

SUPPORTING COMPUTER-MEDIATED COLLABORATION THROUGH
USER CUSTOMIZED AGENTS

Letatia Bright Ducksworth, B.S., M.S.

Dissertation Prepared for the Degree of
DOCTOR OF PHILOSOPHY

UNIVERSITY OF NORTH TEXAS

December 2001

APPROVED:

Kathleen Swigger, Major Professor
Robert Brazile, Committee Member
Donald B. Cleveland, Committee Member
Brian O'Connor, Committee Member
Philip M. Turner, Program Coordinator and Dean of
the School of Library and Information Sciences
C. Neal Tate, Dean of the Robert B. Toulouse School
of Graduate Studies

Ducksworth, Letatia Bright, Supporting Computer-Mediated Collaboration through User Customized Agents. Doctor of Philosophy (Information Science), December, 2001, 83 pp., 7 tables, 5 illustrations, references 78 titles.

This research investigated a neglected problem – interruption of groups by agent advisory systems. The question was whether interruption by the agent advisory system was beneficial. A survey of literature in four areas is included in this dissertation. The areas surveyed were Agents, Online Help, Computer Supported Cooperative Work(CSCW) and Awareness in CSCW. Based on the review, a human subject experiment was conducted.

The study investigated whether the style of agent advisory interface improved the performance of group members. There were three sets of groups, a control set that did not have advisory agents, a set that had system provided advisory agents and a set that had group customized advisory agents. The groups worked together using a CSCW application developed with GroupKit, a CSCW toolkit. The groups with group customized advisory agents used an Agent Manager application to define advisory agents that would give them advice as they worked in the CSCW application.

The findings showed that the type of advisory agents did not significantly influence the performance of the groups. The groups with customized agents performed slightly better than the other groups but the difference was not statistically significant. When notified that advice was issued, groups with customized agents and groups with provided agents seldom accessed the agent's advice.

General design guidelines for agent interruption have not been solved. Future work is needed to finish the job. The definitive solution may be some mixture of the three known individual design solutions.

Copyright 2001

by

Letatia Bright Ducksworth

ACKNOWLEDGEMENTS

I thank Dr. Kathleen Swigger, my committee chair, for her untiring efforts in the creation of this work. Thanks to Dr. Swigger and Dr. Robert Brazile for providing their classes as subjects for this research. I appreciate Drs. Jack Becker and Donald Cleveland, members of my committee for their guidance and patience. A special thanks goes to Dr. Brian O'Connor for joining my committee with very short notice.

Thanks to Dr. Brazile, Dr. Dongil Shin and Xioaobo Peng for their technical support during this research. This project was supported by a grant from the State of Texas through the Texas Advanced Research Program (TARP).

I acknowledge and appreciate my husband, mother, family and friends who supported me emotionally and spiritually during this work.

TABLE OF CONTENTS

	Page
ACKNOWLEDGMENTS	iii
LIST OF TABLES	vi
LIST OF FIGURES	vii
CHAPTER	
1. INTRODUCTION	1
Aims and Objectives	
Overview	
Background	
Problem Statement	
Purpose of the Study	
Significance of the Study	
Research Design	
Research Questions	
Hypotheses	
Limitations and Assumptions	
Approach	
2. LITERATURE REVIEW	14
Agents	
Online Help	
Computer Supported Cooperative Work	
Awareness	
Summary	
3. METHODOLOGY	31
Aims and Objectives	
The Original Agent Advisory System	
Customized Agent Interface	
Main Hypothesis	
Method and Procedures	

4. DATA ANALYSIS AND RESULTS	50
Introduction	
Sample	
The Main Hypothesis	
Analysis of the Subhypotheses	
Summary and Conflicting Results	
5. RECOMMENDATIONS FOR FUTURE RESEARCH.....	58
Introduction	
Findings	
Conclusions	
Implications for Future Research	
Summary	
APPENDIX	
A FIGURES	65
B ASSIGNMENTS	69
REFERENCES	73

LIST OF TABLES

Table	Page
1. Descriptive Statistics for Average Grades on the Task Assignment for Each Agent Group	52
2. Simple ANOVA for Grades on Group Assignment	52
3. Ranking from Best to Worst for the Three Treatment Groups	53
4. Descriptive Statistics for Average Times for Each Agent Group	54
5. One-Factor ANOVA of Time on Task for Agent Advisory Interfaces	54
6. Descriptive Statistics for Number of Accesses of Agent Advice for Each Agent Group	55
7. T-Test for Number of Times Each Agent Group Viewed Advice	56

LIST OF FIGURES

Figure	Page
1. Item Summary Window.....	66
2. Customized Agent Architecture.....	67
3. Agent Creation Window	68

CHAPTER 1

INTRODUCTION

Aims and Objectives

This chapter introduces a recognized but currently neglected problem – interruption of groups by an agent advisory system. Further, this chapter discusses why this problem is important and timely and describes the approach chosen for this dissertation research. After reading this chapter, the reader should be familiar with the topic of interruption by intelligent agents during a computer-supported collaborative work session and understand the scope and important contributions of this dissertation.

Overview

Software agents have been the focus of attention in a number of research areas such as artificial intelligence (Maes, 1994), computer interfaces (Smith, Cypher & Spohrer, 1994), and distributed computing (Swigger, Brazile & Depew, 1994). A computer agent is a coherent set of interface objects or features that interact with the user as a mediator or facilitator of computer support (Whitaker, Frohlich & Daly-Jones, 1994). Intelligent agents have been created to provide valuable services as complex tools such as decision aids, intelligent software, or autonomous robotic vehicles. One of the goals of this research was to create agents that empower groups to work more effectively in the vast, rich, ever-changing world of electronic communication and information. Indeed, the animated "paper clip" has become a familiar sight to users of Microsoft's products.

Determining whether this particular feature is useful, however, is less clear. For example, do people actually listen to their agent's advice, or do they ignore it? Because liking does not always translate into "using," it was imperative that this study collect data about how advisory-style agents could be made more "useful" to their clients. This is especially true in Computer-Supported Cooperative Work (CSCW) environments where relatively little is known about how agents can be used to assist groups in a computationally distributed environment; an environment that requires people to cooperate at very high levels.

Computer-Supported Cooperative Work technology has been shown to be clearly useful in many domains that require the accomplishment of complex tasks by a small group of people (Huhns et al., 1994). Synchronous collaborative computer systems, however, have different user interface requirements than traditional, single user systems. Groups only intermittently interact with other members of their team because some of the time they work alone. Such multitasking systems require users to switch between concurrent or even simultaneous tasks. This particular style of intermittent interaction causes a dramatic increase in the side effect of user-interruption, particularly when the interruption comes from an autonomous agent. When an intelligent agent must communicate with its user, it must first interrupt the user from the other activity(ies) they are performing.

Thus, the idea of interrupting people, particularly a group of people, is a complex topic and one for which we currently have very few answers. Although the current literature recognizes the complexity and importance of this topic (Bowman, Tecuci &

Boicu, 2000), no one has yet published design solutions for building user interfaces for systems that must interrupt their users. This is particularly true of researchers within computer-supported collaborative environments that must deal with interruptions on a grand scale. This dissertation makes a contribution by dealing with this important topic.

