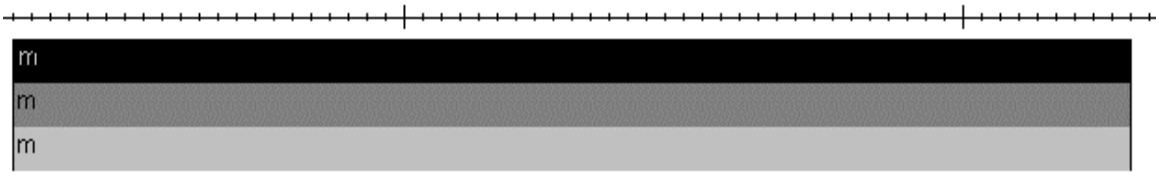

Figure 29. A visualization of an optimal synchronization in the coactive condition by Group 8.

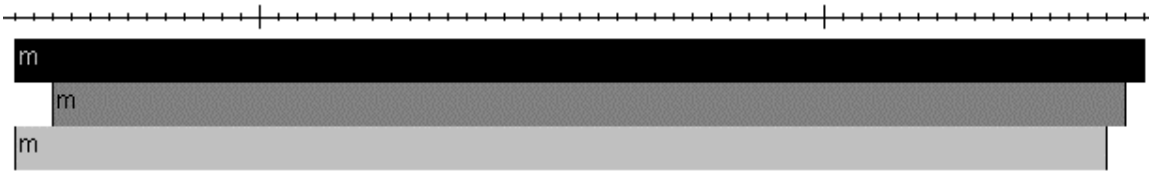

Figure 30. A visualization of a typical synchronization in the coactive condition by Group 10.

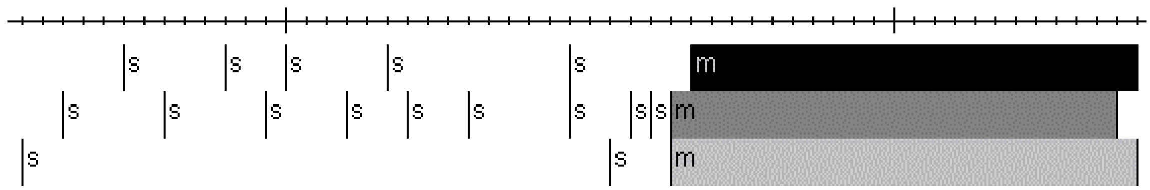

Figure 31. A visualization of the difficulties Group 5 had synchronizing in the coactive condition.

As one might expect, the average JA for the conjoined condition ($\overline{JA}_c$ = 2.78, s = 0.45) is greater than the average JA for the serial condition ($\overline{JA}_s$ = 1, s = 0) and less than the average JA for the coactive condition ($\overline{JA}_a$ = 2.38, s = 0.22). However, this was not the case for all of the groups. In particular, Groups 9 and 10 did not follow the expected pattern, although in each case the difference between JA_a for the conjoined and coactive conditions is less than 3% of the value of the JA_a (0.05 and 0.03 respectively). We suspect that this is due to a learned effect. Both of these groups were presented the conjoined condition last, so the users had time to learn the interface and how to activate and move their vertices smoothly. For example, the blue user in Group 9 (shown as black in Figure 32) activated their cursor (pressed the A-button on the joystick) for each small movement she did. We call this behavior "bouncing" because we observed both of her thumbs bouncing rapidly and simultaneously up and down on the buttons to attain a smooth movement of the vertex smoothly. This type of behavior will reduce the JA since there is less time when all three users appear to be moving the object simultaneously, and more time where a pair (green and red in this case) are moving. This is shown in Figure 33.

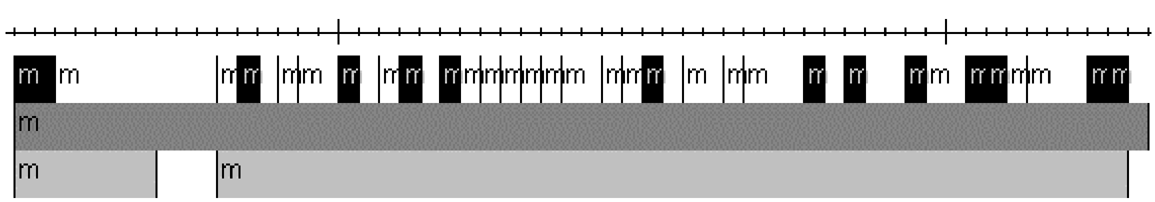

Figure 32. A visualization of the "bouncing" behavior exhibited by the blue user in Group 9.

In contrast, by the last trial of the last condition, which happened to be conjoined condition), the same user appears to have figured out that simply pressing and holding the A-button while moving the cursor affords smoother movement of the vertex. This is shown in Figure 33. A solution to this problem better explain how to use the joysticks at the beginning of the session. One way to do this would be to explain that the joysticks function the way a mouse works, with the A-button being equivalent to the left mouse button.

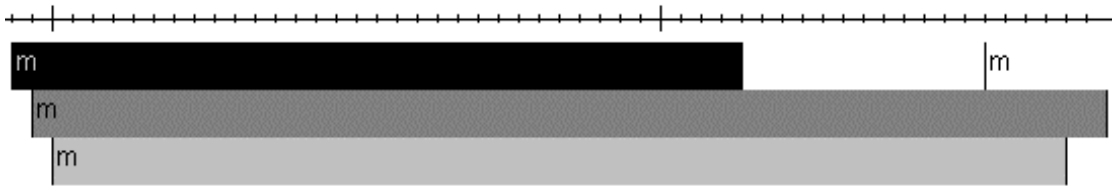

Figure 33. A visualization of the last conjoined trial for Group 9. The blue user seems to have learned how to move the vertex more smoothly.

The data does support the fact that $\overline{JA}_s < \overline{JA}_c < \overline{JA}_a$ with $p < 0.05$ using a statistical test called the *sign test*. We can explain sign test in the following way: if we had a perfectly random sample of numbers, the probability that $\overline{JA}_c < \overline{JA}_a$ is ½. Thus we would expect that out of the twelve groups in the study, six of them would have $\overline{JA}_c < \overline{JA}_a$ and six would have $\overline{JA}_c > \overline{JA}_a$. However, the data shows that only two have the latter property. The probability of this occurring (using Equation 9) is:

$$\left(\binom{12}{0} + \binom{12}{1} + \binom{12}{2}\right) * \left(\frac{1}{2}\right)^{12} = (1+12+66) * \left(\frac{1}{2}\right)^{12} = 0.019$$

The JA for the conjoined condition is between that of the serial and the coactive condition, as we expected. Our hypothesis states that the users in the conjoined condition will act more like the coactive condition than the serial condition. If this were true, we would expect the difference between the JAs for the conjoined and coactive conditions ($D_a = abs(JA_c - JA_s)$) would be less than the difference between the JAs for the conjoined and serial conditions ($D_s = abs(JA_a - JA_c)$), or $D_a < D_s$. These values are shown in Table

9. The data shows that, in all but two cases, this fact does indeed hold ($p < 0.05$ using the sign test as above). Groups 5 and 8 manipulated the triangle more individually or in pairs than all three of them together when performing the conjoined condition. There does not seem to be any specific reason to explain this behavior, simply that the users in these groups did not work much simultaneously in groups of three. A visualization of this is shown in Figure 34.

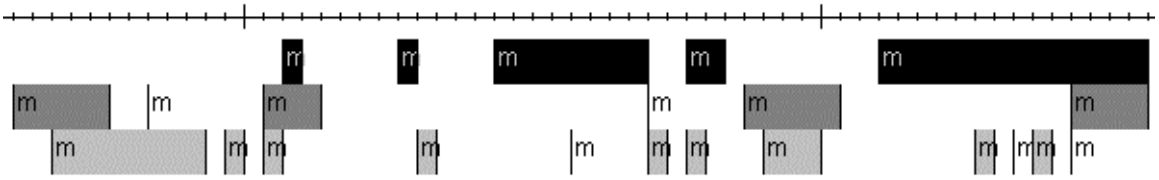

Figure 34. A visualization of Group 5 performing a trial in the conjoined condition.

Table 9. Difference between values of JA for each condition.

Group	D_a = abs(JA_a - JA_c)	D_s = abs(JA_c – JA_s)	D_t = JA_a - JA_s	D_a / D_t *100%	< 50%?
1	0.27	1.32	1.59	17.00	yes
2	0.39	1.45	1.84	21.28	yes
3	0.11	1.83	1.94	5.79	yes
4	0.26	1.44	1.70	15.13	yes
5	1.03	0.48	1.51	68.10	no
6	0.53	1.44	1.97	27.07	yes
7	0.16	1.69	1.85	8.40	yes
8	0.99	0.91	1.89	52.16	no
9	0.05	1.36	1.31	4.07	yes
10	0.04	1.89	1.86	2.00	yes
11	0.71	1.05	1.76	40.27	yes
12	0.21	1.75	1.95	10.62	yes
average	0.40	1.38	1.77	22.66	
stdev	0.35	0.41	0.20	23.13	

Note that the JA computed in this study did not include an adjustment for the significance of each user's interactions. However, we did notice that there might be a need to weight

the JA by the significance, as discussed in Section 4.4.4. Group 14 actively gestured during all of their trials, including 20 gestural events during the coactive condition. However, the structure of the GamePad joysticks used in this study requires both hands to continuously move the cursor. The users were obviously able to activate their part of the triangle, begin a *Move* program event so that the triangle could be manipulated, then continued to be "feign" activity (even though their vertex was not actively moving) while gesturing. One user even said "I think in order to avoid confusion we should, even if we all hold it down, only move one person at a time."

5.3.3.3 Hypothesis 2 results and discussion

The videotapes were analyzed to attempt to verify the second hypothesis as well as the categories of communication listed in Section 5.3.2. One initial result of this analysis was that there was only one instructional and one off-task communication event during the actual part of the study. We included these categories because we felt that the users might exhibit them. Additionally, we found a couple of groups in the pilot study did do instructional communication, although not very often. The lack of instructional communication events may be due to the nature of the activity rather than the interaction style. However, since there was no data for instructional communication, we will ignore that category for the balance of this analysis.

Although we had a veritable lack of instructional communication events, the categories used for this study seem fairly reasonable. As stated previously in the description of the study method, the results of two groups were compared from a primary and secondary rater. Even though the secondary rater was untrained, the difference between the two raters was an average of 16% for social communication and 11% for problem solving communication. We expect that training raters would secure closer results.

We compared the relative frequencies of each type communication event in each of the conditions to verify the second hypothesis. This hypothesis is that the types of communication will change depending on interaction styles enforced by the software. The results of the analysis of the videotape are listed in Table 10.

Table 10. The number of groups where this comparison of the frequencies of communication events held true. (n=12 groups). The sign test was used to determine significance.

Comparison of frequency of communication events	**Number of groups where this comparison held true for communication type:**		
	Social	**Problem Solving**	**All**
Coactive > Serial	8 (p=0.194)	7 (p=0.387)	7 (p=0.387)
Serial > Conjoined	9 (p=0.073)	7 (p=0.387)	7 (p=0.387)
Coactive > Conjoined	11(p=0.003)	9 (p=0.073)	9 (p=0.073)

From this table we can see that the only statistically significant result is that social communication events are more frequent in the coactive condition than the conjoined condition. Three other comparisons were nearly significant ($p<0.10$): there was more social communication in the conjoined condition than in the serial condition; there was more overall communication in the coactive condition than in the conjoined condition; and there was more problem solving communication in the coactive condition than in the conjoined condition. Only this last result seems counter-intuitive. Based on our theory that the users would spend more time coordinating their work in the coactive condition, we expected that the users would spend less time problem solving in that condition. Apparently this was not the case. Perhaps this is because they were encouraged to communicate overall, so even the amount of problem solving communication was increased by the interaction style.

We were also interested to see how the users felt they communicated with others in the group. We used the users' responses to the statements on the questionnaire to gauge their impressions. The average responses from the questionnaire data are listed in Table 15 in Appendix C. For each statement on the questionnaire the users circled a value between 1 and 7, where 1 meant they strongly agreed with the statement, and 7 meant they strongly disagreed. A response of 4 meant they were felt neutral about the statement.

There were three sets of statements to determine if the users had different opinions about

their communication in the three conditions. We used the Freeman Rank Sum two-way layout distribution-free test (with large sample approximation, S'), described in [48] to ascertain if there was a difference between the responses in each condition. The hypothesis of this statistical test is that there is no difference between the responses from the different conditions. If the results S' are greater than the chi-squared (χ^2) distribution for two degrees of freedom at $p < 0.05$, then we reject this hypothesis. The results for S' on the questionnaire data are shown in Table 11. According to these results, we cannot reject the hypothesis, and thus the conditions did not have an effect on the responses from the users.

Table 11. Results from the distribution-free two-way layout test (S') for statements about the users' communication. For each statement we reject the hypothesis if $S' > \chi^2(2, 0.05) \cong 6$ (χ^2 is the chi-squared distribution for two degrees of freedom). The results indicate that the users' responses are different for the three conditions.

Set of statements about condition *C* (n = number of users)	**Statement numbers**	**Freeman Rank Sum (S')**	**Reject?**
Our group talked a lot in the *C* task of the program. (n=36)	5-7	0.795	No
I feel that in the *C* task of the program, our group didn't communicate much. (n=36)	11-13	1.420	No
I think that the *C* task of the program encouraged our group to communicate a lot. (n=36)	35-37	0.951	No

The results of this statistic allow us to average all of the responses to the statements for each condition. The average response to "Our group talked a lot in the [serial, conjoined, coactive] task of the program" was indicated that they mostly agreed with the statement (average = 3.28, s = 1.98). The average response to the converse statement, "I felt that in the [serial, conjoined, coactive] task of the program our group didn't communicate very much" was not quite as strongly negative as the previous statement was positive (average

= 4.72, s = 1.95). Finally, the average response to "I think that the [serial, conjoined, coactive] task of the program encouraged our group to communicate a lot" indicated that the users did agree with the statements (average = 2.92, s = 1.71). These responses are shown graphically in Figure 35.

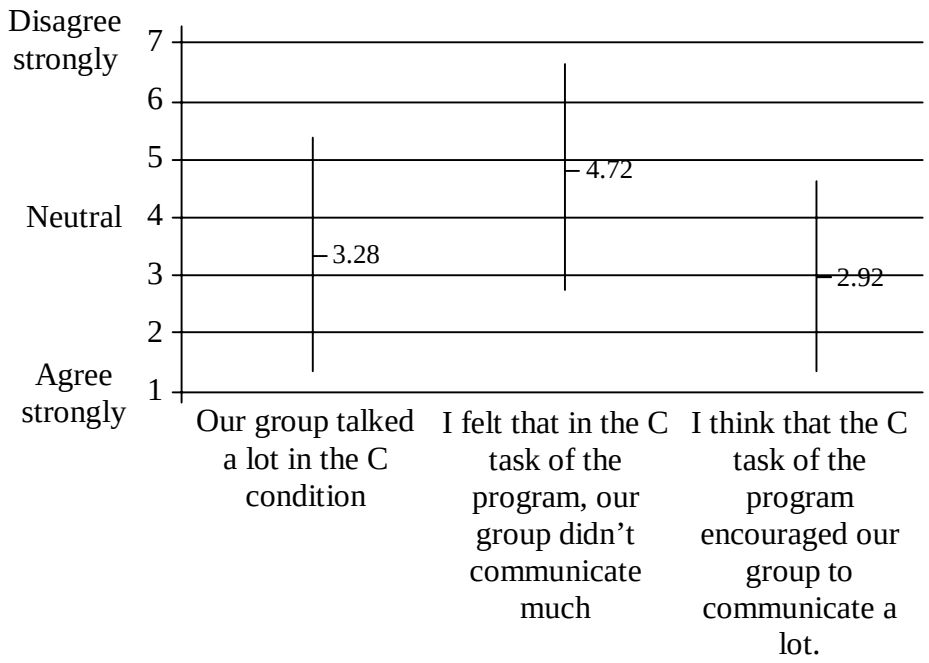

Figure 35. Average responses (± standard deviation) to the statements about the users' perception of communication.

5.3.3.4 Hypothesis 3 results and discussion

One of the major criticisms of this work has been "why would anyone want to work this way?" Critics express concern that the highly synchronous interactions being explored in this research are not the most productive method of completing a task, and that they may frustrate the users. Our contention in Section 2.6 is that there are times when a designer might want to bring the users together in certain ways, perhaps in an attempt to help them learn or communicate more. Additionally people seem to enjoy working closely together. Still, we felt we should address the issue of whether the user enjoyed this type of interaction and whether they were frustrated by it.

We used a combination of questionnaire responses and communication behaviors to validate the third hypothesis. From previous studies, we expect that the users will enjoy the activity and that this type of interaction won't frustrate them. If there is a difference between the conditions, we also expect that they will be least frustrated by the conjoined condition. Intuitively, if there were any difference between the conditions, it would be because the users might react negatively to software that restricts how they must interact.

As stated previous, the questionnaire consisted of a set of statement that users responded to by circling values from 1, meaning they agree strongly with the statement, to 7, meaning they disagreed strongly. The average responses from the questionnaire data are listed in Table 15 in Appendix C.

The first statement we presented, "I like doing these tasks with others," was simply to get an overall feeling for whether or not they enjoyed these tasks in a group situation. The answer was positive, with an average response of 2.42 (standard deviation, $s = 1.57$). The users disagreed with the statement "I think the tasks were boring" (5.42, $s = 1.18$) and, correspondingly, agreed with the statement, "I think the tasks were interesting" (2.89, $s = 1.39$). The response to the statement "Overall I thought the interface for these tasks were easy to use" was also positive (2.56, $s = 1.30$). This is shown in Figure 36.

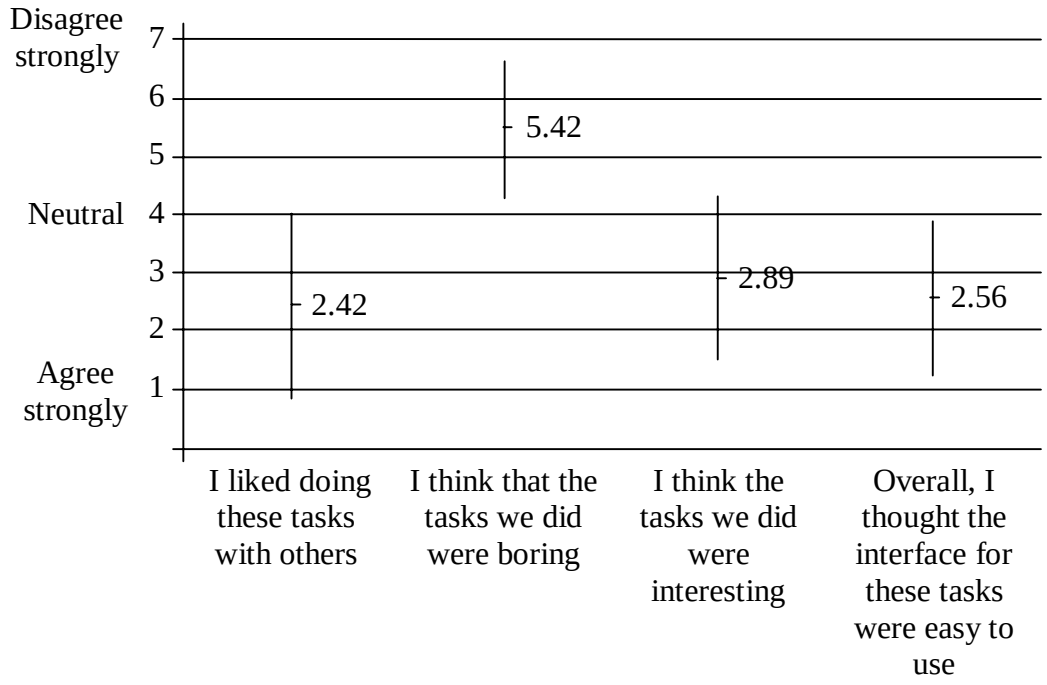

Figure 36. Average responses (± standard deviation) to some general statements about the Color Matcher activity.