Background

Help systems, whether they are intelligent or otherwise, have been the focus of attention in a number of research areas. Some research has been accomplished already in demonstrating the potential of using agent software to advise single users about communicating and seeking information (Barron, Hoffman, Ivers & Sherry, 1994). Missing, however, is research that explains how people use intelligent help systems or why, more specifically, certain types of help systems are more useful than others. One group of researchers argue that the success of an intelligent interface is dependent on either coercion or user satisfaction (Wooldridge & Jennings, 1995). Such studies argue that an interface will be used if the user likes it. A second line of argument assumes that fit to the social situation enables an interface's success (or failure). These studies claim that the agent's advice and interaction style must conform to the reward schemes and mental maps of its users (Selker, 1994). While we know that such explanations might be true of advisory-style agents that support individuals, we also know that programming agents for users of collaborative software is a much more complex task that requires a consideration of the group's goals, shared understandings, and system affordances. Thus, the prototypes and models in this study must clearly demonstrate how advisory-style agents can convey information, communication, and innovative mentoring that

supports groups of people working together. This research must ascertain what technical and social affordances are required to encourage groups to listen to the advice of an agent in a computer supported cooperative work (CSCW) system.

While the agent literature shows that intelligent aids can be made effective, it also shows that most users are unaware of the costs associated with delegating tasks, specifically the time that it takes to handle interruptions. Intelligent agents are usually constructed so that they make reports and requests of their users. This means that when the agent is ready to report its progress or display information, it must interrupt or distract the user from whatever he or she is currently doing. Thus, delegating a task to an agent does not necessarily free the user from the demands of listening to the agent. For example, Kirlik (1993) observed that the costs of delegating a task to a computer aid can sometimes outweigh the benefits, and that it is possible for peoples' performance on a multitask to actually decrease whenever they delegate tasks to intelligent agents, particularly if the user does not know how to manage the agents effectively.

Swigger and others addressed some of these problems in an attempt to improve group performances in collaborative software environments (Swigger, Brazile & Shin, 1995; Swigger, Brazile, Shin & Ducksworth, 1997). In order to enhance a group's understanding of various activities (i.e., requirements' elicitation), a special computer-supported cooperative interface was built to facilitate cooperation. Groups learned to be more effective collaborators by using specially designed computer tools that enabled them to organize and systematize their interactions. The cooperative interface relies on a shared, real-time window system (i.e., What you see is what I see) that allows groups

who are geographically distanced from one another to execute controlled tools in shared windows. The interface itself consists of a series of shared tools designed to support specific group competencies deemed important to effective group communication. It was also embedded with a set of advisory style agents that were designed to help groups with their initial planning tasks as well as assist them in resolving any conflicts that occurred. Agents were developed to detect ineffective collaboration among groups and provide them with information on how to work together more effectively. Results from studies that compared groups who used the initial experimental cooperative environment versus those who performed the same task face-to-face indicate that the system was able to improve task performance (Swigger & Brazile, 1995).

While the above advisory-style agents seemed effective in supporting collaborative human work, many users complained that they were intrusive and/or distracting to group work. In response to these complaints, the agent portion of the interface was changed to allow the user, not the system, to initiate the interaction with the agent. Although users were more satisfied with the interface, they tended to access the agent's advice less frequently and, as a result, task performance gradually declined. It became apparent that the more "passive" version of the agent system was less effective than the more intrusive version. What seemed to be required, therefore, was a different type of interface that would allow groups to customize an agent's advice to fit specific needs. In order to meet this challenge, this researcher concluded that an Agent Manager should be built that would allow groups to indicate their preferences for such things as rate of intervention, types of notifications, and message content. The new system could

then be tested with groups, to determine if it could increase group performance over the more "passive" agent system.

Previous research showed that interveners, that is individuals mediating technology-use for others, improved the successful adoption, implementation, and use of CSCW; these interveners played a major role in helping CSCW applications to succeed (Okamura, Fujimoto, Orlikowski & Yates, 1995). This research investigated software agents as the interveners. It was assumed that intelligent agents would perform the role of an intervener and help groups to succeed. Furthermore, Eveland and fellow researchers (1995) showed that human interveners were generally chosen from among the members of the team. In addition, they expressed the need for CSCW designers to consider social relationships when designing interfaces to support group activities. Again, this research assumed that allowing teams to create their own agents would provide groups with a method of self-help that could address certain social relationships, implicitly or explicitly.

Problem Statement

The integration of computer networks and communication systems in the workplace has led to the development of software that provides computer-based tools for communication, coordination, and decision-making within an organization. However, collaborative systems and the interfaces that support them are extremely complex (Greenberg, Gutwin & Cockburn, 1996). Such complexity often leads to cognitive overload, frustration, and error. Intelligent agents and/or intelligent interfaces offer some degree of relief from these types of problems. For example, (Lieberman, van Dyke, &

Vivacqua, 1999) uses a collaborative filtering technique to recommend Web Pages that are chosen by a group of users whose overall tastes match in explicit ratings of pages.

Unfortunately, intelligent collaborative interfaces also have their own set of problems, many of which relate to how people interact with each other and the interface. For example, intelligent agents are usually programmed to make reports and requests of their users. This means that whenever the agent gives advice or displays information, it must first interrupt or distract its user from what he or she is currently doing. Delegating a task to an agent does not necessarily free the group or individuals within the group from the demands of managing the agents. Someone within the group (or all the members of the group) must accept the costs associated with managing the different requests from the agent.

The second hidden “agent cost” is that which is associated with being interrupted and then having to listen to each message from an intelligent agent (Kirlik, 1993). The potential for interruption increases obviously as the number of users on the system increases. Most intelligent agents adopt a method of interrupting users whenever needed; thus facilitating speedy input or output of information (Maes, 1994). While such a method may be effective for single users, it is annoying for users of collaborative systems who are generally performing several different tasks at the same time. Thus, there seems to be a trade-off between when and how help and advice is displayed by an intelligent agent within a collaborative environment, and whether it needs to be immediate or delayed. This research investigated whether agents that captured a team’s self-help would

improve the cooperation among group members and enhance the overall level of performance of the group.

Purpose of the Study

This study sought to determine whether groups defining their own agents would perform better and more efficiently than groups that had no agents or had agents that were pre-programmed by developers. An Agent Manager was developed that allowed groups to program their agents to detect and respond to things such as a team's preferences for intervention or conflict resolution. Thus, the group and the system collaborated in developing and managing a set of procedures that embodied the team's shared understanding of collaboration. It was postulated that groups who developed their own agents would not become frustrated by constant interruptions and would not ignore the advice of a passive advisor. From this, it followed that the customized agents would be accessed more frequently and, as a result, would prove more useful to the group. Thus, groups with customized agents should perform a specific task more effectively than groups without customized agents.

Significance of the Study

There is an increasing need for the development of tools for improving the productivity of groups in their daily work, particularly when large geographical distances separate those individuals. Unfortunately, most tools that support computer collaboration are extremely complex, and performance is often impeded rather than enhanced. For example, Mandviwalla and Olfman (1994) complained that many users of collaborative applications were frustrated over both the number and type of collaborative tools, leading

group members to question whether such tools help or hinder performance. The Human Computer Interface (HCI) community has long been fixated on the “desktop” metaphor. Thus, it has been generally skeptical of the introduction of techniques from artificial intelligence, statistics, and other fields to help simplify complicated interfaces. Researchers within the CSCW and HCI communities have only recently begun the process of investigating how to incorporate intelligent advising systems into its software, but this has been limited to a very small number of applications. The software developed for this research represents a carefully designed intelligent collaborative interface that supports the study of the use of an intelligent advisory system for supporting group collaboration.

This research also establishes the importance and timelines of the problem of agent interruption during human computer interactions. The problem is discussed with the perspective that interrupting people is not “bad,” only complicated. For example, there is important research that concludes that intelligent agent systems can be distracting, difficult to manage, and difficult to program. This research tries to establish that effective agent systems appear to be those that respond or adapt to user actions (Lashkari, 1995). This is particularly true in multi-user environments that require agents to adjust to group, rather than individual styles. Agent systems also need to possess some mechanism for minimizing interruptions and interferences. Humans seem to respond to their agent interruptions by using two basic reactive strategies: (1) passively allowing the interruption, or (2) actively fighting the interruption by disabling or ignoring the agent advice. This research tries to address this problem by designing an agent management

system that would allow users, not programmers, to create their own advisory systems; thereby minimizing the degree of interruption.

Research Design

This study sought to determine how to increase the "usefulness" of agent advice within computer-supported collaborative environments. In order to do this, this researcher developed an Agent Manager that made it possible for a group to customize a set of agents to a particular task. To measure the difference in productivity among the groups using provided agents versus those groups using no agents or customized agents, the study assessed the grades attained by groups on a collaborative programming assignment.

Research Questions

The main research question for this experiment was whether the particular method for agent advice that is implemented in a user interface would affect groups' performance on a requirements specification task. A single measure of a group's performance was chosen as appropriate for this experiment: the grade on the requirements specification task. A second measure, the group's total time to complete the task, was selected to assess some of the secondary questions in the study. The primary research question became whether agents defined by the various experimental groups would improve the productivity or success of the group as measured by the grades attained by the group on a specific collaborative task. Since the task was performed within the context of a specific course, it was determined that grades were a reasonable index of performance. A series of secondary questions dealt with whether groups using group-defined agents would

complete the task in less time than other groups. Thus, the three specific questions explored were:

1. Will the success of the experimental groups be improved by the availability of customized agents?
2. Will groups view the advice of their agents more often after they have defined the agents?
3. Will groups using a customized agent system use less time to complete their tasks than groups using other methods of advisory style agent interfaces?

Hypotheses

Given the questions listed above, the following hypotheses were derived:

Hypothesis: The particular type of advisory style agent that is implemented in a user interface will affect a group's performance on a collaborative task. Research question one leads to this hypothesis. An exploration of this specific hypothesis suggests the following subhypotheses.

Subhypothesis One: The total time to complete the task will be affected by which method of agent advice system is implemented by the user interface. This hypothesis is related to research question 3.

Subhypothesis Two: Groups using a customized agent interface will access their agent advice system more frequently than groups using other agent advice systems. This hypothesis is based on the assertion that people are unreliable in their ability to perform situational awareness during multitasks. However, if given the opportunity to define their

own agents, they become more responsible in accessing the advice system. This hypothesis is related to research question two.

Subhypothesis Three: Groups using a customized agent system will use less time to complete their tasks than groups using other methods of advisory style agent interfaces. This subhypothesis is based on findings that people regard the predictability of response times an important influence on their behavior. Thus, groups that are allowed to manage their own interruptions should be more efficient than users of other types of agent systems.

Limitations and Assumptions

Because of both the nature and size of the experiment, the study was undertaken with the following limitations:

1. Due to the large number of groups required for the study, the method for population sampling was convenience rather than random.
2. The sample was also limited in that only computer science graduate students enrolled in Database or Natural Languages courses at the University of North participated in the study.

Finally, the study assumed that every group participating in the experiment would come to some consensus or agreement on the definitions and actions of their agents.

Approach

This research creates one of the first generalizable approaches for addressing the problem of agent interruption within computer supported collaborative environments. A broad survey of current literature was conducted to analyze and identify issues related to

some of the theoretical constructs presented in the research. This set of identified constructs was used to synthesize issues related to agent interruption and computer supported collaboration. The utility of these concepts was then validated, in part, with a human subjects experiment.

The following chapter contains an extensive survey of the published research about help systems, intelligent agents, and computer-supported collaborative interfaces. This survey is structured in a way that facilitates the generalization of previous disparate work. The survey provides support for the claim that it is useful to investigate agent interruption in the context of collaborative systems. This knowledge is then used to guide the creation of a hypothesis and its operationalization into a detailed empirical study.

This study is a contribution because it provides the first general foundation for investigating the problems associated with agent interruption in computer-supported collaborative systems. This study makes it possible for future researchers to discover general design guidelines for this particular user interface problem.

CHAPTER 2

LITERATURE REVIEW

The aim of this chapter is to identify a set of theoretical constructs about help systems, particularly intelligent help systems, and their possible application within computer supported collaborative environments. A broad analysis of relevant existing theory served as a foundation for the experimental study that was performed.

The current literature does not yet present a general and comprehensive theoretical model of the factors that are most relevant in intelligent help systems for real-time collaborative systems; building such a model is beyond the scope of this dissertation. It is postulated, however, that within the current literature there exist sufficient theoretical constructs about intelligent agents, help systems, and collaborative software to form a strong foundation from which to synthesize useful information. The object of this chapter is to form such a foundation.

Authors of the current research literature have proposed useful theoretical constructs for this particular study in each of four categories: agents, online help, computer-supported collaborative work, and group awareness. The agent literature shows that intelligent aids can be both effective and annoying. For example, research shows that users are often unaware that there is a cost associated with delegating tasks, that is generally translated into the time that it takes to handle interruptions. These same issues are problems for developers of online help systems. In general, the user activates online

help systems and so interruptions are kept to a minimum. However, developers of recent systems are taking a more active role in both the detection and intervention of help. Thus, questions such as how to interrupt the user and what to say when the interruption occurs are becoming an increasing problem for developers of online help systems. These issues are even more troublesome for developers of computer supported collaborative work (CSCW) who must design help systems for multiple users within multiple contexts. In part because the field is relatively new, there are few studies that discuss the effectiveness of online help for groups using CSCW systems. What seems apparent, however, is that problems such as awareness (that is, knowing what each member of the group is doing at any particular time) and knowing when to interrupt users are as much issues for designers of collaborative systems as they are for designers of online help systems.

This survey examines literature in each of the four categories in such a way that facilitates the generalization of various works and their applicability to this research. After reading this chapter, readers should understand the several individual theoretical constructs relevant to investigating intelligent help for computer-supported collaborative systems. Furthermore, readers should understand how this set of available theory can serve as a foundation for studying different factors that might affect group performance.