The questionnaire contained four statements to determine if the users' enjoyment and frustration levels were effected by the different conditions. As with studying the users' communication (described in Section 5.3.3.3), we used the two-way layout distribution-free test and compared the results to the chi-squared distribution (χ^2 (2, 0.05) ≅ 6). The results for two of the questions "I was frustrated by the [serial, conjoined, coactive] task" and " and "I thought it was easy to do the program in the [serial, conjoined, coactive] task" were indicated that the conditions did have an effect on the response from the users. This analysis is shown in Table 12[6].

[6] Three users did not respond to the statement "I thought the serial task was entertaining" their data was eliminated from the analysis for this set of statements. We suspect that they missed this statement since it was hard to see on the questionnaire in its location and due to a lack of surrounding white space.

Table 12. Results from the distribution-free two-way layout test (S'). For each statement we reject the hypothesis if S' > χ^2 (2, 0.05) ≅ 6 (χ^2 is the chi-squared distribution for two degrees of freedom). The results indicate that the users' responses are different for the three conditions.

Set of statements about condition *C* (n = number of users)	**Statement numbers**	**Freeman Rank Sum (S')**	**Reject?**
I was frustrated by the *C* task. (n=36)	20-22	10.424	Yes
I thought the *C* task was a productive way of performing the program. (n=36)	27-29	3.156	No
I thought the *C* task was entertaining. (n=33)	31-33	3.042	No
It was easy to do the program in the *C* task. (n=36)	38-40	11.094	Yes

The users, as expected, somewhat disagreed with the statements "I was frustrated by the [serial, conjoined, coactive] task" (serial = 4.94, s = 1.64; conjoined = 5.47, s = 1.46; coactive = 5.14, s = 1.57). Also as expected, the users disagreed most with the statement for the conjoined condition. Perhaps surprisingly, however, they responded that they disagreed more with the statement for the coactive condition than the serial condition. This is shown graphically in Figure 37.

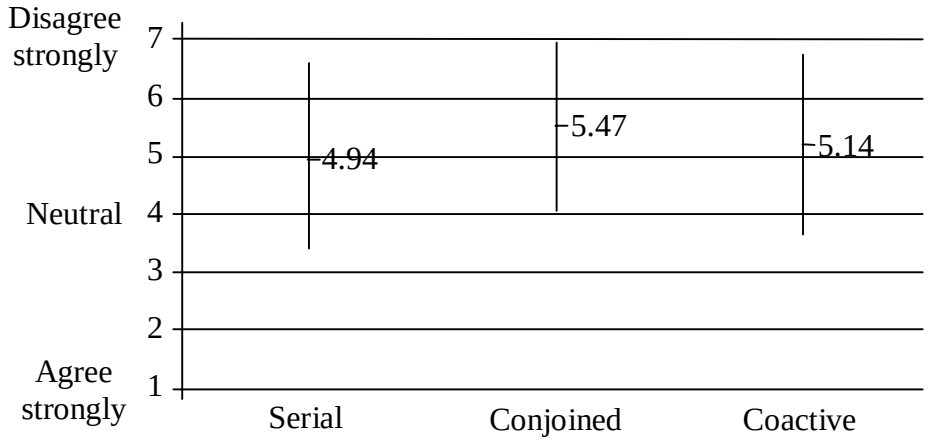

Figure 37. Average responses (± standard deviation) to the statement "I was frustrated by the [serial, conjoined, coactive] task."

These results of the Freeman Rank Sum indicate that we can average all of the responses from all three conditions to the statement “I thought the [serial, conjoined, coactive] task was a productive way of performing the program.” The users slightly agreed with the statement (average = 3.35, s = 1.52). We were surprised at this positive response, especially considering productivity is not necessarily one of the goals of a cooperatively controlled object, such as the triangle. Also based on the results of the Freeman Rank Sum we averaged the results from all three conditions to the statement “I thought the [serial, conjoined, coactive] task was entertaining.” As expected, the users responded with general agreement (average = 2.87, s = 1.53). These responses are shown graphically in

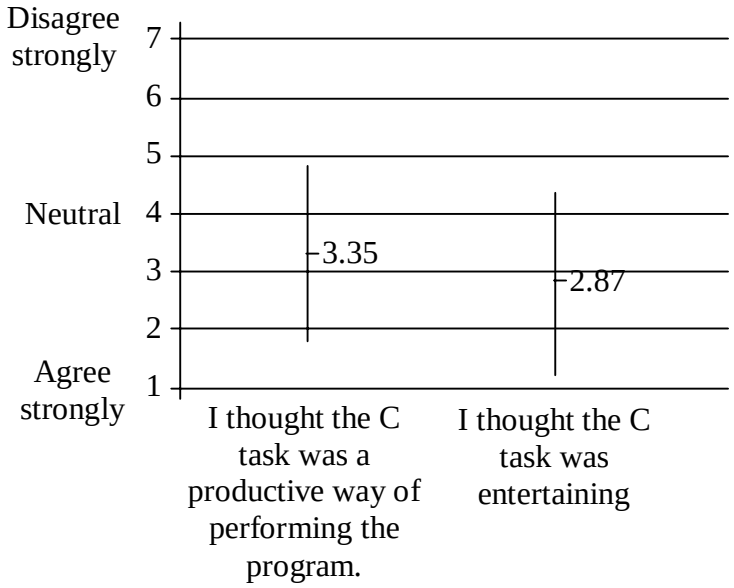

Figure 38. Average responses (± standard deviation) to the statements about how productive and entertaining the program is.

Another surprise was the general agreement with the statements “I thought the [serial, conjoined, coactive] task was entertaining” (serial = 3.81, s = 1.53; conjoined = 2.89, s = 1.35; coactive = 3.36, s = 1.46). As expected the users agreed the most with the statement for the conjoined condition, but they also agreed less with the statement for the serial condition than the coactive. This is shown in graphically Figure 39. We can only assume that the users felt that taking turns, as the serial condition forces them to do, was

seen as harder and less productive than all of them working simultaneously together. Furthermore, it seems that the users were not terribly bothered by the need to coordinate their actions in the coactive condition.

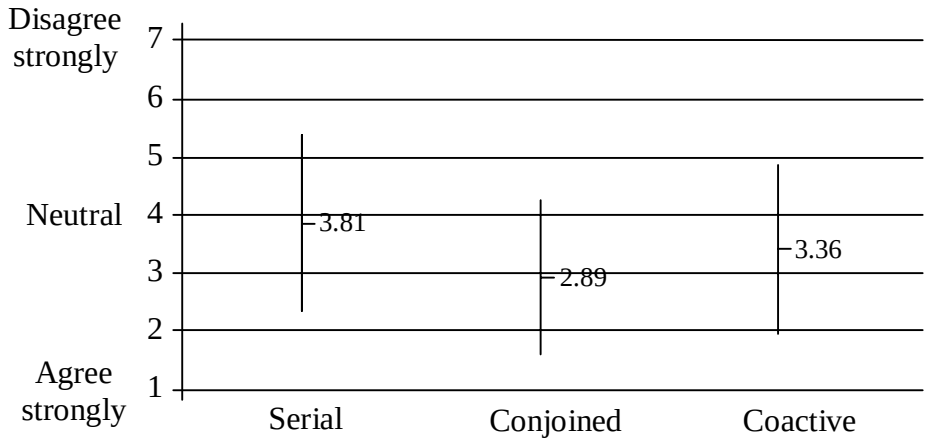

Figure 39. Average responses (± standard deviation) to the statement "It was easy to do the program in the [serial, conjoined, coactive] task."

Table 13 shows a the correlation statistics on whether the users experience or comfort level with computers had any effect on their frustration with the task or how entertaining it was to them. We wanted to make sure that the users weren't hindered by lack of experience with computer systems or the non-standard input devices. We take it as a positive sign that there was not much correlation between these factors. This means that users, regardless of their experience level, should be able to perform tasks that require highly synchronous interactions.

Table 13. Correlation between user experience and their frustration and enjoyment level doing the task.

The users felt	In condition	Users are comfortable with computers	Users play video games	Users play computer games a lot
Frustrated	Serial	-0.13	0.33	-0.08
	Conjoined	-0.25	0.34	-0.22
	Coactive	-0.21	0.36	-0.17
Entertained	Serial	-0.19	0.37	-0.09
	Conjoined	-0.00	0.42	0.01
	Coactive	-0.05	0.45	-0.04

We also used the data from the video tape analysis to verify that the users enjoyed each condition and were not answering positively on the questionnaire just to be polite. We assume that the laughter on the videotape is a good indication of a user enjoying the activity. Admittedly we cannot be certain that all of the laughter we classified indicated enjoyment, some may indicate that the user felt uncomfortable. However, statements such as “This is a real good time” (group 16) and “I'm going to get this game, laugh” (group 3) are a fairly good indication. Similarly communication such as sighs or statements of frustration can be taken as an indication of the users frustration with the activity. We found that in each condition the users no more behaviors of frustration than of enjoyment, which corresponds to the results we got from the questionnaires that they enjoyed the activity. The results of this analysis are shown in Table 14.

Table 14. Results of comparing the number of frustration and enjoyment communication events on the videotape.

Condition	Number of groups where there were more frustration events than enjoyment events (n=12)	Sign test
Serial	3	$p < 0.1$
Conjoined	0	$p < 0.05$
Coactive	1	$p < 0.05$

5.4 Summary

This chapter has presented a model of the human-human interaction and a method for measuring communication between users working on a task. We have presented a classification of communication that is used in highly synchronous collaborative problem solving tasks. We also have presented a user study that illustrates the use of the model and Colt's analysis tools. In doing so we verified our classification of communication. We also found that the users communicated in a social way more in the coactive condition, when they were required to work simultaneously together than in the conjoined condition, when they were only encouraged to do so. We also found that the users, when not restricted by the software to working together or apart, tend to work fairly closely together on a task. Finally, we found that users were not terribly frustrated by the interface, and that they found the interaction entertaining.

The next chapter will present a system for developing collaborative applications. The system uses concepts developed in the model that has been presented in this chapter as well as Chapter 3 and Chapter 4.

CHAPTER 6: COLT: A SYSTEM FOR DEVELOPING SYNCHRONOUS COLLABORATIVE APPLICATIONS

6.1 Background

Most software developers agree that designing and implementing any application is a non-trivial task. Creating a collaborative application, that is where more than one user interacts with the same application or data, is even more difficult because of the added complexity involved with the cooperative interactions. Furthermore, only recently have designers explored new ways in which users can work together on a problem, and thus only recently have these issues received much attention.

Until recently, software was typically developed using the "waterfall" model; that is, the product was first designed, then implemented, tested, and released. This model worked fairly well for batch systems that did not require much human intervention. With the advent of interactive computers, this system was found to fall short. Application developers found that it was difficult to get the requirements for an application right the first time, so they moved towards an *iterative* design process that includes several cycles of prototyping and testing. Boehm coined the term "spiral" model to describe the iterative nature of this form of design [2].

Baecker, et al, define development tools as "tools that help the developer convert interface specifications into an interactive system and that support all phases of system refinement including prototyping, implementation, testing, maintenance and system enhancement" [4, p313]. This chapter presents Colt, a tool to aid in the iterative design process of computer supported collaborative activities. The Colt system is comprised of three parts a design methodology, a software toolkit, and visualization and analysis tools. The design methodology described in Section 6.2, focuses developers during the creative

process of designing a collaborative activity. The toolkit, described in Section 6.3, supports the rapid development of collaborative programs. The visualization and analysis tools, described in Section 6.4, present different views of how the users worked together on the system. These tools can be employed by designers, to see if the implementation meets the original design criteria, or by teachers (or researchers), to see how well the students (or subjects) are collaborating.

6.2 A design methodology for cooperative applications

There are many techniques that have been devised to facilitate system development, including models, principles, guidelines, or design methodologies [3]. One reason to develop a model is to aid in the design of an application and to help the developer remember specific aspects or features of a design [87]. However models, such as the one presented in Chapters 3-5, can be too overwhelming or specific for someone who is just beginning the design of a collaborative activity. *Principles* are statements of recommendation aimed at a designer. For example, Cockburn and Jones outlined four principles of groupware design [19]: maximize personal acceptance of the software, minimize both the requirements and the constraints the software imposes on the users, and to maximize the integration with external systems the users employ. However, principles tend to give advice at a very high level. *Guidelines* are tests that can help one to judge if an interface is satisfactory. Baecker et al, state that "*Methodologies* are more or less formalized procedures that are believed to guide and structure the process of effective design when followed in sequence" [3, p. 74].

The Colt design methodology is based on the model described in Chapters 3-5 and includes the worksheet shown in Appendix C. The worksheet provides a framework for organizing thoughts on the design in an attempt to reduce development processes. The design methodology is intended as a technique can that bridge the gap between a very specific model of a computer supported collaborative activity and the implementation of that activity using the toolkit described in Section 6.3.

The next few sections will describe the four major portions of the worksheet. Questions in each section will be related back to our model. The answers to these questions will influence the implementation, and how the toolkit is used. We also recommend that the worksheet be used in conjunction with storyboarding techniques most notably employed by film animators. One technique that seems to work well for storyboarding is to put up large sheets of paper in a conference room, then have a group brainstorming session. The designers are allowed to draw and develop ideas in a freeform manner on the sheets.

6.2.1 The activity

The first few questions on the worksheet focus the designer on the activity that is to be presented to the user. Question 1 is "First, think of an activity or a scenario that you want to develop. Briefly describe the activity or scenario." This simply asks the designer to identify the activity to be developed, including who the target audience of the product is. The goal of the activity was discussed in Section 3.2 as being an important part of the activity, so part *a* of question 1 asks "What is the goal of the activity".

Parts *b* ("What context do you see this activity used in") and *c* ("What is the age range of the people doing this activity" relate back to the taxonomies of computer supported collaboration that were presented in Section 2.1. These questions help classify the activity in terms of computer supported cooperative work (CSCW), learning (CSCL) or entertainment (CSCE). This distinction may be very important in terms of how you might want the users to interact. For example, whereas the designer of a CSCL or CSCE application might want the users to manipulate an object in a simultaneous manner, this might not be a useful technique for objects in a CSCW application. Part *d* of the question categorizes the activity based on the spectrum of activities, shown in Figure 2. This classification of the activity may help keep the design focused.

Parts *e* and *f* simply ask "Is the activity a solely a computer application, or does it require other materials (like worksheets, etc)?" and "What role do you see a computer application taking in this activity (as a tool, as the activity itself)?" This is to compel the designer to

think about the role a computer plays in the activity. A collaborative activity might include a computer-based component as a small part of the activity, or may not include technology at all. A computer-based collaborative activity might also require other materials, such as workbooks.

6.2.2 The users

Part II of the worksheet focuses the designer on the people who are going to use the activity. This section relates to the model presented in Chapter 5. Question 2 is "How many people will use your activity simultaneously?" Large groups of users will impact where they may be situated in a collaborative interaction. Groups of two to four users can reasonably sit together around a standard computer monitor. If the expected group size is greater than four users, the activity may have to be developed for a system that uses large screen technology (such as a meeting room whiteboard system) or for collaboration at a distance.

Question 3 asks, "Where are the users located?" to specifically address the location of the users. The answer to this question may have an impact on the ways in which the users can manipulate the objects in the application. As stated in Section 2.1, simultaneous cooperation at a distance may be hampered by network delays using current technology. Additionally, the answer to this question will determine the type of environment used for development. These options will be covered in greater detail in Section 6.3.3 and Appendix D.

The answer to question 4, "Are the users going to know each other before using the activity? Or do they meet through using the activity?" might also impact the type of environment the users are able to work with. It may be easier to get a group of users who don't know each other to use collaboration at a distance because they feel the technology shields them from the other unknown people. Alternatively, users in this situation may prefer to work in a co-present situation so they get to know each other as they are working together. Whether or not the users know each other may impact the way the

users communicate. This relates back to the model of human-human interaction described in Chapter 5.

Question 5 asks "Are there reasons to hide some information from some of the collaborators (each user having a private space, the group has a public space)? If so, why?" The answer to this question will have a direct impact on the technology used for the application. If the activity definition requires that each user have his or her own private space, a single standard computer screen might not be able to fit the entire interface. Furthermore, one screen will not provide each user enough privacy if the activity requires that they "hide" information from one another, such as in a guessing game.

Questions 6 and 7 focus the designer on issues regarding the potentially dynamic nature of groups. Question 6 is a seven-part question asking the designer how and when users can join an activity. It also asks the designer to think about how each user appears on the screen. A user's screen presence could be as simple as a cursor, or as complex as a human-like avatar in a virtual world. Question 7 focuses on what happens in the activity if a users leaves. If the group activity requires a certain number of users and one leaves, the activity may not be able to proceed without something like a computer agent taking over. Unfortunately, the current model definition and toolkit implementation do not take into consideration this dynamic aspect of group work.

6.2.3 The objects

Part III of the worksheet helps to focus the designer on the types of objects presented to the users of the activity. Question 8 simply asks the designer to list some of the objects they foresee in the activity. These objects may be elaborated during brainstorming or storyboarding sessions. The designer is then instructed to answer a series of questions for each of the objects listed.

Question 9 asks, "How do you envision people interacting with these objects?" By

answering this question, the designer should decide if the objects are manipulated through direct means, through other objects or through other agents. This question relates back to Section 4.5.2, which described Cooperatively Controlled Objects (CCOs). CCOs can be implemented in the Cooperative Object-based Applications Program Interface (CO-API) described in Section 6.3.1.

6.2.4 The interactions

The goal of Part IV is to focus the designer on the ways in which the objects are controlled in the activity. As with question 9 in the previous section, this relates to Section 6.3.1 as well as Chapter 4, the model of human-computer interactions. Question 10, in particular, asks the designer to specify the how simultaneously the users should be allowed to interact with an object, as described in Section 4.5.1. The answer to this question will directly influence how many users are allowed to simultaneously manipulate an object in the object hierarchy.

Question 11a), asks, "How is this object controlled or manipulated?" Possible answers are given including the concept that manipulating other objects may change the object. The answer to this question might influence the designer's choice of how simple events combine to create a program event, as described in Section 4.3. The definition of a program event will impact the types of events stored by the system and used in the analysis tools described in Section 6.4, as well as the method(s) used to measure the significance of the interactions, as discussed in Section 4.4.3.

As with some of the previous questions, the answers to questions 11b and 11c might direct the designer's choice of interaction styles. Question 11b asks, "Is this manipulation designed to be fun, instructional, or other?" Question 11c asks, "how difficult will the interaction with this type of object be?" A required simultaneous interaction may be frustrating to a user who is trying to get productive work done, but it might have a different reaction from a user in an entertainment situation.