Agents

Humans have always been fascinated with non-human agencies: androids, humanoids, robots, cyborgs, and science fiction creatures (Bradshaw, 1997). It is not surprising that one of the most popular areas of computer research is the design of intelligent agents. One of the most important functions of intelligent agents is their ability

to accept requests for delegation of certain laborious, complex tasks. Thus, intelligent agents are computer programs that perform tasks on behalf of users (Selker, 1994). In short, agents are generally composed of anthropomorphic presentation and adaptive behavior, accept vague goal specification, give you just what you need, work while you don't, and work where you aren't (Shneiderman, 1997b).

The potential of using agent software in combination with various network software products for communicating and seeking information has been clearly demonstrated by a number of studies (Maes, 1994). For example, there are agent systems that incorporate information from different sources (Gruber, 1991) and that search for loosely specified articles from a range of document repositories (Voorhees, 1994). Malone et al. (1997) built an agent system to manage electronic messages (find, filter, sort and prioritize). The follow-on system (Oval) allows users to tailor their own systems for information management and cooperative work through objects, views, agents and links that help groups organize and respond to knowledge. A number of advisory style agent systems show that software agents can enhance cooperative learning by allowing documents to appear simpler and easier to use (Boy, 1997; Selker, 1994).

On the other hand, there are a number of studies that suggest that users become disenchanted with software agents and, as a result, end up disabling the intelligent aids. Intelligent agents are usually constructed so that they are required to make reports and requests of their users. This means that when the agent reports its progress or displays information, it must first interrupt or distract its user from what he or she is currently doing. Thus, delegating a task to an agent does not necessarily free the user from the

demands of listening to the agent. For example, Kirlik (1993) observed that the costs of delegating a task to a computer aid can sometimes outweigh the benefits. Kirlik found that it was possible for people's performance on a multitask to actually decrease when they delegated tasks to intelligent agents, and that the managerial strategy a person chose was critical to the success of the tasks (Kirlik, 1993). When people delegate tasks to intelligent aids, they must also accept the costs of personally managing that delegation. One of these costs is interruption and listening to the intelligent aid (Sheridan, 1988).

In addition to the costs associated with managing the agent, there is the issue of when and how an agent should interrupt its user with its advice or information. The potential for interruption obviously increases as the number of users on the system increases. Most intelligent agent systems adopt a method of interrupting their users whenever needed; thus facilitating speedy input or output of information (Maes, 1994). While such a method may be effective for single user systems, it is generally inappropriate for multi-users who are trying to perform multiple tasks at the same time. Thus, there seems to be a trade-off between when and how help and advice is displayed in a collaborative software environment, and whether it needs to be immediate or delayed. It should be possible, for example, to design a computer agent that will both support a group's efficient performance of tasks and, at the same time, manage group-interruption.

The activation frequency of an intelligent agent system may also be affected by the method used to communicate with the agent. Users activate intelligent agents in one of three ways: The agent can "learn" when to activate itself; the user can program (end-

user programming) when the activation will occur; the user can show (programming by demonstration) the agent when and how to activate itself (Terveen & Murray, 1996). All three techniques require at least some end user programming which is usually done through a special interface that allows users to specify instructions using precise rules or commands. For example, Malone et al. (1997) implemented agent programs using Oval, an extremely restricted and simplified language that allowed groups to share information, mailing objects, databases, and files. Groups entered their agent instructions using pull down menus and fill-in the blank forms.

While there is important research to conclude that intelligent agent systems are very useful, there is also some evidence to suggest that they can be distracting, difficult to manage, and difficult to program. Effective agent systems appear to be those that respond or adapt to user actions. This is particularly true in multi-user environments that require agents to adjust to group, rather than individual, styles. Agent systems also need to be easy to program, or possess some mechanism for learning user behavior. Finally, an effective agent system needs to find strategic ways to minimize interruptions. Humans seem to respond to their agent interruptions by using two basic reactive strategies: (1) passively allowing the interruption, and (2) actively fighting the interruption by disabling the agent system. The obvious solution seems to be to design an agent to present information about the interrupted activity in ways that help people resume those activities more successfully than otherwise (Lashkari, Metral & Maes, 1994).

Online Help

Questions related to the design of intelligent help system in human computer interaction are not completely solved problems; however there are some useful sources in the current literature, particularly in the area of the design of online help systems that can clarify issues within this field. The major objective of an online help system is to provide both training and assistance. The need for ongoing training has been recognized and documented (Johnson & Rice, 1987; Strassman, 1985; Eveland et al., 1995; Okamura et al., 1995). However, the steep learning curve for computer systems usually flattens out before sufficient skill is acquired (Pilkington, 1992). Thus, online help systems have been created to step in and provide ways for the user to understand and use the portions of the system that have not been mastered. In contrast to agent systems, a user generally initiates access to online help, which is normally done when the individual has a problem and perceives online help to be useful (Duffy, Palmer & Mehlenbacher, 1992; Henin, 1998).

Online help systems were originally designed to mimic the interaction style of reading a book (Rubens & Krull, 1988). Thus, most of the early online help systems were computerized versions of hardcopy manuals and written documentation (Turk & Nichols, 1996). However, a number of studies suggest that online help should provide more interaction between the system and the user (Pilkington, 1992). For example, Pilkington (1992) used common patterns from expert-user dialogues to create tutoring strategies and linguistic forms for help output. Similarly, Jones (1997) investigated ways to use an expert system to deliver help. Jones concluded that users need a help system that

identifies decision points, possible actions, and their corresponding results. The expert systems approach of addressing the current question or problem seems less complicated to the user, and user's responses to questions presented by the system guides them to the proper information (Jones, 1997). In order to better facilitate this type of interaction, a number of researchers suggest that help systems should be given natural language capability (Pilkington, 1992).

More recently, the use of Hypertext and HTML has been used to take online help systems up to the next level. Help systems have become more integrated into the actual performance of tasks instead of separate entities. For example, wizards and coaches now watch users perform tasks and show, rather than simply tell, them about how to perform tasks. Wizards use interactive, question-and-answer interfaces that step the user through the different stages of completing the task (Turk & Nichols, 1996).

Yet, there remain questions about whether these external interruptions help or hinder task completion. For example, Ummelen (1997) found that users continue to request help, even when you combine the help messages with the performance of the task. He also found that task results were improved when the help system did not take additional time to use or did not interfere with task performance (Ummelen, 1997). On the other hand, Rieman (1996) demonstrated that an effective and attractive strategy for learning a new system or investigating unknown features of software is exploratory learning. In exploratory learning, users are allowed to investigate the system in their own way, sometimes in pursuit of a task. There are no predefined sequences of steps to follow in learning the software, so the help system is accessed only on demand. Similarly,

Shneiderman (1997b) argues that mixed results have been achieved by the use of friendly human-like guides, introductory explanations of what to try first, and natural language dialog, and that the best approach to online help is to allow users to select and initiate help facilities

Regardless of how and when the help system is accessed, there is still the problem of determining what to say to the user whenever help is invoked. Hackos (1997) suggests using minimalist design principles to design effective online help systems, similar to those developed by Carroll (1987), formerly with IBM Research. This particular principle advocates displaying the correct information, using just the right words and graphics to ensure successful performance, emphasizing troubleshooting and correcting mistakes, and taking care to provide information for both novice and expert users. Online help is improved by the quality of the writing, its abilities to orient to tasks, and its sensitivity to context and appropriate examples. Finally, online help must be designed to address specific user communities by determining its goals and objectives (Hackos, 1997; Yetim, 1993).