6.2.5 Evaluation

Part V focuses the designer on the methods that can be used to evaluate the users' collaboration on the task. Question 12 asks the designer five questions, such as, "What might you want to learn about the user's interactions with each other?" and "What type of information do you think would be useful to store about an object?" These questions relate directly to how the interactions can be visualized and analyzed using methods such as the measures of joint activity and significance, as described in the model in Section 4.4. These measures can be used in the analysis tools discussed in Section 6.4.

6.3 The Collaborative Toolkit (Colt)

> *Development tools should minimize the effort it takes to translate the semantics of the interface designed into the semantics of the toolkit offerings.*
>
> -- Baecker, et al [4, p313]

There are many toolkits designed to support computer-based collaboration. These toolkits facilitate the development of cooperative software by providing abstractions for the 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 CoSARA [104], CoLab [98], JAMM [10], Rendevous [86], DistView [88], and GroupKit [34], Habanero [77] support synchronous cooperation over a network. GroupWeb, the collaborative WWW browser mentioned in Section 2.3 was built using GroupKit.

Interfaces and toolkits that support distance synchronous applications are commonly categorized by how much a developer must adapt their software to support multi-user sharing. A *collaboration transparent* toolkit or run-time environment allows multiple users to share an existing single-user application. CoLab and JAMM are examples of such a system. Conversely, CoSara, Rendevous, DistView, GroupKit, and Habanero, are all examples of *collaboration aware* toolkits. Developing a multi-user application with one of these toolkits does require additional specialized code to enable the users to

cooperate. However, these toolkits are designed to make this special code be relatively easy to understand and add.

Another classification of distance synchronous collaborative systems is based on their underlying architecture. In a *Replicated* or *Distributed* system, a copy of the application runs on each client location. The input events from each user go to a centralized server, where they are serialized, then broadcast back out to all of the client applications. The client applications each individually act on the input events, and update the individual displays. JAMM, DistView and CoSARA are examples of replicated systems. In a *Centralized* system, on the other hand, user input arrives from each client to one copy of the application that is run on the server. The application broadcasts the resulting display events back to each of the client locations. Rendevous is an example of a centralized a system. Begole, et al. [10] contains a good pictorial description of replicated and centralized architectures.

The methodology in the previous section aids in the process of designing applications that allow users to manipulate objects with simultaneous cooperative interactions. We have designed and developed *Colt*, the *Collaborative Toolkit* to facilitate the rapid prototyping or implementation of such applications. *Colt* 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 41. (A note on notation, the class diagrams in this section are shown using the UML method. The key to these diagrams is shown in Figure 40.)

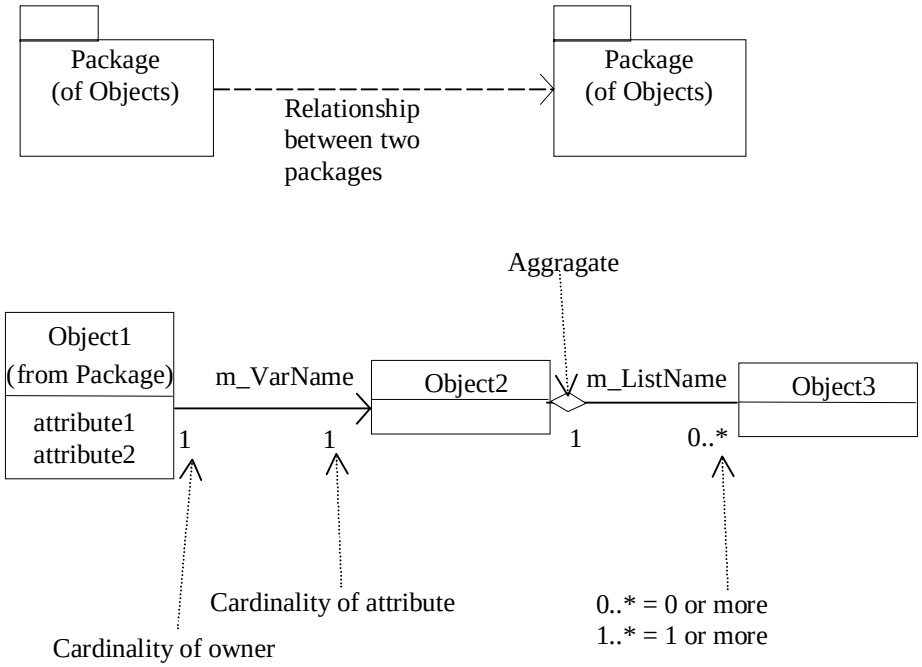

Figure 40. Key to UML diagrams.

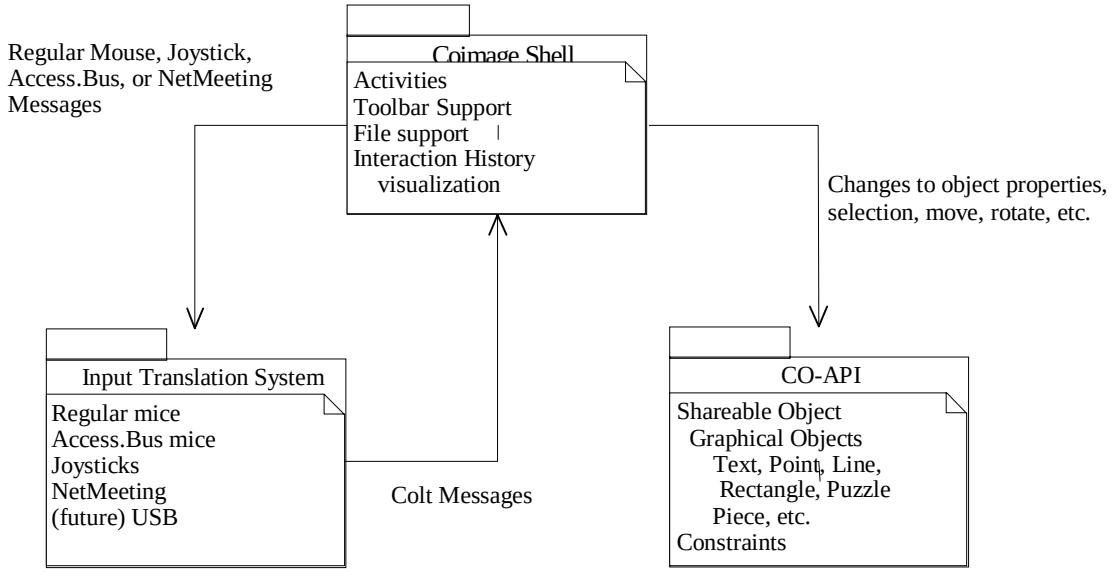

Figure 41. The Colt system architecture.

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 sections 6.3.1, 6.3.2, and 6.3.3, respectively. Section 6.3.4 details the system requirements for Colt.

6.3.1 The CO-API

A high level view of the Collaborative Object-based Application Program Interface, or CO-API, is shown in Figure 42. 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. Each of these parts will be discussed below.

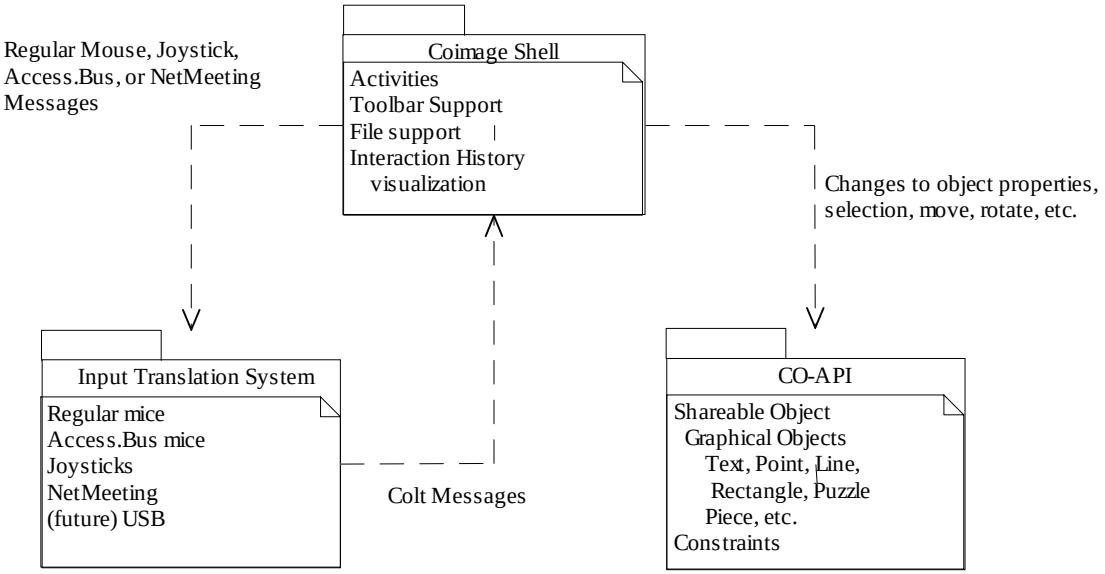

Figure 42. A high level overview of the CO-API.

6.3.1.1 Cooperatively Controlled Objects

6.3.1.1.1 Properties of Cooperatively Controlled Objects

Cooperatively controlled objects are currently 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 [86, 93]. 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 the access control lists). Each object in the CO-API also maintains a list to store the history of actions. The type and information about that action are added to the list. The action information stored depends on how the object is manipulated by the users. This information may be restored at a later time for evaluation purposes.

6.3.1.1.2 Hierarchy of Cooperatively Controlled Objects

Figure 43 shows the CCO hierarchy included in CO-API. The class MObject is the base class from which all CCOs are derived. This class contains methods to manipulate information related to the object's access control as well as serialization. Derived from

the MObject base class is the MGraphicalObject class. This class of object can be displayed on the screen. Other classes, such as MPixmap, MPoint, MLine, MRect and MText are examples of classes derived from MGraphicalObject. Properties of the MGraphicalObject and any subclasses include methods to get or modify the size and location of the object, check for whether or not a mouse point is within that object, and to draw itself.

6.3.1.1.3 User interface design

A number of interface issues must be taken into consideration when software is being written to support two or more people working in a synchronous collaborative situation. In particular, if users are to manipulate a single object simultaneously, each user must have access to his or her own input device. It is possible, if the users are co-present on a single display, to use an off-the-shelf computer system with one mouse and one keyboard, but typically this scenario will cause frustration either from an unintuitive interface or conflict over the input devices. Obvious solutions include having the users interact over a network, or to support multiple input devices on a single machine.

In addition, there should be some indication of which objects each user may access. Furthermore, if a user is currently in control, shared control, possession or shared possession of an object, there should be some evidence to that fact displayed to all users who may need to know. Color typically affords a reasonable sign of object ownership, particularly if the color matches that of the owner’s cursor.

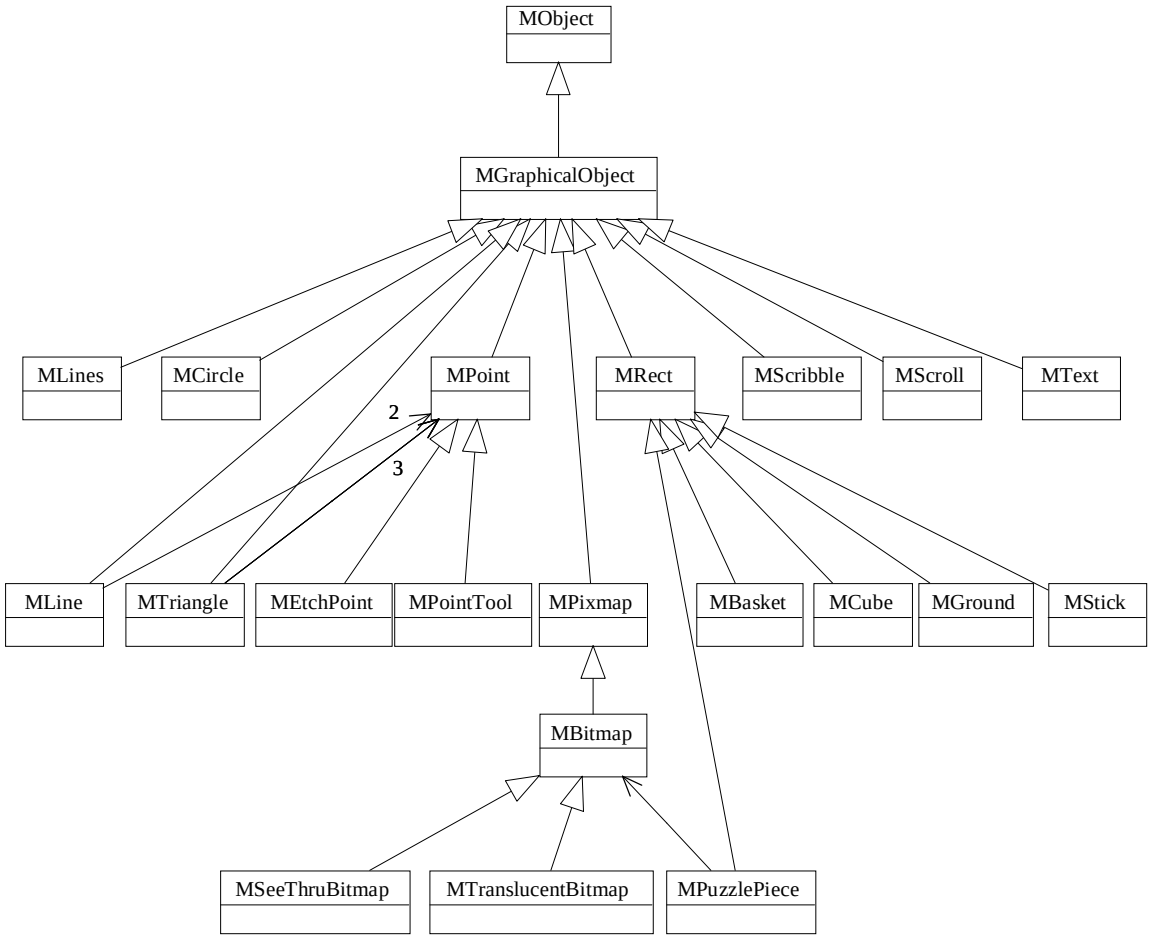

Figure 43. Hierarchy of Cooperatively Controlled Objects in the CO-API.
(See key in Figure 40 for the meaning of the arrows.)

6.3.1.1.4 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) [84]. While the definition of a CCO is not dependent on the location of the users, different interfaces may be required for each situation.

In co-present collaboration, communication is not always verbal. Some forms of communication are based on a shared understanding of actions and gestures. 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. In many co-present activities, however, conflicts can

arise over the use of the physical input devices. One solution to this problem is to provide a separate physical device for each user.

Implementing CCOs for networked cooperation is fundamentally the same as a co-present situation since CCOs respond to changes in object properties and not directly to the events emanating from the input devices. However, the low degree of apparent presence and the lack of actual gaze and contact through the use of current video teleconferencing technology can hamper distance collaboration with other users. Admittedly, as the technology improves and becomes less expensive, the dichotomy between co-present and distance communication will become less clear. In the interim, the appearances of shared objects can be adjusted to make up for some of the communication deficiency of long-distance cooperative control. For example, CCOs could dynamically 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.

6.3.1.1.5 History of user interactions

One focus of our research involves analyzing the communication between people working together on computer activities. We are particularly interested to determine if activities designed with CCOs encourage more frequent communication and what this communication might entail. If we assume that users must communicate to coordinate or problem-solve when manipulating an object, we would expect the communication to be correlated to periods of time when the users are jointly manipulating objects.

In order to track when the users cooperatively control objects, we 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 will typically include the changed property and the new value, which user effected the change, when the change occurred and how long it took. The history of interactions may be saved and analyzed after the activity is completed. The interface of an activity can be designed to include visualizations to aid in 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.

6.3.1.2 The Constraint Manager

In addition to the object hierarchy, the Colt system includes a *Constraint Manager* (shown in Figure 44) 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. To date we have not seen 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.

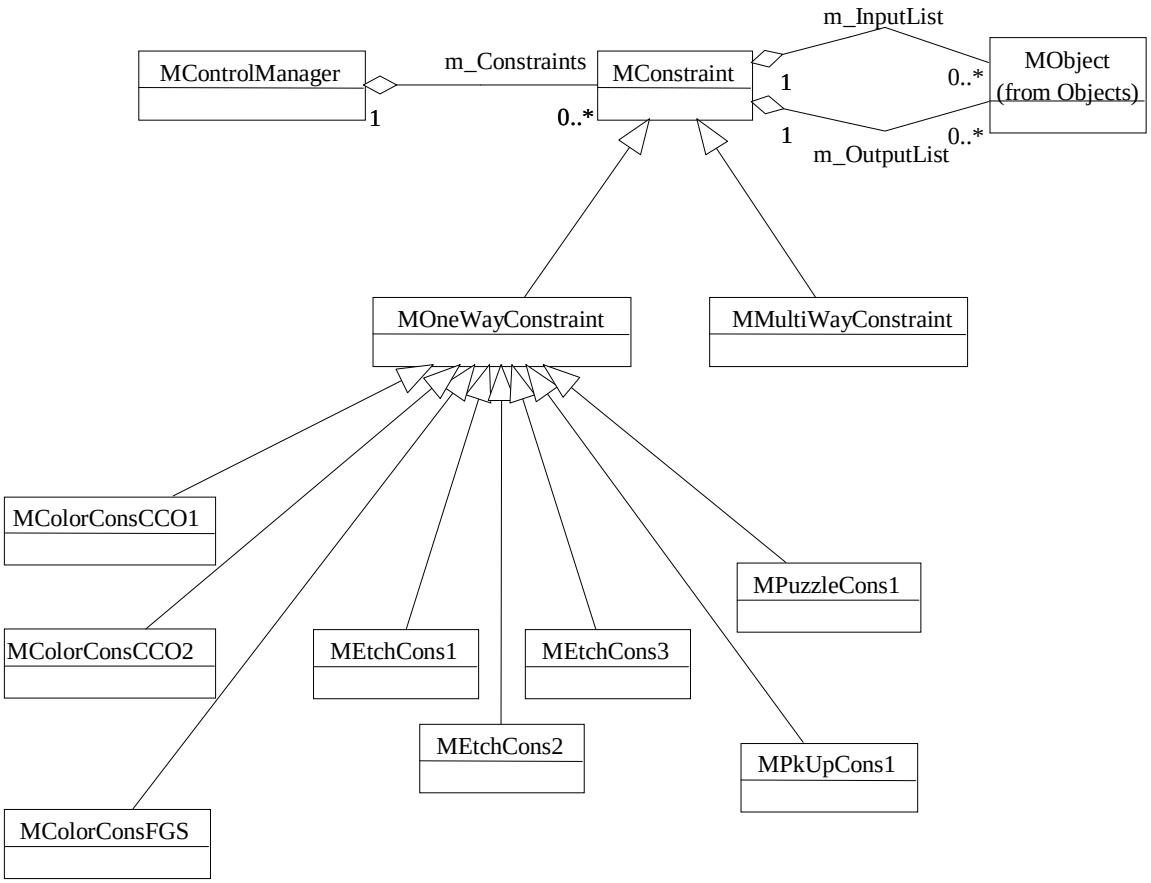

Figure 44. The Constraint Manager and a hierarchy of constraint classes.

6.3.2 The CoImage Shell

We support development of collaborative activities using CCOs by providing an application shell that handles the operations, such as saving and restoring files and handling input from toolbars, which are shared by all activities built in the environment. The hierarchy of objects in the CoImage Shell is shown in Figure 45.