Thus, the current research literature seems to suggest that a good online help system should deliver performance-oriented information in an easily accessible manner that leads to efficient knowledge transfer from help to performance of the task (Duffy et al., 1992; Yetim, 1993). Furthermore, online documentation should be modular, contain terms that the users understand, and have strong topic sentences that are independent of other topics. The primary goal of online help systems includes getting the user back on task. Anticipating the users' questions and providing easy ways of accessing and using

the answers should do this. An acceptable online help system is expected to model the user and give help-information about both the current display and the overall application (Strauss, 1993). To be useful to the user, help must be pertinent to the context of the application at a specific point in time. There remain some questions, however, about whether the help system should be under the control of the user (Yetim, 1993) or the application. While many of the current wizards and coaches actively interrupt the user's task and offer advice, there is evidence to suggest that users prefer to receive help only occasionally during the execution of a task, and that they often want to delay the viewing of the information. All these questions and more exist when trying to develop a help system that supports real-time collaboration. Some of the features of collaborative systems are discussed below.

Computer Supported Cooperative Work

The term, computer supported cooperative work (CSCW), is used to refer to groups of people working together with the assistance of a computer. This area of research, also called computer-supported collaboration, GroupWare, Workflow and Group Decision-Support Systems (Palmer & Fields, 1994), was introduced at a workshop organized in 1984 by Irene Greif of MIT and Paul Cashman of Digital Equipment Corporation (Grudin, 1994).

The two features that seem to distinguish CSCW systems from other types of applications are the mode of interaction they support and the geographical distribution of the users (Shneiderman, 1997b). Mode of interaction can be either asynchronous (that is, occurring at different times) or synchronous (that is, occurring at the same time), and

geographical distribution can be local, meaning that users are co-located in the same environment, or remote, meaning that they are in different locations. Most developers of collaborative software generally agree on describing their applications using this time and place framework. While these categories are distinct, actual GroupWare applications usually fit more than one category. For example, a teleconferencing system is generally classified as a same time/different place application but may support a recording system that allows team members to review the meeting at a later or different time (Grudin, 1994).

For the past decade, there has been a wide range of application development to support the sharing of information and tasks among groups. For example, multiple users located in different cities can now edit the same document simultaneously (Shneiderman, 1997b); conduct brainstorming sessions (Bly, Harrison & Irwin, 1993), check each other's calendars (Reinhard, Schweitzer, Volksen, & Weber, 1994); have real-time team meetings (Bly, Harrison & Irwin, 1993); play games (Virtanen, Gleiss & Goldstein, 1995), and even take classes (Baker & Gloster, 1994; Foster, 1995). Controlled experimentation on performance within these environments slowly determined their effectiveness (Ellis, Gibbs & Rein, 1991). For example, developers of CSCW systems emphasize the need for convenient turn taking and document sharing by using terms such as smooth, lightweight and seamless integration (Bowers, 1994). Other important features found to be effective include a mixture of private and public workspaces, identity of participants, location of actions, and care with updating (Bowers, 1994). There is also evidence that the dynamics of the network itself can impact a group's performance. For

example, researchers at the University of Toronto (Manteo et al., 1991) noted that many users complained of slow response time, distracting background noise, and flickering images.

It appears that users of CSCW systems use both a large variety and number of different tools to support collaboration such as video conferencing systems, electronic whiteboards, email, file sharing, and interactive computer aided design software. The research within this field that is particularly useful is that which assesses the effectiveness of these tools. Researchers have been able to identify specific work processes for distributed teams that make the most effective uses of technology, resulting in improved efficiency and quality of output (Greenberg, Hayne, & Rada, 1995). These studies suggest that collaborative tools are seen as one way to address the particular needs of a group by enhancing communication, fostering a sense of community, and supporting the work process (Javenpaa & Leidner, 1998). However, the increased complexity of these types of environments has caused some researchers to suggest that collaborative systems can sometimes impede group performance (Olson, Olson & Meader, 1995). For example, Mandviwalla and Olfman (1994) complained that many collaborative applications support only group activities, ignoring individual work that might ultimately affect a group's performance. There is also an absence of ways to customize a particular collaborative environment. Most collaborative systems support a small subset of a group's activities within a single display (Mandviwalla & Olfman, 1994). Although most collaborative software is designed to accommodate different types of communication (i.e., written, voice, graphical), there are only a handful of systems that allow participants

to take on individual roles with explicit rights that permit individuals within a group to determine how information will be shared among the different members of the group (Reinhard et al., 1994). Other types of customization features have been suggested which include view-level sharing or medium coupling that allow users to select how and where they want data displayed (e.g., graphic vs. text) (Bentley, Rodden, Sawyer, & Sommerville, 1994).

Another major concern for developers of CSCW systems is the problem of managing the different levels of messages across the network. For example, the various software components that make up a system can reside on either the client or server machine. Whenever a component is replicated, a copy of the component is distributed to each user's machine. Replicated components reduce network traffic but sometimes lead to inconsistent components (Reinhard et al., 1994). On the other hand, centralized components ensure consistency but lead to greater network traffic and slower communication among the members of the group (Reinhard et al., 1994). Previous research recommends that the centralized method is best for small networks or when the machines are located physically close together. On the other hand, the distributed approach, or peer-to-peer model of computing, is best for large or remote networks because it accommodates a large number of heterogeneous workstations (Reinhard et al., 1994).

In addition to architectural issues, there is the problem of what technique to use to share different types of information among the members of the group. One approach is to share a single-user application; a second is to build special-purpose collaboration-aware

applications; a third is to provide support in the collaboration system outside of the application. Since the application developer is usually unconcerned with the physical distribution of components, the collaboration system approach has become the dominant mode of sharing because it provides support for multiple displays and different views (Bentley, 1994).

Because of these complex issues, second generation research has been involved in developing frameworks for building synchronous group applications (GroupKit, CoEx, Rendezvous, and CoLab). These frameworks provide developers with a number of high-level coordination and communication tools that can then be customized for a particular type of application. More recently, Tou et al. (1994) developed an application tool called CoSARA that insulates the application developer from low-level details. This second-generation software has allowed developers to create customized environments for different users and types of applications.

Although the literature on help systems for CSCW applications is fairly sparse, there have been some studies that indicate that such aids would be useful if offered in this environment. For example, Okamura et al. showed that companies who had individuals mediating technology-use for others improved the successful adoption, implementation and use of CSCW. These authors found that groups' "interveners" played a major role in helping CSCW applications succeed (Okamura et al., 1995). Similarly, Eveland et al. (1995) found that teams developed their own help networks, and that there was greater usage of the collaborative software among groups who formed these types of networks than other groups.

Computer supported collaborative systems are becoming an increasing part of mainstream computing. Coordination within projects or between organizations can now be facilitated by text, graphics, voice, and even video exchanges (Bly, Harrison & Irwin, 1993). These collaboration tools have been found to be effective (Bowers, 1994); and it seems that they will continue to enjoy success. Recent research has concentrated on creating environments that differentiate between public and private workspace (Martin & Corl, 1995). It is also looking at ways to customize the interface by developing the notion of different views or roles (Reinhard et al., 1994). The recent growth of collaborative architectures or frameworks has enabled programmers to create collaborative applications more quickly and easily (Liang et al., 1994). Answers to questions about how to share both the application and the users inputs have enabled developers to understand how to build more effective distributed applications. Although evidence is scarce, there are a number of studies that suggest that help systems can actually increase the usage of collaborative systems.

Group Awareness

Within the broad category of CSCW literature, researchers have become increasingly concerned about how to increase “awareness” among group members using their systems. As Greenberg, Gutwin and Cockburn, state: “Awareness is part of the glue that allows groups to be more effective than individuals” (1996, p.299). Awareness improves group members’ ability to make conscious decisions by keeping them up-to-date on important events (Dourish & Bellotti, 1992). For example, Bowers (1994) shows that a lack of awareness causes problems among users because groups do not know what

is happening in the current state. The need for awareness is based on people's need to know past, current and future actions within a shared environment for unstructured tasks (Schlichter, Koch, M., & Burger, 1998).

Dourish and Bellotti (1992) believe that the need for awareness information is independent of the task domain. As a result, they divide awareness mechanisms into two categories: informational and role restrictive. They found that collaborators use explicit facilities whenever they wish to inform others of their actions. Examples of informational awareness mechanisms include editing a log entry and sending a note. In role restrictive situations, a user's situation or personality and not the content provide information about the type of activity possible. A role describes a person's relationships to shared objects and other participants and is usually linked to a set of possible operations.

In order to increase awareness among groups who use collaborative software, researchers have developed a number of different specialized interface techniques. For example, Gutwin et al. (1996) developed widgets such as telepointers, radar views, multi-user scrollbars, graphical activity indicators, and auditory cues to enhance the group's knowledge of different user inputs. Most of these devices were found to help users better anticipate the actions of others. Awareness can also be maintained through the sounds of background activities (Gaver, Smith & O'Shea, 1991). For example, background sounds of a bottling factory floor were added to a CSCW team process control system for a remote and distributed team (Gaver et al., 1991). The previously unavailable factory sounds helped users maintain subconscious awareness of the various factory control activities they had externally backgrounded to floor workers. Robertson et al. (Robertson

et al., 1993; Card and Robertson, 1996; Rao et al., 1995) have successfully used peripheral information to help users maintain awareness of their location in information spaces.

Gaver and colleagues (1991) introduced action sounds (sonification of otherwise noiseless computer-based activities) into the CSCW system, SharedArk for shared virtual environments. Users could hear not only sounds for their own actions but also the sounds for everyone else's actions too. Users found this useful for staying aware of each other's activities and for locating people within the information space.

On the other hand, Honda and colleagues (1997) have become concerned about keeping a user's personal space separate from the group space. They considered the tradeoff between providing awareness and maintaining privacy, and chose to use a shared room concept as a way of keeping the areas separate. To show the presence of different members of the group, they use a combination of keying and chair representation methods. To maintain different awareness levels, they represent members in different body positions and use sound. In a somewhat different approach, Pedersen and Sokoler (1997) combine the CSCW group awareness ideas of video and audio access of team members' activities with the use of sonification for team activity awareness. Privacy is maintained by presenting only an abstraction of other team members' physical and computer-based activities. Users see each other as abstract images doing abstract things. The authors seek to provide the user with awareness that comes naturally in face-to-face settings. Their desire is to expand the application to include social interaction in general not just working life.

The literature on awareness within computer supported collaborative systems is helping developers understand how to create interfaces that facilitate both coordinated and individual tasks at the same time. The use of multi-colored telepointers, radar views, and sound are being used to help groups coordinate their activities within these collaborative environments (Greenberg, Gutwin & Cockburn, 1996). Issues concerning privacy versus awareness, however, continue to pose problems for developers of collaborative systems.

Summary

This chapter presents an overview of the important ideas deemed relevant for this particular study. These ideas were synthesized from the significant and generally relevant theoretical constructs identified in an extensive analysis of the current literature. The breadth and depth of this analysis, and the resulting summaries, made the theoretical product of this chapter a powerful tool for guiding this study. The importance of this study has been helped through the extensive analysis of each of the following domains of current literature: artificial intelligence (i.e., intelligent agents), human-computer interaction (online help systems), and computer-supported cooperative work.

The general question that was faced is how to make agents both useful and used. The four main examples of how to approach this problem were discussed in great detail in this chapter. These four areas represent the four general classes of design solution for this study's user interface design.

CHAPTER 3

METHODOLOGY

Aims and Objectives

The previous studies on agents, on-line help, and collaborative work guided the formation and operationalization of the primary hypotheses about the effects of different methods of agent intervention upon group performance within a computer supported collaborative environment. A computer-supported collaborative application was developed that was intended to help improve group performance through an agent advisory system. Several versions of the agent advisory interface were then created in order to improve group performance. A detailed human subjects experiment was then developed to validate hypotheses about how the different agent advisory interfaces would affect group performance. After reading this chapter, the reader should have experienced the utility of the different intervention methods on user performance. The reader should also be familiar with the results of this investigation.

Overview

The study presented in this chapter represents a practical solution to the questions raised by the previous research work on agent intervention in collaborative systems, the human interruption problem, and intelligent help systems. It provides a detailed description of what factors contributed to successful and unsuccessful group performance. These factors are the significant and practically useful part of dissertation.

The study builds on work previously presented (Swigger et al., 1997) that shows that while groups perform better when provided with immediate advice, they are often dissatisfied with the frequent interruptions caused by the constant display of windows containing the different agents' messages. The descriptions of this initial system and of its provided, immediate agents are presented in the section on the Original Advisory Agents. In response to the user dissatisfaction, the agent advisory system was changed to allow users to have more control over when the agents' advice was displayed on the screen. Although groups were more satisfied with the new advisory system, they performed less well than groups who were given immediate advice. The current hypothesis was formed about how a group's customized agents would affect performance. It was hypothesized that the difference in the implemented method of intervention would affect a group's behavior on a requirements specification task. A third interface was developed that allowed groups to program (or create) their own agents' advice and method of intervention. The software was then used to help operationalize a human-subjects experiment to test the hypothesis.

A complete description of the human-subjects experiment with volunteers is reported in section labeled Method and Procedures. Briefly, subjects were placed into groups and asked to define requirements for a specific programming task. Each group was asked to use only the collaborative software to perform this task. Some groups performed the task without any interventions from any agents, some performed the task with the provided, delayed advisory agent interface, and some used the customized agent interface. All groups did their work on the class assignment using only the requirements

specification collaborative software. The control groups used the application without assistance from agents. The experimental groups used the application with assistance from programmer provided advisory-style agents or from advisory-style agents that their group had customized.