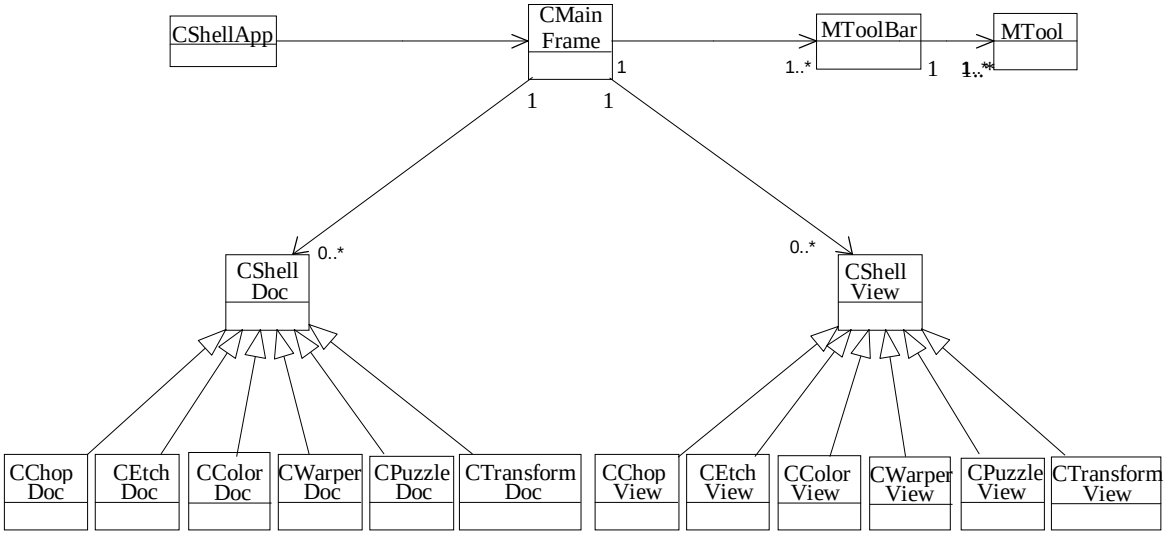

Figure 45. CoImage Shell hierarchy

6.3.2.1 Cursor and toolbar support

As in Bricker, et. al.[13], 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. For Example a tool on the toolbar, shown in Figure 46, is outlined with the user's color if they are permitted to use that tool. If both users may select a tool, it is outlined with both of the users' colors.

Figure 46. The CoImage Shell toolbar.

The design and implementation of the 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 model for the whole activity. Furthermore, depending on the design of the activity, a user's cursor may change shape to indicate that they currently own an instance of the tool just selected (this is true in the Collaborative Image Warping activity described in Section 7.2).

6.3.2.2 Saving and restoring data

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 (with the extension .coi) 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 (.txt) form. This form is to allow 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 (.coi) whenever the text version is written to disk. The CoImage shell can also save in standard Microsoft Windows bitmap (.bmp) format, and in a format that can be read by a World Wide Web (WWW) browser (.htm).

The CoImage shell also includes methods to read and write a text-based problem description file (.prb). Recall that Section 3.2 defined a problem description as including an initial state, goal states or criteria 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 written 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 criteria.

6.3.3 Input Translation Subsystem

In many co-present activities, however, conflicts can arise over the use of the physical input devices. One solution to this problem is to provide a separate physical device for each user. In our experiments with co-present cooperative control, we support Access.Bus multiple mouse system as in [21] as well as Microsoft Sidewinder™ Game Pads [69] and eventually the new Universal Serial Bus standard [106]. In each of these systems, each user has access to his or her own mouse or joystick and the users share a single display. Microsoft NetMeeting™ [65] is a peer to peer conferencing technology.

It was chosen to support distance collaboration in Colt since it 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 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.

6.3.3.1 Implementation details

The architecture for the ITS is shown in Figure 47. ITS is implemented as a set of mouse handler classes, each derived from the class MMouseHandler. A mouse handler is designed to do a number of tasks. First, it translates the device dependent input events into device independent events. The mouse handler also associates a device with a specific user of the activity, this way each device independent event can include information about what user generated the event. Finally, the mouse handler takes care of updating the cursor associated with each device. This is necessary because Microsoft Windows does not support multiple cursors. The cursors in ITS are implemented as a bitmap sprite.

Depending on the target platform, the mouse handler associated with regular mice (MRegularHandler), Access.bus (MAccessHandler), joysticks (MJoyHandler), or the network (MNetHandler) will be instantiated at program started up. Additional initialization is also done at that time. The implementation specific details for the Access.Bus, Joystick and Network handlers will be discussed below. The handler for the regular mouse is a trivial and will not be discussed.

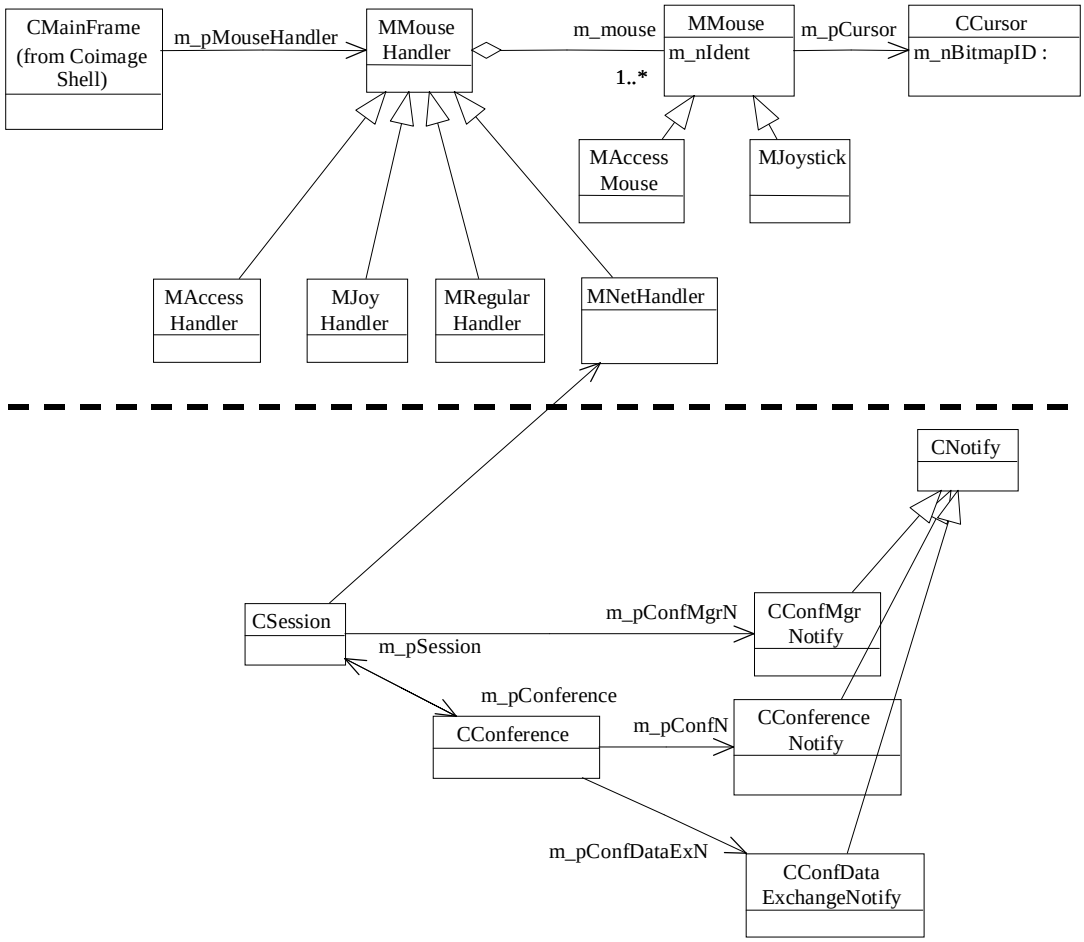

Figure 47. Input Translation Subsystem architecture. The portion below the dotted line is only used in the networked version of the system.

6.3.3.1.1 Access.Bus handler

When the MAccessHandler is initialized, the Access.bus driver is registered with the main application window, a requirement imposed by the implementation of the driver. Unfortunately, this driver was implemented so that the events from the multiple mice are only sent to that one window. The MAccessHandler must not only translate these events into the Colt messages, but it must also determine (if possible) to which child or popup window the new event must be delivered.

6.3.3.1.2 Network handler

When the MNetHandler is initialized, a NetMeeting session is started but the actual

passing of messages between clients does not occur until a conference is started between the users. In this implementation of the system we assume that all conference members join at the same time. A future enhancement would be to allow users to dynamically join and leave an existing conference.

NetMeeting conferences are implemented through peer to peer broadcasting (although a centralized server may be used to locate other participants for a conference). However, when a user attempts to select or modify an object, the application must check with a centralized location to determine that the object is not already being manipulated by another user, and then whether or not the user may have access to that object. In order to support this determination, we specify one user's application as the "server" application (admittedly this was done to avoid having to write a separate server application). The mouse events are sent the network handler, which, in turn, translates it into a Colt message and sends it off to the server. (Obviously there will be some advantage if the application generating the mouse event is the server.) The server application verifies that the operation is valid, and if so, it will rebroadcast the message to all other sites. Each application then updates its state based on the events it receives. This implementation is classified as a collaboration-aware, replicated architecture, as defined in Section 6.1 and by Begole, et al [10].

The networked version of Colt could have been implemented in a number of different ways. This implementation was chosen based on the state of technology at the time the project was begun. For example, had Colt been implemented using Java, we could have used Habanero or JAMM to handle the network collaboration with less work. However, using these systems would not afford us the automatic use of an audio/visual connection.

6.3.3.1.3 Joystick handler

Unlike the Access.bus and network input options, the joysticks are not interrupt-driven devices. In other words, the joysticks will not send events to the application window or its children when the user changes the state of the device. Alternatively, the joystick is polled at a specified time interval to determine its current state. This timer is started in the

main window when the joystick handler is initialized. Each joystick is polled in order depending on the joystick ID

6.3.3.2 Implementation issues

There are a number of implementation issues with the ITS. First and foremost, many child or popup windows, such as dialog boxes, buttons or menus any child windows, are not designed to respond to specialized messages either from Colt or the devices themselves. These child windows receive mouse based input messages directly, bypassing the main window, and change them into a menu or button command message. Furthermore, the mouse handler is never called to translate these messages, and cannot associate the command with the user who generated it. Even if the handler was able to translate these messages, Windows command messages do not contain enough additional storage to include which user has generated the command.

One solution might be to re-implement all Windows controls used by Colt (or the activities developed with Colt) to respond to these specialized events. However, this is time consuming and not particularly extensible. Another time consuming option might be to implement a low-level driver to handle the input translation. Both of these options are potential areas for future enhancements to the system.

There is an additional constraint on handling input from the joysticks. As stated in the previous section, the joysticks are not interrupt-driven devices and are polled for their state at each timer interval in an order based on the joystick IDs. If two users press a button to select an object at exactly the same time, the handler will always send a message from the joystick with the lower ID first. This could potentially block the user with a higher joystick ID from producing input and gaining control of objects on the screen.

Although it would seem that this might also be an issue with the Access.Bus handler, hardware requirements for the input devices and a scheduler in the bus microcontroller prevent it from occurring. The interactive devices cannot control the bus for more than a

specified amount of time and they must give up control of the bus for another specified amount of time.

The implementation of the sprite-based cursors is another area with problems. As with standard sprites, a copy of the background under the cursor is saved before the cursor is drawn on screen. When the cursor is moved, the saved background bitmap is replaced. This works fine when the cursor is simply being moved across the screen. However, if an object is being dragged with the cursor, part of the object gets saved as part of the background bitmap. This leaves "mouse trails" on the screen (see Figure 48). This would not happen if the cursors were implemented as sprites superimposed on top of the frame-buffer [33, p. 180], or if the frame buffer allowed for objects to have different priorities on the screen [33, p. 1065].

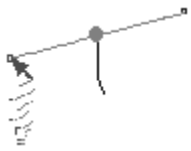

Figure 48. The appearance of "Mouse Trails" while moving the endpoint of a line segment.

6.3.4 System Requirements

The CO-API library, CoImage shell and existing activities have been developed on Pentium based PC platforms running under Microsoft Windows 3.1 and Windows 95. For the co-present, single display situation, we currently support the Access.Bus multiple input system from Computer Access Technology Corporation [21] for Windows 3.1 and the Microsoft Sidewinder™ Game Pads [69] for Windows 95. The CoImage Shell supports distance collaboration through the use of Microsoft NetMeeting™ [65] conferencing technology.

6.4 Visualization and analysis tools

The Colt system also provides a visualization of the history of actions stored by each CO-

API object used in the activity. It 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 user produced the event, and the brightness of the bars depicts how significant the interaction was (as defined in Section 4.4.3). The less significant an interaction was, the darker it will appear in the visualization.

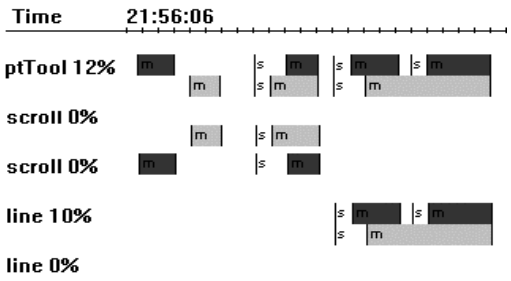

Figure 49. A visualization of the *history of actions* in the drawing activity. A bar with a *s* indicates the user selected an object, a bar with a *m* shows the user moved that object.

Figure 49 shows an example of a visualization from the "drawing" activity described in Section 7.3. 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.

Colt also includes an analysis tool for calculating the measure of joint activity (JA), as

described in Section 4.4.2. This JA analysis tool can be used while viewing the event spans for an activity and has a very simple interface that takes an input file, an output file and the object to analyze in the activity. The input file is text based and contains a list of time periods for each condition (serial, conjoined, or coactive as defined in section 5.3.3.1). The tool produces the output file also as text. The output file contains, for each condition, the total activity for that object (Equation 8), the total activity for the subgroups of each possible size (Equation 11), and the percentage of time that the group manipulated the object as individuals, in subgroups, or all together (Equation 12). The current version of this analysis tool does not include weighting the JA based on the significance of an interaction, as described in 4.4.4.

6.5 Summary

This chapter has presented a system that aids in the iterative design of collaborative activities that allow the users to work simultaneously together. This system includes three parts; a design methodology, a toolkit, and data visualization and analysis tools. The goal of the design methodology is to help focus application designers on the many aspect involved with developing a collaborative activity. The toolkit facilitates the rapid development of collaborative applications that can contain cooperatively controlled objects (CCOs), objects that encourage or require users to simultaneously interact. The visualization and analysis tools can be used to gauge how well the implementation matches the original goals of the design. A teacher or researcher can also use these tools to determine how well the users are collaborating.

CHAPTER 7: ACTIVITIES DEVELOPED WITH COLT

7.1 Prologue

To date, we have developed a number of activities that employ CCOs using the Colt system: the CoImage Warper, a "drawing" program, a jigsaw puzzle, a cooperative "chopstick" activity, and a new version of the Color Matcher. Each of these activities will be presented below, along with an analysis of the activity using the model presented in Chapter 3.

As stated in Section 6.3.2.1, the activities described below allow each user to own an input device and corresponding colored cursor on the screen.

7.2 The Collaborative Image Warper

The first activity developed in the Colt system was based on the Image Warper by the Mathematics Experiences through Image Processing (METIP) project [63]. Figure 50 shows the interface for the *Collaborative Image Warping* activity in which two users may cooperatively control the lines, through their endpoints, and image transformations.

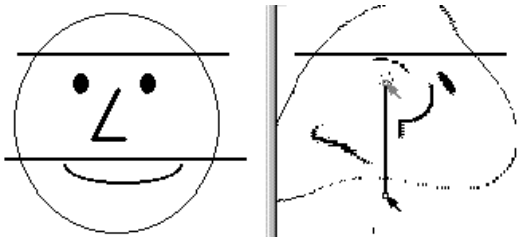

Figure 50. Collaboratively warping an image.

In this activity, either user may choose to draw the warping-control line segments by selecting the appropriate drawing tool. The lines are drawn individually, but once completed, one or both users may alter them simultaneously. Lines are translated, by

clicking and dragging on the middle of the segment, or rotated, by manipulating the location of the endpoints. Depending on the tool a user selects from the activity toolbar, the cursor will change shape to indicate whether a line is being drawn or manipulated. When all the lines are in place, one user may begin the actual warping of the image. The image warping transformation itself is a CCO, manipulated by the two users changing the number and locations of the warping control lines.

7.3 A Collaborative "Drawing" Activity

Figure 51. Etch-a-Sketch™ from Ohio Art

The collaborative Etch-a-Sketch™ is one of the activities implemented using Colt. It is based on the toy developed by Ohio Arts, shown in Figure 51. The toolbar for this activity is shown in Figure 52. Here, the *pen drawing state* tool (on the bottom toolbar) sets the state of the drawing pen: the pen tool on the left allows the pen to leave a trail, while the pen tool on the right stops the trail from drawing. *The clear screen* tool modifies the state of the entire window, by erasing the trail left by the pen.

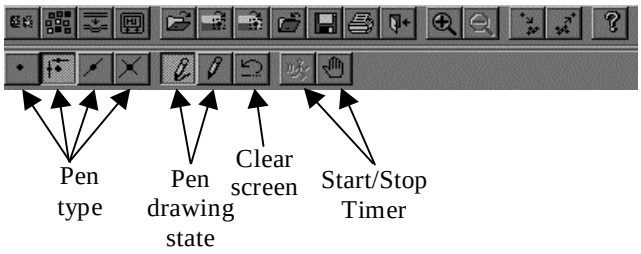

Figure 52. The toolbar for the collaborative Etch-A-Sketch

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, 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 goal of the more complex activities is to trace the background object or solve the maze in the minimum amount of time.

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 53.

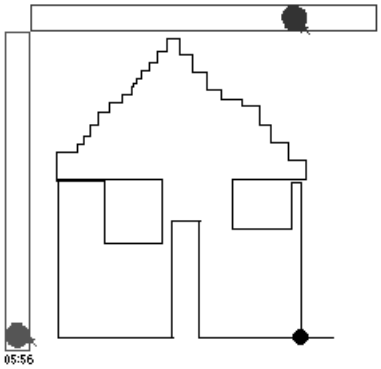

Figure 53. A Fine-grained sharing implementation of the pen used to draw a house

Figure 54 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.

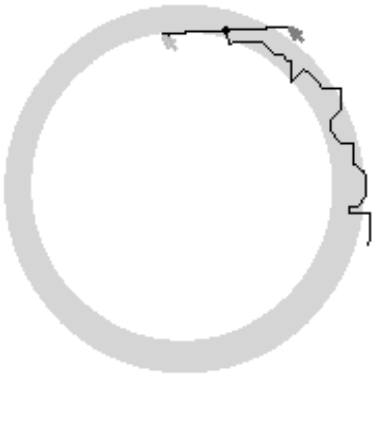

Figure 54. Two users manipulating a cooperatively controlled pen.

The most difficult control, shown being used to solve a maze in Figure 55, requires four users to manipulate the pen. The location of the drawing nib is defined as the intersection of two lines (also shown in Figure 22). 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.