The Requirements Specification Collaborative Software

The requirements specification collaborative software was completed in the Fall of 1996 and is used to support distributed groups who are working in a real-time environment. The application software was initially developed as a tool for programmers located in different cities, states, or countries who need help in specifying requirements for a particular programming project. The specific components of the collaborative software are a shared graphical user interface, communications support software, and a database management system. The graphical interface was developed using GroupKit (Roseman & Greenberg, 1996), an applications development package designed by researchers at the University of Calgary for the purpose of creating and integrating special tools that support collaborative activities such as brainstorming, whiteboards, group management systems, etc. The collaborative software itself uses a client/server architecture in which specific items are stored in a database on a server and then made available for viewing independently on individual client workstations. The server process manages the object-oriented database and interacts with the client processes through a special support environment.

The graphical interface consists of an item summary window and various tools that support the collaborative process such as voting, chatting, and floor-management.

The item summary window serves as the primary interface for viewing and manipulating specific requirements (see Figure 1). There is no fixed way that the groups interact with the software. Rather, individual members of the group are free to enter items in any order or sequence. Because the information is stored in the system as an item or a part of item, it can be deleted, re-sequenced, displayed, etc. at any time through the item summary window. Other attributes associated with items include time and date created, author, priority, and type of activity. This information is also stored and used by the agent intervention software (see discussion below) to make decisions about when to interact with the users.

As previously mentioned, the application contains several tools that support the collaborative process. For example, users can “chat” with one another about various items or general information. If the user invokes the chat window, a similar window appears on all the screens of other group members. If a user requests a conversation about a single item, a corresponding chat window is created that contains the sequence number and content of that selected item. Under normal conditions, all the members of the group see the comments that are entered in a chat window. However, a user can stipulate an “individual chat” in which the number of users engaged in the discussion is restricted to selected members of the group. The system supports multiple types of chats for multiple groups at the same time.

Other collaborative tools supported by the application include a voting tool and a floor management tool. The voting tool can be configured to handle both unanimous and majority rule voting styles. The floor management tool supports three different

interaction styles: Free-for-all (all members can interact with the system at any time), Baton (the person who currently controls the system selects the next person to interact with the system), and Director (a single person selects who and when members interact with the system). When going from Free-For-All to Baton or Director floor management styles, the program conducts a vote. If the result of the vote is positive, the floor management changes. Based upon which floor management style is selected, members may then enter, view, and change any information on the screen.

The Original Agent Advisory System

In addition to supporting the requirements elicitation process, the special collaborative software had the “instructional” goal of helping groups improve their collaborative skills. The system accomplished this objective by invoking a series of provided agents that intervened and offered advice whenever they detected ineffective collaboration among programmers. In order to do this, the system recorded all the groups' activities. Two types of agents were then designed to monitor the various activities and intervene whenever certain events occurred. One type of agent monitored activities relating to the entire group (i.e., Group Agents), and the other type monitored individual actions (i.e., Member Agents). For example, Group Agents focused on items such as whether the group was working towards a definitive solution, while Member Agents were concerned with a particular individual's actions such as whether they were participating in the discussion. Group agents were programmed to display their messages to all the members of the group, while member agents displayed their messages to only the specific user who needed help. All activities were processed through a Master Event Handler,

which was responsible for managing requests in the queue, activating those requests, and re-directing information. After a pre-set number of group actions, the Master Event Handler checked the status of the active agents in the system to determine which ones needed processing. If the agent's processing status was yes, then the agent was executed. The individual agents, in turn, were represented as: (1) A list of activities to check, including the scope and relationship of those activities; (2) A threshold value for those activities; and (3) A set of actions to be performed, consisting of things such as displaying a message (to one, or all members), logging an activity, starting/stopping another agent, and/or adding another item. Since all user activities were stored in a database, the advisory-style agents were designed to retrieve specific data that allowed them to recognize a sequence of effective and ineffective behaviors and intervene whenever the group was demonstrating ineffective behavior. For example, whenever the Master Event Handler checked the "non-participating member" agent's status and detected the "yes" condition, it executed the corresponding agent for this particular type of action. The agent examined the database and when appropriate issued a message cautioning the user to start participating.

The individual agents were created by defining 1) a set of statistics that can be used to determine intervention (in the form of an SQL request to the database), 2) the event(s) that trigger the intervention (in the form of a rule), 3) the specific content of the intervention message, and 4) the location(s) of the intervention message. Thus, each agent was designed to look for evidence of good and poor behaviors, and then coach group members to become more effective in a particular area.

The system's provided agents were originally coded to detect four different types of poor collaborative behavior: Levels of participation, levels of collaboration, levels of discussion and evaluation, and group preferences. For example, the "participatory" agents checked a user's level of participation to determine if it was outside a particular range. These range values were usually set as percentages of the group's average transactions; members participating less than 20 percent below average or more than 20 percent above average caused one of the participatory agents to display its advice. Similarly, the "collaboratory" agents were designed to check the degree or frequency of use of the collaborative tools. For example, one type of collaboratory agent monitored the activities of the floor management tool, and another looked at how often the voting tool was used, etc. To determine the group's level of discussion or evaluation, another set of agents were designed to monitor the group's editing activities. For example, special agents were used to check whether a sufficient number of items had been added, changed, deleted, etc. Finally, there were agents that monitored the group's personal characteristics such as color preferences, window settings, etc. and warned the group whenever conflicts occurred.

This version of the agent advisory interface was tested with a group of students in Spring, 1995. Results from studies that compared groups who used this experimental cooperative environment versus those who performed the same task face-to-face indicated that the system was able to improve task performance (Swigger et al., 1997). Interaction was significant when a t-test was computed on these data ($F=2.66$; $p < .01$). While the above system provided agents seemed effective in supporting collaborative

human work, many users complained that they were intrusive and/or distracting to group work. Because the agents were hard-coded to intervene immediately whenever a certain percentage of the task was completed (or after a designated elapse of time), the actual appearance of an agent was unanticipated by most users. In response to these complaints, the agent portion of the interface was changed to enable the user, not the system, to display the agent's advice. For example, whenever the agent detected inappropriate behavior, the agent's advice was placed on a task list rather than displayed immediately, thus giving the user the option of opening the window at a later time. Additional awareness features were also added to the item summary display. For example, users could gain access to a history list that indicated the most recent actions of each member of the group. Special color features were also added to indicate whether a group member was logged into the session.

Although users were more satisfied with the second version of the agent interface, they were also less inclined to access the agent's advice. As a result, groups' task performance gradually decreased. In the fall of 1995, a test was conducted on the second agent interface (Swigger et al., 1997). Scores on the group task performed indicated that students were less effective in creating requirements using the new interface as compared to student groups using the system provided, immediate agent interface ($F=3.25$; $p < .01$). Thus, it became apparent that the current agent system lacked specific knowledge about how groups wanted to receive advice about this particular domain. It was the researchers' belief that such knowledge would make the agent system more useful because the advice could be tailored to a group's goals, habits, and preferences. Because the groups

themselves would be allowed to define the agents, then they might be more inclined to access the agent and listen to its advice. What was required, therefore, was an end-user-programming interface that would make it easier for groups to customize their agents' advice to fit their specific needs. In order to meet this challenge, it was concluded that an Agent Manager should be built that would allow groups to indicate their preferences for such things as rate of intervention, notification actions, and message content (See Figure 2). Once the group's preferences were entered, the collaborative system could then call on the group's agents to identify and resolve conflicts. The new system could then be tested on a group of users to determine if it could increase group performance over the current, provided agent system.