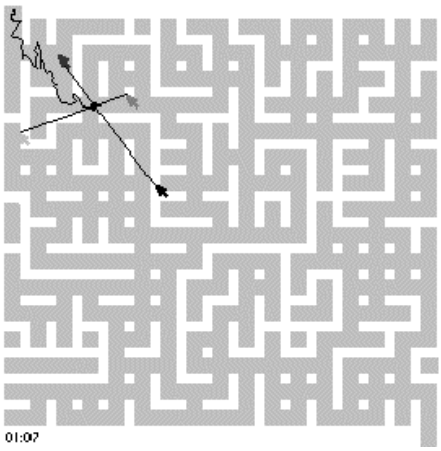

Figure 55. Four users manipulating a cooperatively controlled pen to solve a maze.

This activity could also be modified to simulate a “labyrinth.” A labyrinth activity is one in which a marble is placed in a maze on a platform. The maze contains holes at various

locations. The goal is to get the marble through the maze without falling through a hole and down underneath the platform. The marble is controlled using two knobs, one that rotates the platform forward and back, the other which rotates the platform side to side. The knobs, therefore, not only control the position of the marble, but they also control its velocity. If a knob is turned slightly, the ball will move very slowly in that direction. The more the knob is turned the more the platform leans and the faster the ball rolls. The labyrinth activity could be developed by modifying the FGS drawing activity (shown in Figure 53) to have the scroll bars control the velocity in the x and y directions, rather than x and y positions, and presenting the users with a maze, such as the one in Figure 55.

7.4 The Collaborative Puzzle Activity

The collaborative puzzle is an activity implemented by a student using Colt [7]. In this activity, 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 56, 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.

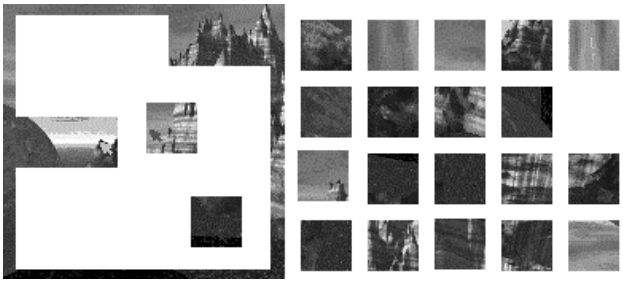

Figure 56. Users working in parallel on the Puzzle activity.

In the *CCO* version of the activity, shown in Figure 57, 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.

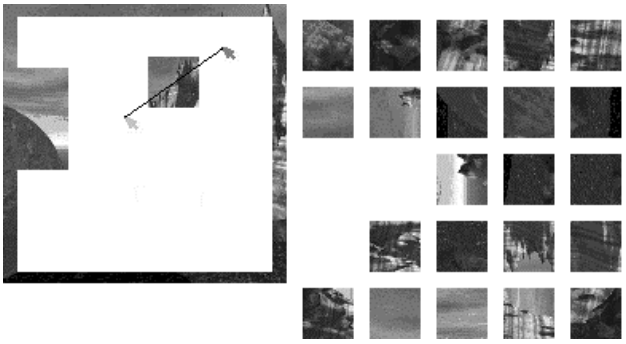

Figure 57. Users manipulating a cooperatively controlled puzzle piece.

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 [7].

7.4.1 User feedback

The graduate student who authored the Puzzle activity (Marla Baker) was the first user of the Colt system. Her comments included:

> *[The system] made my job a lot easier in a number of ways. First, it hid a lot of the details that are necessary to a multiple user system, including displaying multiple cursors on the screen, the mapping of users to cursors, and creating intermediate messages for event handlers that contained pertinent information like which user caused an event to fire. It also took care of a number of other conceptually difficult tasks like managing the multiple layers of graphical objects that must be drawn to the screen in a particular order. [7]*

The user also noted that she had some difficulties including that she had to learn Colt and Microsoft Visual C++ and the Microsoft Foundation class hierarchy. Her only negative

comment about using the system 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. The later problem was latter rectified.

As part of her project, Marla ran a pilot study to examine how users cooperate during the puzzle activity. Her observations, described in more detail in [7], suggest that her cooperatively controlled object encourages users to work together to solve the problem posed by the activity.

7.5 An Exercise in Coordination: The Chopstick Activity

The Chopstick game was designed and developed by an undergraduate at the University of Washington (Emi Fujioka) as a way to test the Colt System. This activity is based on a game where two people each use a chopstick to pick up a “bean.” Although there may be little educational value in an activity such as this, the users must coordinate tightly in order to be successful at the game.

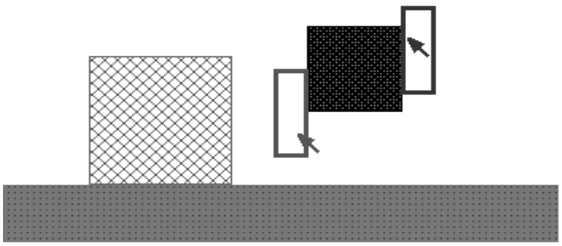

Figure 58. Two users manipulating the large square "bean" in the Chopstick activity.

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 58 shows the simplest version of this activity in action.

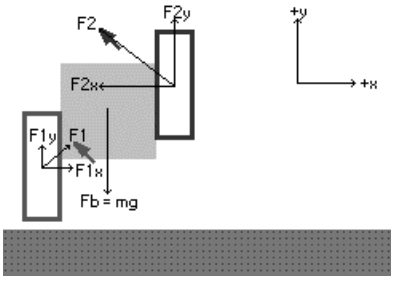

Figure 59. Adding visualization of forces on the bean.

We have considered making the chopstick activity into an exercise in Newtonian mechanics by adding a visualization of the forces imparted on the bean by each stick, as shown in Figure 59. We describe this to indicate the educational potential of activities like these. In this visualization, the force exerted by each user is displayed as a vector from the centroid of their chopstick to the current location of their cursor. The force vectors *F1* and *F2* are broken down into their component parts, (*F1x, F1y*) and (*F2x, F2y*), respectively. The weight of the bean (force of the bean pulling downward) is shown as *Fb*. Ignoring friction between the bean and the chopsticks, and the forces due to the rotation of the chopsticks, the bean will lift when $F1y + F2y + Fb > 0$. The bean moves to the right or the left depending on the sign of value $(F1x + F2x)$. Color coding the force vectors depending on which vector is imparting the most force at any given time could also enhance this visualization.

7.5.1 User feedback

The design and development of Emi's Chopstick activity was begun after Colt had become more stable than for Marla's project and Appendix D had been written as a user manual. 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 multi-way constraints, but since the system is not yet supporting them, she had to use

other methods to simulate this. She did notice that it is impossible to get the information for both of the sticks simultaneously and that the device events arrive at the application serially. Thus, she had to implement some tricks to permit users to move the ball with the sticks.

Her only confusion was in how the list of graphical objects is displayed on the screen using an image/imageView pair. This part of the system is a relic from a previous implementation, and will be removed from the system in a later release

7.6 The Color Matcher: Colt style

The Color Matcher activity, as described in Section 2.5.1, was originally implemented in Microsoft's Visual Basic with the MultiIn system to support the Access.Bus multiple input devices. Unfortunately, MultiIn can only be used under Windows 3.1, and our testing required a version of the application that could be used under Windows 95. Whereas the original version of the Color Matcher activity took an undergraduate student a quarter to implement, the Colt re-implementation took only a few days, and included adding a new CCO method for selecting the users' color.

The functionality of the MultiIn version of the Color Matcher (Figure 60) and the Colt/fine-grained shared (FGS) version (Figure 61) are the same, and the interfaces do look similar, although the check button is now located on the toolbar, and the players are not specifically named. The CCO version of the activity is shown in Figure 62. In this version, 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. The vertices of the triangle are each controlled by one of the users. 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.

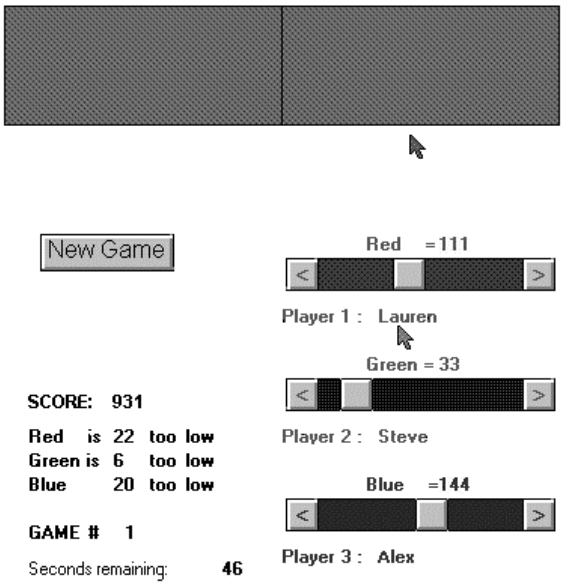

Figure 60. The MultiIn Version of the Color Matcher Activity.

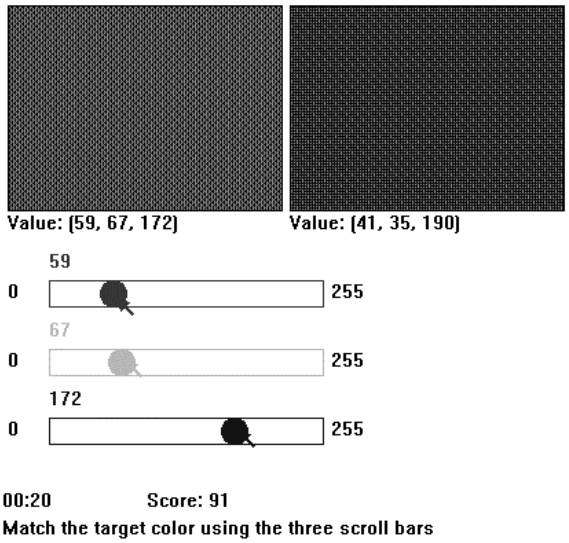

Figure 61. The Colt/FGS version of the Color Matcher Activity.

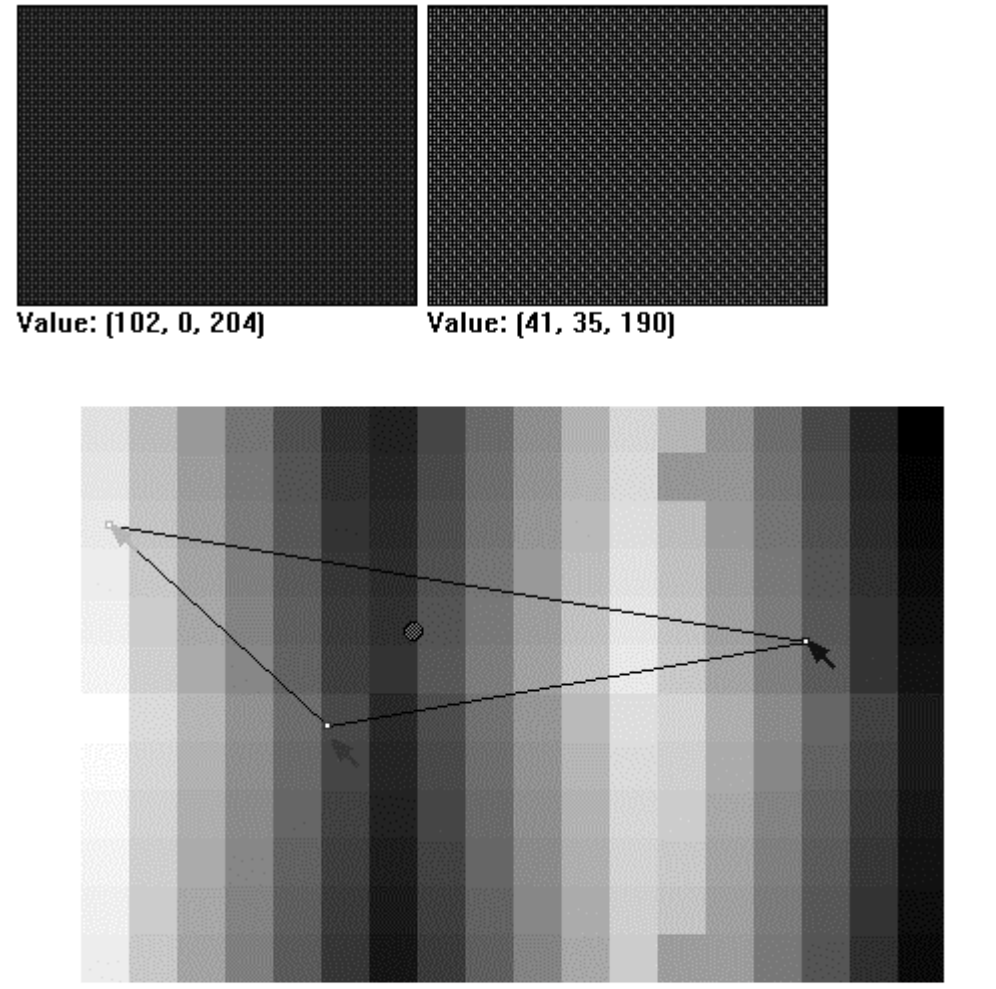

Figure 62. The Colt/CCO version of the Color Matcher Activity.

CHAPTER 8: FUTURE WORK

8.1 Enhancements to the Colt toolkit

8.1.1 The constraint manager

The CO-API should continue to be expanded to support other types of collaborative interactions and activities. The current implementation of the Constraint Manager (described in Section 6.3.1.2) is simple; it handles only one-way, equational constraints. The Constraint Manager also assumes that there are no cyclical dependencies between the constraints. As more complicated cooperative controls are designed, it may be necessary to enhance the Constraint Manager to handle cyclical dependencies, constraint hierarchies consisting of both required and preferential constraints, and inequality constraints. Cassowary is an incremental constraint solving system that would solve all of these problems [13]. It could be added as an underlying system for solving the constraints in the Constraint Manager, or could be used to replace the Constraint Manager all together.

8.1.2 Input translation subsystem

Colt currently supports the CATC Access.Bus, Microsoft SideWinder Game Pads and Microsoft NetMeeting for multiple-user input. The Universal Serial Bus (USB) is a new standard currently being developed by a consortium, with the goal of providing an inexpensive method for multiple serial input, including multiple mice. As the USB standard gains acceptance and is supported, we intend to modify the Input Translation subsystem of the CoImage Shell to use it.

In principle, the Input Translation subsystem of the CoImage Shell should be able to support co-present and distance users simultaneously. This would allow small groups of

users working on one machine to interact with other small groups of users across a network.

8.1.3 Supporting dynamic changes in a group's structure

As stated in Section 6.2, the current implementation of Colt does not allow users to dynamically join or leave a collaborative activity. All users might not be present during the entire course of the activity, particularly when the group is collaborating at a distance. The CoImage Shell should be enhanced to include a system whereby a designer could join a group after an activity started, or leave during the course of an activity. This system may also allow the users to identify themselves, pick the group they wish to join, and could include a repository of user information for those who have already used the system. The repository would allow the users to quickly identify themselves when they wish to join the activity.

8.1.4 Transparent use through the World Wide Web

The Colt system is currently implemented as C++ objects in a stand-alone application shell for Microsoft Windows. If Colt were re-implemented as Microsoft ActiveX or Java objects, activities created with this system would be accessible to more users transparently through the World Wide Web (WWW). Additionally, an activity designer could create a WWW document that includes educational materials or instructions on how to use their activity.

Since ActiveX objects are implemented in C++, this should be the first step to making the Colt system available through the WWW. However, ActiveX objects can only be used in Microsoft Internet Explorer, which is currently only available for Microsoft Windows 95. While it would take more time and resources to reimplement Colt in Java, it would give more users access to highly collaborative activities developed in this system through Netscape Navigator, which is available on Unix and Apple Macintosh systems, as well.

8.1.5 Enhancements to the graphical user interface

Although not critical to the use of the Colt system, we would like to allow users to specify their preferences about the color and, perhaps, shape of their screen cursor. We feel this would give the users more of a sense of ownership and presence when interacting during an activity. We also do not currently have a method for allowing users to join or leave during the middle of an activity. This would be particularly useful in networked CoImage applications.

8.1.6 Saving and restoring data

There are two major areas where saving and restoring data from an activity could be refined. The first improvement would aid the users of a collaborative activity developed with Colt; the second would aid an experimenter examining data saved during a user study.

To a user, viewing an existing binary data file saved from a previous encounter with an activity is a two-step process. The user must begin by starting an instance of the activity that was used, then must open up the file containing the saved data. In the future we would like to store additional information in a data file that informs the application which activity to start automatically.

As stated in Section 6.3.2.2, when the history information is written to disk, Colt saves the data as two different files, a tab delimited text file that is easy for an experimenter to read or use in a spreadsheet program like Microsoft Excel, and a binary data file. However, in order to visualize the data that is written in text form, the experimenter must open the binary data file. Another enhancement to the CoImage shell would be to give the developer methods that support reading and visualizing a text based history file that has been annotated with specific types of codes by the experimenter. An example of how this might be useful is the following situation. During a user study, an observer records the types of communication events by the users, as well as the times that they occurred. These events could be added to the history file as “communication event spans” in the

same format as the program event spans. The history file could then be read back into the system, and visualized using the current event spans view. This would allow the experimenter to visually compare when the users are communicating with when they are interacting with the computer.

8.1.7 Enhancements to the analysis tools

The existing Colt programming environment contains tools for the analysis and visualization of event spans collected by the application as it is progressing. There are two ways in which these tools should be enhanced; one is to add the weighted measure of joint activity (JA), and the other would fix a problem with the event span visualization.

The methods for calculating and storing the significance of an interaction (described in Section 4.4.3) are currently employed by some of CoImage activities, but are not general enough to be employed by every activity. It is also not used to weight the JA, as described in section 4.4.4, and thus is not presented in the JA analysis tool, as described in Section 6.4. We did, however, evidence the need for the weighted JA during the user study. During one group's coactive (required simultaneous) interaction, one user commented that it was possible for all of the users to be active on their portion of the cooperatively controlled triangle (by pressing their control button), but that only one user really had to move. In this way two users could deceive the computer program by feigning activation while the third did the bulk of the work.

Currently viewing the event span visualization for an activity is a three-step process. The user must begin by starting an instance of the activity that was used, then must open up the file containing the saved data, and then must change to the event spans view of that data. In the future we would like to be able to open an "event spans only" view which does not require knowledge of which activity was used to generate the data. Furthermore, the data that is read is must be as in the binary data file format. As stated in the previous section, it would be nice if we could add specific visualization to the event span only view, such as communication event spans.

8.2 Implementing additional collaborative activities

Creating new collaborative applications in the Colt system requires knowledge of object oriented programming techniques, and how to implement Microsoft Windows programs using Microsoft Visual C++. An application programmer must modify the basic CoImage shell document and view objects that are part of the code.

Including a "Wizard" with the Colt system would greatly help the process of implementing a collaborative application. A *Wizard* is a tool to aid users with a complex task and is designed to hide the intricacies of the task from the user. Typically a wizard will ask a series of questions and will create an object based on the specifications given in the answers. A Colt wizard would ask a subset of the questions listed in the Collaborative Activity Design Worksheet, shown in Appendix C. Based on the given responses, the wizard could generate a version of CoImage Shell, CCOs and make file appropriate for the design. The developer would then modify the objects created by the wizard to finish the implementation process.

While a wizard would reduce the amount of work to implement a collaborative application for developers, it is not useful for non-programmers. There are a number of ways in which Colt could be enhanced to support non-technical application designers. A Graphical User Interface (GUI) for Colt might look like Figure 63. A designer would pick from the palette of available CCOs, and drag them into place on the application workspace. Modifying the behavior of objects or linking objects together could be done in a manner similar to the Alternate Reality Kit (ARK) [94]. ARK is a system that allows users to build and modify interactive simulations of physical world objects using a simulated hand and they keyboard to type in some property values. As with ARK, new CCOs would still have to be developed by programmers, but once implemented, an activity designer could modify properties of those objects and use them in their application.

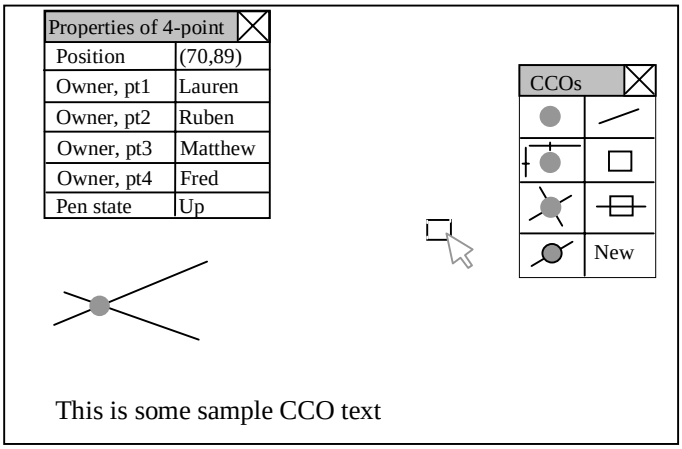

Figure 63. A graphical user interface for Colt.

While these development aids would make Colt easier to use, there are still many collaborative applications that can be implemented using Colt. Some of these activities could include collaborative versions of popular games such as: Tetris, where one person moves side to side while the other controls rotation, The Incredible Machine, or a multi-user version of the Geometer's Sketchpad.

8.3 Evaluation

8.3.1 Evaluating the Colt development system

The developers who designed and implemented the Puzzle and Chopstick Activities (Section 7.4 and 7.5) were the first to test the Colt system. Additional testing needs to be done with other developers to further improve the design methodology, toolkit and analysis tools described in Chapter 6. Questions that could be explored include:

- Do developers actually want to enforce the control of objects in a collaborative learning situation?
- What difficulties are there in developing multi-user activities that require high levels of interactivity?
- Did the environment aid in designing such activities?

8.3.2 Evaluating activities developed with Colt

The preliminary user testing of the Collaborative Puzzle activity [1] provided some evidence that students do work more closely together when using CCOs in a co-present situation. The second user study with the Color Matcher activity (described in Section 5.3.3) was also promising. This second study was designed to investigate how the user worked and communicated together. However, we encountered two main difficulties in video taping the study, both of which contributed to the fact that we could not discern which user was speaking at any given time. The first problem was the low fidelity of the audio on the videotapes. The communication from excessively quiet users was no doubt missing from the audio transcription. The second problem was that in order to keep the students' identities anonymous, we were not able to videotape their faces during the test. This hampered our ability to collect the communication data in general, but it also hindered us from determining the speaker at any given time or from seeing subtle gestures, such as "off-task" behaviors like looking away from the computer. Additionally, had we been able to see the users, we probably could have solved the first problem.

These problems had an impact on the types of questions we could ask in the study. One of our original goals was see if there was a correlation between the times of user communication (verbal and non-verbal) and the event spans captured by the system. We had also wanted to see if there was a difference in this correlation based on the condition (serial, conjoined, or coactive) in which the users were working. Our plan to accomplish was to create a new kind of general object that contained "communication" events. The communication by a user would be recorded as an event, such as a "problem solving event," with a span of time associated with it. These communication events would be added to text based event history file and read back into the system, as described in Section 8.1.6. Then correlation between the communication events and the program events could be visualized with the event span history, or analyzed with tools written to measure the correlation.

Another problem with the study was that we did not store enough information with each program event to calculate the measure of significance (as described in Section 4.4.3) and adjust the measure of joint activity (JA) as described in Section 4.4.4. It would be interesting to see if there is a relationship between how significant a human-computer interaction is and how much that user contributes to the discussion. For example, if would be interested to see if "off-task" behaviors are correlated with a low significance measure.

Finally, there other hypotheses that we did not get to answer simply because of time. For example, we would expect that the users are the most effective in the conjoined condition. This would be due to the fact that the users enjoyed the condition the most, and that they had the most "problem solving" communication during these trials. This information could be measured from a combination of the scores recorded by the system and the users' responses to the questionnaire. However, an appropriate scoring mechanism would have to be added to the existing software in order to do this. A side result from the user testing was that the scoring mechanism, which worked for the original FGS version of the Color Matcher, confused the users of the CCO version. This scoring mechanism should be changed from the existing measure, which is based on the distance in RGB color space between the users' color and the target color, to a distance measure based on the how many squares the cooperatively controlled point is from the square that best matches the target color.

Finally, both of these studies had relatively sample small sizes, and were limited to undergraduate and graduate students at the University. Further testing needs to be done with a larger number of users, K-12 students for CSCL tasks, both in co-present and distance situations. Additional user testing with these subjects may answer questions such as:

- Do CCOs help or hinder in the learning process?
- Can CCOs involve students who might otherwise be removed from the learning experience?
- Do CCOs help to focus students on a specific task? How can this be

measured?

8.3.3 Evaluating local and distance implementations

One of the goals in designing the Colt system was to support tightly focused, highly collaborative interactions in computer applications. Although manipulating a CCO in the CO-API hierarchy is not dependent on the users' locations, the users' interactions with respect to that object will be undoubtedly depend on their physical presence. Users in a co-present situation will be able to communicate more effectively through verbal and non-verbal communication than users in a distance situation. Additional audio and visual communication software can help to increase the users apparent network presence, thereby increasing the ability to collaborate. However, a system that is slow to respond to control of objects across the network could be distracting to a user in a close interaction. Thus, applications written in Colt should be evaluated to determine the network latency in distance (as opposed to co-present) collaborations, and how this latency effects the user interactions.

CHAPTER 9: CONCLUSIONS

Isn't solving the user's problem the true priority of programming?
-- Kim [61, p. 306]

Much of the work in the field of computer science is devoted to making the process of creating software easier. The research presented in this dissertation also attempts to solve that problem, within the domain of human-computer interactions in collaborative programs. In this sense, we have chosen the "user" in the quote above to mean designer of the application, and we wish to make his or her job easier.

We have developed a model of computer supported collaborative applications that focus on two aspects: the interactions between one or more humans and one or more computers, and the interactions between two or more people. Although the work we present can be used to model general collaborative applications, we have chosen to focus on simultaneous cooperative interactions, particularly when the users are co-present. The motivation for investigating this type of interaction is that it is enjoyable and that it may mimic certain real world situations that require multiple users, such as moving heavy furniture. Furthermore, collaborative learning is regaining support in the classroom and we'd like to harness that energy on the computer.

The model provides a number of metrics to measure human-human or human-computer interactions. These metrics are useful to a designer during the iterative development process, or to teachers or investigators who need to know how each user contributed to the solution of a problem. The measure of joint activity (JA) quantifies the way in which the users work together. The result from the JA is designed to show whether the users work relatively independently, in subgroups or as the whole group together. The measure of significance can describe how well the users are working together, or the quality of their interactions with the computer. This may give the investigator insight

into how well the users communicate outside the computer. We also discussed types of interaction techniques and ways to measure how "cooperatively controlled" an object is in the system. The model also describes categories that can be used to classify the communication between the users, such as off-task, social, teaching and learning, and problem solving. We performed a user study to verify our classification of communication. We also found that the users communicated in a social way statistically more in the coactive condition, when they were required to work simultaneously together than in the conjoined condition, when they were only encouraged to do so. We found that the users tend to work fairly closely together on a task even when not forced to work together or apart by the software. Finally, we found that users were not terribly frustrated by the interface, and that they found the interaction entertaining.

While the model is helpful in understanding the interactions between the users and the computer(s), it may be too specific to aid the designer of a cooperative application. There are many factors that go into designing a collaborative application, not the least of which include the location of the users, how they synchronize their interactions, the type of task they are doing and the constraints on that task. The Colt system was designed to relate the model to the implementation of a collaborative activity. Colt includes a design methodology, a software toolkit, and visualization and analysis tools, each intended to aid with a part of the software life cycle. The methodology supports the design, the toolkit supports the implementation, and the visualization and analysis tools support the testing. The analysis tools can be used to determine if the implementation of the application met the original design criteria as specified using the design worksheet.

Finally, we presented five activities that were implemented using the toolkit. Although the author wrote three of the activities, two were written by other students as a way to test the system. The students did give us positive feedback that the system was easy to use, as well as some constructive criticism that will help us in developing future versions.

BIBLIOGRAPHY

1. Allen, J. Chapter 17: Defining a conversational agent. In *Natural Language Understanding, Second Edition*, pp 541-576. Benjamin/Cummings Publishing Company, Inc., 1995.

2. Baecker, R., Grudin, J., Buxton, W., and Greenberg, S. Chapter 4: Software development contexts. In R. Baecker, J. Grudin, W. Buxton, and S. Greenberg, editors, *Human-Computer Interaction: Toward the Year 2000*, pp 273-279. Morgan Kaufmann Publishers, Inc., 1995.

3. Baecker, R., Grudin, J., Buxton, W., and Greenberg, S. Chapter 2: Design and evaluation. In R. Baecker, J. Grudin, W. Buxton, and S. Greenberg, editors, *Human-Computer Interaction: Toward the Year 2000*, pp 73-91. Morgan Kaufmann Publishers, Inc., 1995.

4. Baecker, R., Grudin, J., Buxton, W., and Greenberg, S. Chapter 4: Development tools. In R. Baecker, J. Grudin, W. Buxton, and S. Greenberg, editors, *Human-Computer Interaction: Toward the Year 2000*, pp 313-321. Morgan Kaufmann Publishers, Inc., 1995.

5. Baecker, R., Grudin, J., Buxton, W., and Greenberg, S. Human information processing. In R. Baecker, J. Grudin, W. Buxton, and S. Greenberg, editors, *Human-Computer Interaction: Toward the Year 2000*, pp 573-586. Morgan Kaufmann Publishers, Inc., 1995.

6. Baecker, R., Grudin, J., Buxton, W., and Greenberg, S. Groupware and computer-supported cooperative work. In R. Baecker, J. Grudin, W. Buxton, and S. Greenberg, editors, *Human-Computer Interaction: Toward the Year 2000*, pp 741-753. Morgan Kaufmann Publishers, Inc., 1995.

7. Baker, M. J., Bricker, L. J., and Tanimoto, S. L. *Cooperative interaction techniques in a computer-supported collaborative learning environment*. University of Washington Technical Report TR97-04-02. April, 1997.

8. Barnsley, M. F., Devaney, R. L., Mandelbrot, B. B., Peitgen, H. -O., Saupe, D., and Voss, R.F. *The Science of Fractal Images*. Springer-Verlag, N.Y., 1988.

9. Baron, S. A control theoretic approach to modelling human supervisory control of dynamic systems. In W. B. Rouse ed., *Advances in Man-Machine Systems Research*, pp. 1-47. JAI Press, Inc., 1984.

10. 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.

11. 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.

12. Black Sun Interactive, Inc. *Black Sun Interactive: Products for building 3-D Interactive Communities*. Available as http://www.blacksun.com/, 1997.

13. Borning, A., Marriott, K., Stuckey, P., and Xiao, Y. Solving linear arithmetic constraints for user interface applications. *Proceedings of UIST '97* (Banff, Canada October 14-17, 1997), ACM Press, N.Y., pp 87-96.

14. 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.

15. Cahour, B. and Salembier, P. Cooperation and cooperator modeling. *Computer Supported Cooperative Work (CSCW)*, 5(2-3):285-297, 1996.

16. Campione, J., Brown, A., and Jay, M. Computers in a community of learners. In E. de Corte, M. C. Linn, H. Mandl, and L. Verschaffel, editors, *Computer-Based Learning Environments and Problem Solving*, pp 163-188. Springer-Verlag, 1992.

17. Clark, H., and Shaefer, E. Contributing to Discourse. *Cognitive Science*, 13:259-294, 1989.

18. Clements, D. and Nastasi, B. The role of social interaction in the development of higher-order thinking in Logo environments. In E. de Court, M. C. Linn, H. Mandl, and L. Verschaffel, editors, *Computer-Based Learning Environments and Problem Solving*, pp 229-247. Springer-Verlag, 1992.

19. Cockburn, A. and Jones, S. Four principles of groupware design. *Interacting with Computers*, 7(2): 195-210, 1995

20. Cockburn, A. and Greenberg, S. Children's collaboration styles in a Newtonian MicroWorld, in *Proceedings of CHI '96 Human Factors in Computing Systems* (Vancouver, BC, April 13-18, 1996), ACM Press, N.Y., pp. 181-182.

21. Computer Access Technology Corporation. *Access.Bus Windows Application Development Kit: User's Manual revision 1.0.* 1993.

22. Corman, S. A model of perceived communication in collective networks. *Human Communication Research* 16(4):582-602, 1990.

23. Davidson, N. Introduction and overview. In N. Davidson, editor, *Cooperative Learning in Mathematics*, pp 1-20. Addison-Wesley, Reading, Massachusetts, 1990.

24. Davis, G. Psychology of Problem Solving, Theory and Practice. Basic Books, Inc, New York, 1973.

25. De-Paoli, F. and Tisato, F. CSDL: a language for cooperative systems design. *IEEE Transactions on Software Engineering*, 20(8): 606-616, 1994.

26. Dockterman, D. Cooperative learning and computers. *SIGQUE Outlook*, 21(2):39-43, 1991.

27. Dori, D., Alon, M. and Dori, Y. Team training shell: a groupware, multimedia-supported application generator, in *Proceedings of Ed Media '94* (Vancouver, BC, June 25-30, 1994), ACM Press, N.Y., pp 166-171.

28. Edelson, D., Pea, R., and Gomez, L. The collaboratory notebook. In *Communications of the ACM*, 39(4):32-33, 1996.

29. Elwart-Keys, M., Halonen, D., Horton, M., Kass, R., and Scott, P. User interface requirements for face to face groupware, in *Proceedings of CHI '90 Human Factors in Computing Systems* (Seattle, April 1-5, 1990), ACM Press, N.Y., pp 295-301.

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

31. Ellis, C. and Wainer, J. A conceptual model of groupware, in *Proceedings of CSCW '94* (Chapel Hill, October 22-26, 1994), ACM Press, N.Y., pp. 79-88.

32. Fishman, B. and D'Amico, L. Which way will the wind blow? Networked computer tools for studying the weather, in *Proceedings of Ed Media '94* (Vancouver, BC, June 25-30, 1994), ACM Press, N.Y., pp 203-208.

33. Foley, J., vanDam, A., Feiner, S., and Hughes, J. *Computer Graphics: Principles and Practice*, Addison-Wesley Publishing Co.,

34. Freeman-Benson, B., Maloney, J., and Borning, A. An incremental constraint solver. *Communications of the ACM*, 33(1):54-63, 1990.

35. Freeman-Benson, B. Converting an existing user interface to use constraints. *Proceedings of UIST '93* (Atlanta, November 3-5, 1993), ACM Press, N.Y., pp 207-215.

36. Gaver, W., Moran, T., MacLean, A., Lovstrand, L., Dourish, P., Carter, K., and Buxton, W. Realizing a video environment: EuroPARC's RAVE system, in *Proceedings of CHI '92 Human Factors in Computing Systems* (Portland, September 26-29, 1992), ACM Press, N.Y., pp 27-35.

37. Gould, J. and Lewis, C. Deisgning for usability: key principles and what designers think. *Communications of the ACM*, 28(3):300-311, 1985.

38. Greenberg, S. and Roseman, M. GroupWeb: A WWW Browser as Real Time Groupware, in *Proceedings of CHI '96 Human Factors in Computing Systems* (Vancouver, BC, April 13-18, 1996), ACM Press, N.Y., pp 271-272.

39. Grosz, B. and Kraus, S. Collaborative plans for group activities, in *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence* (Chambery, France, August 28 – September 3, 1993), Morgan Kaufmann Publishers, San Mateo, pp 367-373.

40. Grudin. J. Why CSCW applications fail: problems in the design and evaluation of organizational interfaces, in *Proceedings of the CSCW '88* (Portland, September 26-29, 1988), ACM Press, N.Y., pp 85-93.

41. Grudin, J. Groupware and cooperative work: problems and prospects, in B. Laurel, editor, *The Art of Human-Computer Interface Design*, pp171-185. Addison-Wesley, 1990.

42. Grudin, J. Groupware and social dynamics: eight challenges for developers. *Communications of the ACM*, 37(1):93-105, 1994.

43. Gugerty, L. The use of analytical models in human-computer-interface design. *International Journal of Man-Machine Studies*, 38(4):625-660, 1993.

44. 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.

45. Hayes, J. The Complete Problem Solver, Second Edition. Lawrence Erlbaum Associates, Hillsdale, NJ, 1980.

46. Heath, C. and Luff, P. Disembodied conduct: communication through video in a multi-media office environment, in *Proceedings of CHI '91 Human Factors in Computing Systems* (New Orleans, LA, April 27 – May 2, 1991), ACM Press, N.Y., pp 99-103.

47. Harel, D. Statecharts: a visual formalism for complex systems. *Science of Computer Programming*. 8(3):231-74, 1987.

48. Hollander, M., and Wolfe, D. *Nonparametric Statistical Methods*. John Wiley and Sons, New York, 1973.

49. Hudson, S. A system for efficient and flexible one-way constraint evaluation in C++. Georgia Institute of Technology technical report, GIT-GVU-93-13, 1993.

50. Hruza, Denis. Denis Hruza's Space Duel. http://raven.cybercomm.net/~king/spaceduel.html, 1996.

51. Inkpen, K., Booth, K., Klawe, M., and Upitis, R. *Cooperative Learning In the Classroom: The Importance of a Collaborative Environment for Computer-Based Education*. University of British Columbia Technical Report 94-5. February, 1994.

52. 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.

53. Isaacs, E. and Tang, J. What video can and can't do for collaboration, a case study, in *Proceedings of ACM Multimedia '93* (Anaheim, CA, August 2-6, 1993), pp 199-206.

54. Isaacs, E., Morris, T., Rodriguez, T. Tang, J. A comparison of face-to-face and distributed presentations, in *Proceedings of CHI '95 Human Factors in Computing Systems* (Denver, CO, May 7-11, 1995), ACM Press, N.Y., pp 354-361.

55. Ishii, H., Kobayashi, M., and Grundin, J. Integration of inter-personal space and shared workspace: ClearBoard design and experiments, in *Proceedings of the CSCW* '92 (Toronto, October 31 – November 4, 1992), ACM Press, N.Y., pp 33-42.

56. Jacobs, S. Language, in M. Knapp and G. Miller, editors, *Handbook of Interpersonal Communication*, pp 313-343, Sage Publications, 1985.

57. Jehng, J. J., Shih, Y., Liang, S., and Chan, T. TurtleGraph: A computer supported cooperative learning environment, in *Proceedings of ED-MEDIA '94* (Vancouver, BC, June 25-30, 1994), pp 293-298.

58. John, B. and Kieras, D. The GOMS family of user interface analysis techniques: comparison and contrast. *ACM Transactions on Computer-Human Interaction*, 3(4): 320-351, 1996.

59. John, B. and Kieras, D. Using GOMS for user interface design and evaluation: which technique? *ACM Transactions on Computer-Human Interaction*, 3(4): 287-319, 1996.

60. Johnson, R., Johnson, D., Stanne, M. Comparison of computer-assisted cooperative, competitive, and individualistic learning. *American Educational Research Journal*, 23(3): 382-392, 1986/

61. Kim, S. Interdisciplinary cooperation. In R. Baecker, J. Grudin, W. Buxton, and S. Greenberg, editors, *Human-Computer Interaction: Toward the Year 2000*, pp 304-311. Morgan Kaufmann Publishers, Inc., 1995.

62. Lepper, M. and Malone, T. Intrinsic motivation and instructional effectiveness in computer-based education. In R. E. Snow and M. J. Farr, editors, *Volume 3: Conative and Affective Process Analyses*, pp 255-287. Erlbaum, 1987.

63. Loftus, G. and Loftus, E. *Mind at Play: the Psychology of Video Games.* Basic Books, New York, 1983.

64. Mathematics Experiences Through Image Processing. *Mathematics Experiences Through Image Processing (METIP) Project.* Available as http://www.cs.washington.edu/research/metip/metip.html, 1996.

65. McClay, B. and Hollingsead, C. Computers and cooperative learning: a winning combination. *Counterpoint*, 16(4): 8, Summer, 1996.

66. McDaniel, S., Olson, G., and Magee, J. Identifying and analyzing multiple threads in computer mediated and face-to-face conversations, in *Proceedings of CSCW '96* (Boston, MA, November 16-20, 1996), pp 39- 47.

67. Microsoft. *Microsoft Chat.* Available as http://www.microsoft.com/ie/comichat/.

68. Microsoft. *Microsoft NetMeeting™ Complete Internet Conferencing*. Available as http://www.microsoft.com/netmeeting/.

69. Microsoft. *Microsoft SideWinder™ Game Pads.* Available as http://www.microsoft.com/sidewinder/gamepad/default.htm.

70. Mitchell, C. Models for the Design of Human Interaction with Complex Dynamic Systems. Available as http://www.isye.gatech.edu/chmsr/Christine_Mitchell/requirements.html

71. Mitchell, C. Task-Analytic Models of Human Operators: Designing Operator-Machine Interaction. Available as http://www.isye.gatech.edu/chmsr/Christine_Mitchell/modeldesign.html

72. Monteferrante, S. Implementation of Calculus, Concepts and Computers at Dowling College. *Collegiate Microcomputer*, 11(2): 95-98, 1993.

73. Mühlhauser, M., and Rudenbusch, T. Cooperation support in computer-aided authoring and learning, in *Proceedings of ED-MEDIA '94* (Vancouver, BC, June 25-30, 1994), pp 397-402.

74. Mühlhauser, M. Modeling and design of complex cooperative software, in *Proceedings of First IEEE International Conference on Engineering Complex Computer Systems* (Ft. Lauderdale, November 6-10, 1995), pp 274-277.

75. Myers, B. State of the art in user interface software tools. In R. Baecker, J. Grudin, W. Buxton, and S. Greenberg, editors, *Human-Computer Interaction: Toward the Year 2000*, pp 323-343. Morgan Kaufmann Publishers, Inc., 1995.

76. Nastasi, B. and Clements, D. Motivational and social outcomes of cooperative computer education environments. *Journal of Computing in Childhood Education*, 4(1):15-43, 1993.

77. National Center for Supercomputing Applications. NCSA Habanero. http://www.ncsa.uiuc.edu/SDG/Software/Habanero/, June 1996.

78. Norman, D. Cognitive Engineering. In Norman, D. and Draper, S. editors, *User Centered System Design*, pp 31-65, Erlbaum, Hillsdale, NJ, 1987.

79. Olson, A. Transforming an HCI model to a software design model. *International Journal of Software Engineering and Knowledge Engineering*, 7(1):145-167, 1997.

80. O'Neill, D. K., and Gomez, L. M. The collaboratory notebook: a networked knowledge-building environment for project learning, in *Proceedings of ED-MEDIA '94* (Vancouver, BC, June 25-30, 1994), pp 416-423.

81. O'Neill, D. K, Edelson, D. C., Gomez, L. M., and D'Amico, L. Learning to weave collaborative hypermedia into classroom practices, in *Proceedings CSCL '95* (Bloomington, October 17-20, 1995), ACM Press, N.Y., pp 255-258.

82. Orlikowski, W. Learning from COMMENTS: Organizational issues in groupware implementation, in *Proceedings of the CSCW '92* (Toronto, October 31 – November 4, 1992), ACM Press, N.Y., pp 362--369.

83. Oshima, J., Bereiter, C., and Scardamalia M. Information-access characteristics for high conceptual progress in a computer-networked learning, in *Proceedings of CSCL '95* (Bloomington, October 17-20, 1995), ACM Press, N.Y., pp 259-267.

84. Palmer, J. D. and Fields, N. A. Guest Editor's Introduction: Computer Supported Cooperative Work. *IEEE Computer*. 27(5):15-17, 1994.

85. Payne, S. and Green, T. The structure of command languages: an experiment on task-action grammar. *International Journal on Man-Machine Studies*, 30(2):213-234, 1989.

86. Patterson, J. F., Hill, R. D. and Rohall, S. L. Rendezvous: an architecture for synchronous multi-user applications, in *Proceedings of CSCW '90*, (Los Angeles, CA, October 7-10, 1990), ACM Press, N.Y., pp 317-328.

87. Pew, R. and Baron, S. Perspectives on human performance modelling. *Automatica*, 19(6):663-676, 1983.

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

89. Rational Software, Corp. UML Booch & OMT quick reference. Available as http://www.rational.com/uml/qr/.

90. Rich, C. and Sidner, C. Adding a collaborative agent to graphical users interfaces, in *Proceedings of UIST '96*, (Seattle, WA, November 6-8, 1996), ACM Press, N.Y., pp 21-30.

91. Scardamalia, M., and Bereiter, C.. An architecture for collaborative knowledge building. In E. de Corte, M. C. Linn, H. Mandl, and L. Verschaffel, editors, *Computer-Based Learning Environments and Problem Solving*, pp 41-66. Springer-Verlag, 1992.

92. Shaw, A. Software specification languages based on regular expressions. In W.E. Riddle and R. E. Fairley, editors, *Software Development Tools*, pp 148-175. Springer-Verlag, 1980.

93. 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.

94. Smith, R. Experiences with the alternate reality kit. *IEEE Computer Graphics and Applications*, 7(9):42-50, 1987.

95. Smith, R. What you see is what I think you see. *SIGCUE Outlook*, 3(21):18-23, 1992.

96. Smith, R.E., Smoll, F. L. and Hunt, E. A system for the behavioral assessment of athletic coaches. *The Research Quarterly*, 48(2):401-407, 1978.

97. Spitulnik, J., Studer, S., Finkel, E., Gustafson, E., Laczko, J., and Soloway, E. Toward Supporting Learners in Scientifically-Informed Community Discourse, in *Proceedings of CSCL '95* (Bloomington, October 17-20, 1995), ACM Press, pp 317-320.

98. Stefik, G. Foster, D. G. Bobrow, K. Kahn, S. Lanning, and L. Suchman. Beyond the chalkboard: Computer support for collaboration and problem solving in meetings. *Communications of the ACM*, 30(1):32-47, 1987.

99. Stewart, J. Single display groupware, in *Proceedings of CHI '97 Human Factors in Computing Systems* (Atlanta, GA, March 22-27, 1997), ACM Press, N.Y., pp 71-72.

100. Suzuki, H. and Kato, H. Interaction-Level Support for Collaborative Learning: *AlgoBlock*-An Open Programming Language, in *Proceedings of CSCL '95* (Bloomington, October 17-20, 1995), ACM Press, pp 349-355.

101. Syntrillium Software Corporation. *Cool Edit.* Available as http://www.syntrillium.com/10/cool.htm.

102. Tang, J. Findings from observational studies of collaborative work. *International Journal of Man-Machine Studies*, 34(1):143-160, 1991.

103. Taylor, J. Carletta, J. and Mellish, C. Requirements for belief models in cooperative dialog. *User Modeling and User-Adapted Interaction*, 6(1):23-68, 1996.

104. Tou, I., Berson, S., Estrin, G., Eterovic, Y., and Wu, E. Prototyping Synchronous Group Applications. IEEE Computer, 27(5):48-56, 1994.

105. Twidale, M. B., Nichols, D. M., Smith, G., and Trevor, J. Supporting Collaborative Learning During Information Searching, in *Proceedings of CSCL '95* (Bloomington, October 17-20, 1995), ACM Press, pp 367-374.

106. Universal Serial Bus Consortium. *Universal Serial Bus Implementers Forum Home Page*. Available as http://www.usb.org/.

107. Wenburg, J. and Wilmot, W. *The Personal Communication Process.* John Wiley and Sons, Inc., New York, 1973.

108. Wolf, C. and Rhune, J. Communication and information retrieval with a pen-based meeting support tool, in *Proceedings of CSCW '92* (Toronto, October 31-November 4, 1992), ACM Press, N.Y., pp 322-328.

APPENDIX A: USER STUDY PROCEDURE

The following is a description of the procedure for the user study. The information in *italics* indicate instructions for the experimenter. The information in regular text is a script of what was said to the users during the course of the study.

Start the application and the Color Matcher activity. Bring up the identity dialog box with the F5 key, identify the group, and select the starting condition they will start.

In today's experiment you will be using the Color Matcher activity on the computer. In the color matcher program, your goal is to center the displayed triangle over color that best matches a target color. Each of you will use a control pad, there in front of you, to manipulate a cursor on the screen. Please pick up your control pad and use the large button on the left of the controller to move your cursor around the screen. You should be able to determine the color or your cursor, red, green, or blue.

We will be videotaping the computer screen as you work. At most, only your hands may be visible on the videotape. In no way will you be personably identifiable on the videotape. Please pick up the rings corresponding to your cursor color and put one on each hand.

Each of you will be able to use your control pad to manipulate the corner of the triangle that coincides with the color of your cursor. To manipulate your corner move your cursor over your corner of the triangle and then select the corner by pressing and holding the 'A' button. Now that you are holding down the 'A' button you will be able to move your corner of the triangle. Please keep your corner of the triangle within the white area of the screen. You will be considered active whenever you are holding down the 'A' button. In this fashion the three you will be able to manipulate the center of triangle which is identified by the gray dot.

Please practice moving the triangle now.

Say one of the following depending on the condition.

Serial condition: You will notice that at most one of you can be active at any time. This is the serial task.
Conjoined condition: This is the conjoined task.
Coactive condition: You will notice that you must all be active at the same time in order to move the triangle. This is the coactive task.

You will notice that the rectangle in the upper left-hand corner of the screen displays the color that the center of the triangle is over. The target color, which you are trying to match AS BEST YOU CAN, is located to the right of the display of your selected color. You will see a new color for your next trial when you hit the “next” button.

After a practice trial you will attempt three trials before moving on to the next task. You will be given 60 seconds to complete each of these trials AS BEST YOU CAN. The timer is located on the bottom left hand corner of the screen. If your group feels that you are done with the trial before that time then one of you should click on the button on the screen labeled "check." Once you have either run out of time or you have clicked on "check" button, your trial will be over and a score of your performance for the trial will be computed based on how closely your selected color and the target color match. The score is located next to the timer on the bottom left hand corner of the screen.

We do not expect that any part of this experiment will cause stress. If however you would like to quite your participation in this experiment please let the experimenter know immediately.

Do you have any questions?

Answer questions about information above. If the users ask questions about how they coordinate or how they do the task, tell them that they may talk to each other, but do not give any information about how to coordinate the movement. Do not confirm or deny any comments about problem solving methods. So if in the serial condition they ask, "So we should talk about who is going to move their cursor at what time?" we should say something like "You are certainly allowed to talk amongst yourselves to determine how you are going to work on the task". Doesn't tell them to solve the problem in the subject's suggested way, but if they were not clear that they could talk with their fellow subjects we've clarified that.

So that we can distinguish your voices on the video tape, please say the following: "I'm in group X and I'm color Y"

Wait for users to say the above.

Please speak up a bit while working together.

CONDITION START: The condition will now start. Use the following series of sentences for each task.

Now you are going to perform the _____ task. *Fill in the blank with [serial, conjoined, coactive]*

Describe the condition with the appropriate sentence.

Serial condition: In the serial task only one of you will be able to be active with your part of the triangle at any one time. If one of you wishes to make your part active, whoever is currently active must release their A button.
Conjoined condition: In the conjoined task you may all manipulate your part of the triangle either at the same time or at different times.
Coactive condition: In the coactive task you must all activate your part of the triangle at the same time.

Is that clear?

TRIAL START

Here is your practice trial.

Wait for users to perform their practice trial. If the users complete early, remind them that they can use the check button.

Ok, you scored __. *Allow time for group reaction, if needed*

Now that you have had a chance to practice you can move on to your first trial of color matching by clicking on the button on the screen labeled next.

Repeat from TRIAL START for three more trials for a total of four trials.

Repeat from CONDITION START for all the conditions.

When the users are done, SAVE THE HISTORY FILE!

Now we'd like you to fill out a short questionnaire. Please fill out the question privately. Remember, your order was _____, where the conditions meant:

Say the following in the order they completed the conditions.

In the serial task only one of you will be able to be active with your part of the triangle at any one time.

In the conjoined task you may all manipulate your part of the triangle either at the same time or at different times.

In the coactive task you must all activate your part of the triangle at the same time.

Debrief the users when the have completed the questionnaires.

Discuss what we were investigating at and why.

Colors for trials

Trial	Serial	Conjoined	Coactive
1	Green	Blue	Blue
2	Blue	Pink	Orange
3	Dark green	Purple	Red
4	Grey	Green	Green
5	Light Grey	Light pink	Pink

APPENDIX B: USER STUDY QUESTIONNAIRE AND RESULTS

Sex: Male Female

First task of the program: Serial Conjoined Coactive

Second task of the program: Serial Conjoined Coactive

Third task of the program: Serial Conjoined Coactive

1	How many people in your group did you know before today's study?	0	1	2

	Question	Agree Strongly			Neutral			Disagree Strongly
2	I like doing these tasks with others.	1	2	3	4	5	6	7
3	I am comfortable with using computers.	1	2	3	4	5	6	7
4	I think that the tasks we did where boring.	1	2	3	4	5	6	7
5	Our group talked a lot in the serial task of the program.	1	2	3	4	5	6	7
6	Our group talked a lot in the conjoined task of the program.	1	2	3	4	5	6	7
7	Our group talked a lot in the coactive task of the program.	1	2	3	4	5	6	7
8	Given a choice I'd rather use the program by myself.	1	2	3	4	5	6	7
9	I didn't like working with my partners.	1	2	3	4	5	6	7
10	I don't feel comfortable using a computer.	1	2	3	4	5	6	7
11	I feel that in the serial task of the program our group didn't communicate very much.	1	2	3	4	5	6	7
12	I feel that in the coactive task of the program our group didn't communicate very much.	1	2	3	4	5	6	7
13	I feel that in the conjoined task of the program our group didn't communicate very much.	1	2	3	4	5	6	7
14	I feel that the person controlling the blue cursor did a lot of the talking.	1	2	3	4	5	6	7
15	I feel that the person controlling the green cursor did a lot of the talking.	1	2	3	4	5	6	7
16	I feel that the person controlling the red cursor did a lot of the talking.	1	2	3	4	5	6	7
17	I play video (arcade, Nintendo, etc) games a lot.	1	2	3	4	5	6	7
18	I've done something like these tasks before.	1	2	3	4	5	6	7
19	I play computer games a lot.	1	2	3	4	5	6	7
20	I was frustrated by the serial task.	1	2	3	4	5	6	7
21	I was frustrated by the coactive task.	1	2	3	4	5	6	7
22	I was frustrated by the conjoined task.	1	2	3	4	5	6	7

	Question	Agree Strongly			Neutral			Disagree Strongly
23	I think that the person controlling the blue did a lot of the work on the tasks.	1	2	3	4	5	6	7
24	I think that the person controlling the green did a lot of the work on the tasks.	1	2	3	4	5	6	7
25	I think that the person controlling the red did a lot of the work on the tasks.	1	2	3	4	5	6	7
26	I think that the tasks we did were interesting.	1	2	3	4	5	6	7
27	I thought the serial task was a productive way of way of performing the program.	1	2	3	4	5	6	7
28	I thought the coactive task was a productive way of way of performing the program.	1	2	3	4	5	6	7
29	I thought the conjoined task was a productive way of way of performing the program.	1	2	3	4	5	6	7
30	Overall, I thought the interface for these tasks were easy to use.	1	2	3	4	5	6	7
31	I thought the serial task was entertaining.	1	2	3	4	5	6	7
32	I thought the conjoined task was entertaining.	1	2	3	4	5	6	7
33	I thought the coactive task was entertaining.	1	2	3	4	5	6	7
34	I was comfortable using the gamepad controllers in performing the tasks.	1	2	3	4	5	6	7
35	I think that the serial task of the program encouraged our group to communicate a lot.	1	2	3	4	5	6	7
36	I think that the coactive task of the program encouraged our group to communicate a lot.	1	2	3	4	5	6	7
37	I think that the conjoined task of the program encouraged our group to communicate a lot.	1	2	3	4	5	6	7
38	It was easy to do the program in the serial task.	1	2	3	4	5	6	7
39	It was easy to do the program in the coactive task.	1	2	3	4	5	6	7
40	It was easy to do the program in the conjoined task.	1	2	3	4	5	6	7
41	Any questions or comments?							

Table 15. Average responses from the questionnaire (n = 36).

Question	Average	Standard deviation
2. I like doing these tasks with others.	2.42	1.57
3. I am comfortable with using computers.	2.69	1.39
4. I think that the tasks we did where boring.	5.42	1.18
5. Our group talked a lot in the serial task of the program.	3.28	1.94
6. Our group talked a lot in the conjoined task of the program.	3.25	2.03
7. Our group talked a lot in the coactive task of the program.	3.31	2.01
8. Given a choice I'd rather use the program by myself.	4.58	1.84
9. I didn't like working with my partners.	5.97	1.29
10. I don't feel comfortable using a computer.	5.50	1.58
11. I feel that in the serial task of the program our group didn't communicate very much.	4.56	1.89
12. I feel that in the coactive task of the program our group didn't communicate very much.	4.72	2.09
13. I feel that in the conjoined task of the program our group didn't communicate very much.	4.89	1.91
14. I feel that the person controlling the blue cursor did a lot of the talking.	4.39	1.99
15. I feel that the person controlling the green cursor did a lot of the talking.	3.75	1.73
16. I feel that the person controlling the red cursor did a lot of the talking.	3.75	1.59
17. I play video (arcade, Nintendo, etc) games a lot.	5.50	1.36
18. I've done something like these tasks before.	6.25	1.46
19. I play computer games a lot.	5.56	1.32
20. I was frustrated by the serial task.	4.94	1.64
21. I was frustrated by the coactive task.	5.14	1.57
22. I was frustrated by the conjoined task.	5.47	1.46
23. I think that the person controlling the blue did a lot of the work on the tasks.	3.75	0.94
24. I think that the person controlling the green did a lot of the work on the tasks.	3.50	1.08
25. I think that the person controlling the red did a lot of the work on the tasks.	3.47	0.91
26. I think that the tasks we did were interesting.	2.89	1.39
27. I thought the serial task was a productive way of way of performing the program.	3.86	1.55
28. I thought the coactive task was a productive way of way of performing the program.	3.25	1.44

Table 15 (continued). Average responses from the questionnaire (n = 36).

Question	Average	Standard deviation
29. I thought the conjoined task was a productive way of way of performing the program.	2.94	1.45
39. Overall, I thought the interface for these tasks were easy to use.	2.56	1.30
31. I thought the serial task was entertaining. (n=33)	3.06	1.58
32. I thought the conjoined task was entertaining.	2.81	1.51
33. I thought the coactive task was entertaining.	2.75	1.54
34. I was comfortable using the gamepad controllers in performing the tasks.	2.50	1.11
35. I think that the serial task of the program encouraged our group to communicate a lot.	2.94	1.84
36. I think that the coactive task of the program encouraged our group to communicate a lot.	2.78	1.71
37. I think that the conjoined task of the program encouraged our group to communicate a lot.	3.03	1.61
38. It was easy to do the program in the serial task.	3.81	1.53
39. It was easy to do the program in the coactive task.	3.36	1.46
40. It was easy to do the program in the conjoined task.	2.89	1.35

APPENDIX C: A WORKSHEET FOR DESIGNING COLLABORATIVE ACTIVITIES

The purpose of this worksheet is to help you design a collaborative activity. We highly recommend that you use this worksheet in conjunction with storyboarding your ideas. In this worksheet you will be presented a series of questions aimed at guiding you in your design.

I. The first few questions will focus on the task presented to the user in the activity.

1. First, think of an activity or a scenario that you want to develop. Briefly describe the activity or scenario. Use an extra sheet if necessary.

a) What is the goal of this activity?

b) What context do you see this activity used in?
- ❑ School
- ❑ Work
- ❑ Games/Playtime
- ❑ Home
- ❑ Museum or other public facility
- ❑ Other? (list)

c) What is the age range of the people doing this activity?
- ❑ Preschool
- ❑ Grades K-5
- ❑ Grades 6-8
- ❑ Grades 9-12
- ❑ College
- ❑ Graduate
- ❑ Post College/Work
- ❑ Senior Citizens

d) Would you like users in this activity to:

- explore a world or space of information? Are there obstacles in this space?
- answer a series of questions on a specific topic (drill and practice)?
- solve specific problems?
- use pre-defined simulations?
- modify existing simulations or create new simulations?
- create or build something (pictures, documents, etc)?

e) Is the activity a solely a computer application, or does it require other materials (like worksheets, etc)?

f) What role do you see a computer application taking in this activity (as a tool, as the activity itself)?

II. The next few questions will help you to focus on the people using your activity.

2) How many people will use your activity simultaneously?

3) Where are the users located?

- At the same machine?
- On different machines in the same room?
- At different machines?
  - With an audio link?
  - With a video link?

4) Are the users going to know each other before using the activity? Or do they meet through using the activity?

5) Are there reasons to hide some information from some of the collaborators (each user having a private space, the group has a public space)? If so, why?

6) How would you like the users to identify themselves to the system, or to other users?

a) Can users just join a session, or do they have to ask permission of the system or an existing group using your activity?

b) Is information about users stored in one central location?

c) What type of information could be stored about each person?

d) Can this information be used to allow people to quickly identify themselves to the system?

e) How do users appear in this activity? Are there certain affordances such as color, sound or pictures associated with a person's on screen presence? Does each person control an avatar?

f) Are there groups of people working together?

g) How do users join a group?

7) What happens when a user leaves the activity?

a) If the user is in a group situation, what happens to the rest of the group?

b) Is there warning that the user will be leaving the activity? Do they have to ask permission of the group or the system to leave?

c) What happens to the activity if one user leaves prematurely?

b) Does the user's on screen presence appear to "disappear"?

III. The next series of question will help you focus on the object in the activity.

8) What objects will the user see in this activity? (Is this a microworld with specific objects and properties? Is this a document creation system with tools?). Make a list of some of the objects in the world. Use extra sheets if necessary.

Answer the following questions for each of the objects listed above.

9) How do you envision people interacting with these objects? Are these computer-based objects? How do the users interact with the objects on the computer? (Through direct manipulation or other objects? Are agents involved? Are special input devices needed?)

a) Can the users create new objects in your activity?

b) Do users create objects by manipulating other objects?

c) Can users destroy objects?

d) Can users manipulate objects?

e) Are some users excluded from using objects in your activity?

IV. The next few questions will help you focus on how different types of objects are controlled in your activity. Start by identifying two or three of the most important types of objects and answer these questions. (Start by focusing on one object, then work on other objects)

10) How simultaneous do you want the users to interact with an object
- ❑ The users are allowed to access the object at the same time.
- ❑ Only one user can access the object at a time.
- ❑ If one user is accessing an object, other users can help by accessing it at the same time?
- ❑ If one user is accessing an object, another user can hinder the first user
- ❑ More than one user must work simultaneously to manipulate this object.
- ❑ Other?

11a) How is this object controlled or manipulated?
- ❑ Click and drag, "picking it up and moving it"?
- ❑ Parts of it can be changed, moved, etc
- ❑ It is manipulated through some other, perhaps more complex object.

b) Is this manipulation designed to be:
- ❑ Fun?
- ❑ Instructional?
- ❑ Other?

c) How difficult will the interaction with this type of object be
- ❑ More difficult if done with other users (like a 3 legged race)
- ❑ Easier if done with other users (like moving chairs, each person moving an individual chair.)
- ❑ Requires more than one user to complete the task (like lifting a piano)

V. One method for evaluating how users collaborate is to evaluate how the users interact with objects in your activity. The next few questions will focus on what information you may want to log in order to evaluate user interactions.

12a) What might you want to learn about the user's interactions with each other?
- How much they are working together?
- Are they working separately, even if it means the interactions are more difficult?
- How the users divide up a task in a given situation?

b) What type of information do you think would be useful to store about an object?
- Where the object was located/moved and by whom
- When a user touched/selected an object
- Other?

c) What type of manipulations on an object by a user might be considered a significant contribution to the activity? What might be considered a minimal or insignificant contribution to the activity?

d) How do you think the information stored about the object interactions relate to the user interactions?

e) What do you think you could do with the information you store to help you investigate their interactions? How can you visualize this information?

APPENDIX D: HOW TO IMPLEMENT A NEW ACTIVITY IN COLT

Colt activities can be developed for Microsoft Windows 3.1 and 95 using a variety of input device solutions. The compiler and project or workspace file you choose to use will depend on the type of input device and operating system you wish to support. Table 16 lists the Development environment options for the Colt System

Table 16. Development environment choices for the Colt System.

Microsoft Windows Platform	Multiple Input Allowed	Input Device Type	Visual C++ Compiler Version	Project/ Workspace Name
Windows 3.1	No	Single Mouse	1.5	test.mak
Windows 3.1	Yes	Access.BUS	1.5	coop.mak
Windows 95	No	Single Mouse	4.0	Test32.mak/ test32.mdp
Windows 95	Yes	Game Pad joysticks	4.0	joy32.mak/ joy32.mdp
Windows 95	Yes	Microsoft NetMeeting	4.0	net32.mak/ net32.mdp

Begin by starting the compiler and opening the project or workspace of choice. There will be three major steps to creating your project in the Colt shell. These are listed below. Note that for the purposes of these instructions, the activity being created is called **my** activity. If you wish, you can replace all instances of the word **my** with the actual activity name.

Step1: Adding and modifying resources.

Begin by adding menus and other resources needed by the activity in the resource file. The files you will modify are resource.h, shell.rc, and mybar.bmp.

In resource.h there are some IDs that you will need to add or modify to create a menu and toolbar for your activity. Three IDs have already been created for you, those you may modify to be more self-explanatory. Add other IDs if they are needed.

```
#define ID_MY_MENUITEM1          32900
#define ID_MY_MENUITEM2          32901
#define ID_MY_MENUITEM3          32902
```

Next, use the Visual C++ environment to modify the bitmap for the toolbar for your

activity. This bitmap is in mybar.bmp and is shown in Figure 64. Each bitmap in the toolbar must be 16 pixels wide.

Figure 64. Default bitmap for the “My Activity.”

Finally, you will need to make changes to the menu and string descriptions in shell.rc. To start the menu looks as it does below in the text view. Modify the existing menu item names and IDs and add any new ones.

```
IDR_MYVIEW MENU PRELOAD DISCARDABLE
BEGIN
   …
   POPUP "&My Activity"
   BEGIN
      MENUITEM "&MenuItem 1", ID_MY_MENUITEM1
      MENUITEM "&MenuItem 2", ID_MY_MENUITEM2
      MENUITEM "&MenuItem 3", ID_MY_MENUITEM3
   END
   …
END
```

The string descriptions are entries in the STRINGTABLE corresponding to these menu items. These must also be modified to correspond to the changes to the menu items.

```
STRINGTABLE DISCARDABLE
BEGIN
    ID_MY_MENUITEM1          "Help text for menu item 1"
    ID_MY_MENUITEM2          "Help text for menu item 2"
    ID_MY_MENUITEM3          "Help text for menu item 3"
END
```

Step 2: Adding or modifying the objects in the CO-API hierarchy.

At this point you should be ready to create new and/or use the existing cooperatively controlled objects in the CO-API hierarchy. They will be used in the CoImage shell. Figure 65 shows a partial hierarchy of objects in CO-API. Most objects correspond to their own header and implementation file, for example, the class MRect is defined in rect.h and implemented in rect.cpp

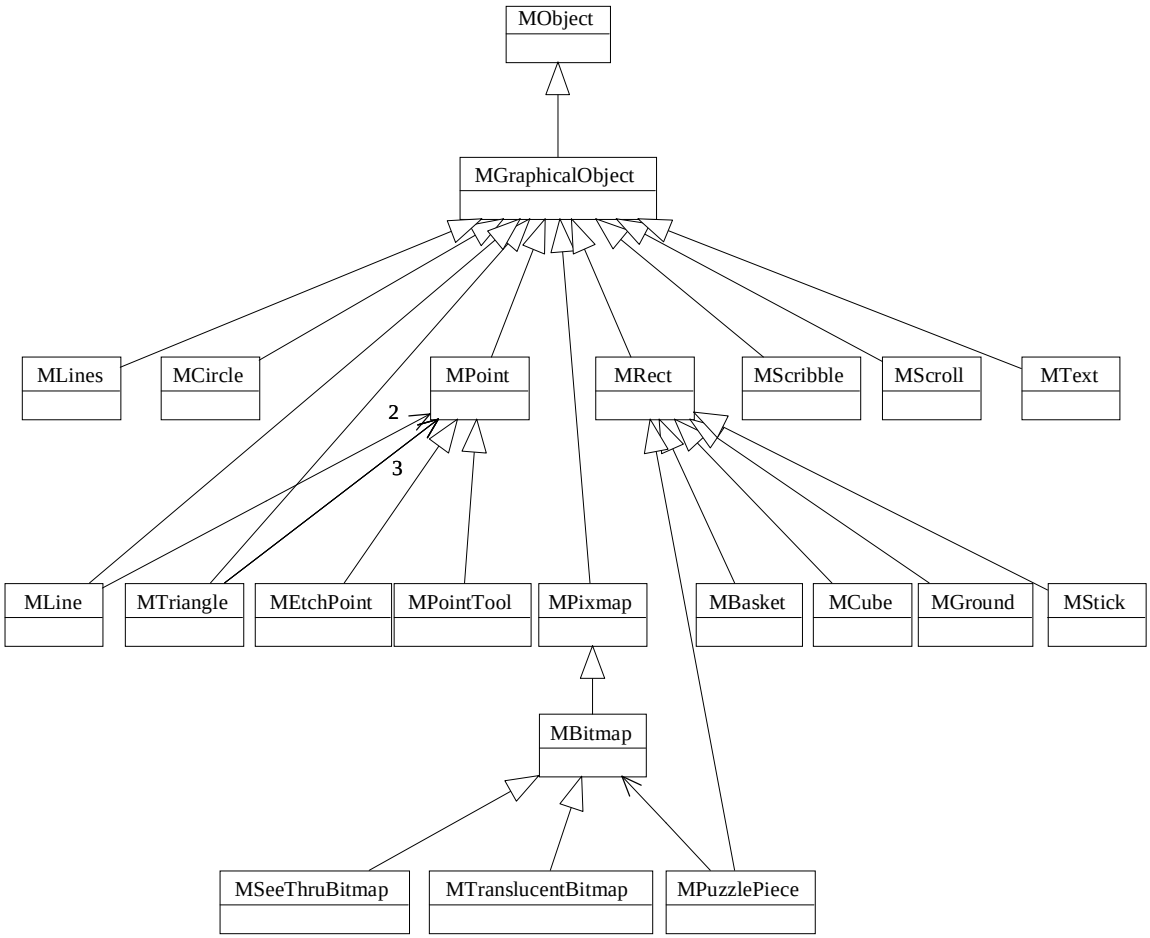

Figure 65. Hierarchy of Cooperatively Controlled Objects in the CO-API.

Step 3: Modify the CoImage Shell application.

Start by modifying the CreateMyBar() method in mainframe.cpp. The code, shown here, sets up a correspondence between the menu item IDs and the index into the mybar.bmp file shown in Step 1.

```
// Sets up My Activity toolbar.  Assumes that the users, etc have been created.
BOOL CMainFrame::CreateMyBar()
{
   // Creates the toolbar
   if (!m_wndMyBar.Create(this, WS_CHILD|CBRS_TOP, IDW_MY) ||
         !m_wndMyBar.LoadBitmap(IDR_MYVIEW))
   {
      TRACE("Failed to create my activity toolbar\n");
      return FALSE;       // fail to create
   }
```

```
    // Set the number of buttons in the toolbar. A separator is
    // counted as a tool bar button.  Note, there may
    // not be as many tool bar buttons as menu items.
    const int numToolButtons = 4;
    m_wndMyBar.SetButtons(NULL, numToolButtons);

    // Set up the correspondence between each toolbar button and
    // a menu ID.  The parameters for SetButtonInfo are
    //  int nIndex – the index of the button in the toolbar
    //  UINT nID   - the ID of the toolbar (corresponds to the menu
    //               ID)
    //  UINT nStyle – the button style (TBBS_BUTTON or
    //                TBBS_SEPARATOR)
    //  UINT nCursor – the bitmap ID (from the resource file) the
    //                 cursor should change into when the button is
    //                 hit.
    //                 Use IDB_DONT_CHANGE if the cursor doesn't
    //                 change.
    //  BOOL bCheckable – is this a checkable menu item/button.  A
    //                    checkable button will remain down once
    //                    pressed
    //                    until another checkable button is pressed.
    //  int iImage – the index into mybar.bmp of the bitmap for this
    //               button.  I don't know why, but it should be 6
    //               for a separator.

    m_wndMyBar.SetButtonInfo(0, ID_MY_MENUITEM1, TBBS_BUTTON,
                             IDB_SRC_CURSOR, TRUE, 0);
    m_wndMyBar.SetButtonInfo(1, ID_SEPARATOR, TBBS_SEPARATOR,
                             IDB_DONT_CHANGE, FALSE, 6);
    m_wndMyBar.SetButtonInfo(2, ID_MY_MENUITEM2, TBBS_BUTTON,
                             IDB_DONT_CHANGE, FALSE, 1);
    m_wndMyBar.SetButtonInfo(3, ID_MY_MENUITEM3, TBBS_BUTTON,
                             IDB_DONT_CHANGE, FALSE, 2);

    // Now set the privileges for these buttons.  This loop just
    // gives access to everyone, but you may want to limit which
    // users or groups access what buttons.  See access.h for
    // different ACCESS_*
    for (int ii = 0; ii < numToolButtons; ii++ )
    {
       // for now give everyone access to all buttons.  This may not
       // always be true.
       m_wndMyBar.AddGroupAccess(m_wndMyBar.GetItemID(ii),
                                 &m_AllGroup, ACCESS_READWRITE);
    }

    // Force the toolbar to redraw the next time a paint comes up.
    m_wndMyBar.Invalidate();

    return TRUE;
}
```

Finally, you will be modifying the files that actually implement the activity within the CoImage shell. The activity is written to conform to the standard Microsoft Foundation Classes (MFC) document/view scheme. In other words, information that is stored by the activity corresponds to variables in the document class, while methods that display and manipulate the document are part of the view class. In the case of this activity, the classes are CMyDoc (in mydoc.h and mydoc.cpp) and CMyView (in myview.h and myview.cpp). There are comments within the each file to indicate where code may need to be added. An example of an activity that was written in this structure is shown in etchdoc.h/cpp and etcview.h/cpp.

Along with creating member variables for the class CMyDoc, the following methods will have to be modified:

```
CMyDoc();   // constructor
~CMyDoc();  // destructor
void Draw(CDC* pDC); // method which is called each time the window
                     // is updated.  This may need to be modified.
void CreateControl(CSize mySize); // Create any cooperative controls
MObject* WhatIsHit(CPoint pt);    // determine what object was hit
virtual void Serialize(CArchive& ar); // overridden for document i/o
virtual BOOL OnOpenDocument(const char *pszPathName);
virtual void ResetDocument();
void SetDrawingArea(CSize mySize);
```

The following methods in CMyView will be renamed and modified. Additional method should be added to handle new menu items added in Step 1.

```
afx_msg void OnMyMenuItem1();
afx_msg void OnMyMenuItem2();
afx_msg void OnMyMenuItem3();

afx_msg void OnUpdateMyMenuItem1(CCmdUI* pCmdUI);
afx_msg void OnUpdateMyMenuItem2(CCmdUI* pCmdUI);
afx_msg void OnUpdateMyMenuItem3(CCmdUI* pCmdUI);
```

The following methods in CMyView should be modified to handle how user input effects the objects on the screen. For most activities, only the first three methods need to be modified.

```
afx_msg LONG OnCoopMouseMove(WPARAM wParam, LPARAM lParam);
afx_msg LONG OnCoopLButtonDown(WPARAM wParam, LPARAM lParam);
afx_msg LONG OnCoopLButtonUp(WPARAM wParam, LPARAM lParam);
afx_msg LONG OnCoopLButtonDblClk(WPARAM wParam, LPARAM lParam);
afx_msg LONG OnCoopRButtonDown(WPARAM wParam, LPARAM lParam);
afx_msg LONG OnCoopRButtonUp(WPARAM wParam, LPARAM lParam);
afx_msg LONG OnCoopRButtonDblClk(WPARAM wParam, LPARAM lParam);
```

VITA

Education

UNIVERSITY OF WASHINGTON
Ph.D. Computer Science and Engineering, 1998.
MS, Computer Science and Engineering, 1993.

UNIVERSITY OF MICHIGAN
BS, Mathematics and Pre-Med, 1985.

Publications

Baker, M. J., Bricker, L. J., and Tanimoto, S. L. *Cooperative interaction techniques in a computer-supported collaborative learning environment*. University of Washington Technical Report TR97-04-02. April, 1997.

Bricker, L.J., Baker, M.J., Tanimoto, S.L. Support for cooperatively controlled objects in multimedia applications, in Proceedings of CHI'97, Extended Abstracts (Atlanta, March 22-27, 1997), ACM Press, N.Y., pp 313-314.

Bricker, L., Tanimoto, S., McLain, E. CoImage - a Cooperatively Controlled Image Warper. Delivered at *the World Conference on Educational Multimedia and Hypermedia*, 1996, Boston, MA.

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.

Advising

Marla Baker, graduate quals project, 1996-1997.
Project: Cooperative Interaction Techniques for Graphical Objects in a Collaborative Activity

Supervisor, Emi Fujioka, undergraduate senior project, 1997.
Project: Cooperative chopstick activity implemented with Colt.

Supervisor, Chris Chapman, undergraduate work study student, 1997-1998.
Project: User study of cooperatively controlled color matcher activity.