Customized Agent Interface

The proposed additions to the above system were designed to act similarly to the system's provided agents described above, but be driven by a user interface (similar to a group editor) that permitted groups to define the necessary information required by the agents. Moreover, the Agent Manager was expected to support a group of users who were located in different geographical areas yet needed access to the Agent Editor at the same time. Thus, the Agent Manager was built similar to a "shared" editor that allowed groups of users to specify the desired agent intervention. A typical sequence of editing tasks included: specifying the trigger conditions for an agent, entering the actions that might be taken as a result of that agent's invocation, indicating how the task would be monitored or how often users would be notified, etc. The Agent Manger itself was equipped with a number of monitoring capabilities that allowed it to offer help whenever a group was

having difficulty defining agent activities. For example, if members disagreed about how they wanted to be notified, the system was programmed to alert the members in the group about the conflict and suggest ways to correct the situation.

A Collaborative Agent Manager was developed that allowed groups to customize their collaborative help. The original advisory agent system had been programmed using a fairly standard set of knowledge representation objects. Each agent had been assigned a context, an action, a user or group of users, a time and date of delivery, etc. This same design was used to guide the development of the customized agent interface. To define their advisory-style agents, groups used an Agent Editor that accepts the context, actions, individual or group designation and the frequency of delivery from the group. The interface itself was similar to a fill-in-form and was intended to capture a group's collaboration style (Terveen and Murray, 1996). For example, a group is able to specify certain values such as the agent's trigger events, actions, message, recipients, and the date/time of arrival. As groups enter these items, the system continuously updates and displays the agent's context. Furthermore, groups are able to stipulate such things as the arrival time and order of particular messages or actions (See Figure 3).

The proposed additions to the original system included (1) a user interface that is similar to a group editor that permits groups to enter information required by the advice system, and (2) a program that links the newly created agents to the requirements specification collaborative software. Moreover, the Agent Manager was designed to indicate who receives the support; that is, whether the advice should be displayed to all members of the group or to a specific individual. The Agent Manager was also able to

specify the types of agent intervention desired by the group. A typical sequence of editing tasks might include listing the team profiles for the advice, listing the individual tasks required for a particular project, assigning an agent to a particular task, indicating how the task would be monitored and how often and which users would be notified.

The Agent Manager was then embedded into the existing requirements specification collaborative software. A special conference was created to run the agent creation program. The agent creation program displays the Agent Manager interface that allows the group to create a list of actions they want to monitor, the frequency of the interventions, and the target for the intervention (i.e., individual or group). Along with each item, the group indicates a trigger event as a logical relationship between how often the activity occurs and when they want intervention to occur (see Figure 3). This relationship can be specified as a single value or as a percentage. The group then specifies the actions that will be performed whenever the agent is invoked. Possible actions include starting and stopping the agent or other agents, logging a message, displaying a message, adding (or deleting) items, etc. Since agent creation is represented as a conference, any member of the group can create, delete, or update any specification for an agent. The selections (context of the agent) remain displayed until the agent is either accepted or rejected by the entire group. Once accepted, the agent is added to the group's agent database.

Each group's agent database is then accessed whenever the group runs the requirements specification collaborative software. For example, whenever a group starts the collaborative software, a special agent loading program places the group's agents into

each member's copy of the item summary display program. At various times throughout the session, the master program sends a message to each group's agent program telling it to check all its agent's premises and determine whether any should fire. If the agent's premises are true, then the corresponding actions are then either displayed or placed in the event queue.

Upon completion of the third agent advisory system, the different interfaces were tested on a number of different groups of students. The major objective of this research was to determine whether groups who follow the advice of customized agents are more effective than groups who follow the advice of provided agents. The primary activities completed during this phase of the project are described below in general order of implementation.

Main Hypothesis

Hypothesis (H₁): The particular type of advisory style agent that is implemented in a user interface will affect a group's performance on a collaborative task.

There are several sources in the literature that indicate the plausibility of the Main Hypothesis. In particular, the body of previous work suggests that individuals are adversely affected by when and how intelligent help is displayed (Carroll & McKendree, 1987; Shneiderman, 1997a; Maglio et al, 2000). It is not the purpose of this dissertation to examine every aspect of these conflicting perspectives. However, it is useful to investigate some different views and determine whether they are supported by the results of the experiment reported here.

Any intelligent advisory system is bound to cause some interruption as users switch between tasks. These interruptions, in turn, should cause some overhead cost in cognitive effort that must be met every time the group must pause to look at a display. This particular type of task has overhead costs and each user-interruption by machine causes potential task switching. However, if users create their own agents, then they have some suspicions of when and how these interruptions may occur. As a result, they should be less affected by interruptions and should be more inclined to accept an agent's advice in a timely fashion. This perspective, together with the Main Hypothesis, suggests subhypotheses 1.

Subhypothesis 1: The total time to complete the task will be affected by which method of agent advice system is implemented by the user interface.

People are unreliable in their ability to uniformly perform all parts of multitasks. This is the recognized problem that drives the study of situational awareness (e.g., Adams and Pew, 1990). People easily become immersed in performing single tasks in a multitask environment and tend to forget about their responsibility to other waiting tasks. Empirical evidence from the study of a proposed Call ID on Call Waiting telephone interface (Katz, 1995) found that when a user interface gives people the capacity to postpone handling interruptions, they sometimes ignore them too long. When people have direct responsibility for timing when to handle calls coming in on Call Waiting, they can sometimes delay so long that the new caller hangs up. When this occurs, the user may have to repeat actions and re-do activities. This perspective leads to subhypotheses 2.

Subhypothesis 2: Groups using a customized agent interface will access their agent advice system more frequently than groups using other agent advice systems.

Uncertainty in when groups receive feedback of their actions on computer-based tasks has been found to be an important design consideration of user interfaces (Shneiderman, 1998). People need to be able to predict when computers will intervene with advice. In human-human multitask environments, scheduling has been found to be the most useful time management strategy for increasing the performance on tasks (Hall and Hirsch, 1982), and that allowing users to stipulate when they want to execute their agents is deemed more satisfying. This perspective leads to the following subhypothesis.

Subhypothesis 3: Groups using a customized agent system will not use less time to complete their tasks than groups using other methods of advisory style agent interfaces.

Method and Procedures

An experiment was designed to test the main hypothesis (and its subhypotheses). Various versions of the advisory style agent software were created for addressing this problem.

The main hypothesis was tested by observing the relative effectiveness of each of two agent advisory methods. These methods are (1) provided advisory agents and (2) customized agents. A third version of the software was created in which no agent software was present. These three treatments were used as the values for the independent variable. Each of these three basis solutions was implemented in the requirements specification collaborative software. The experiment had subjects perform similar tasks

with different versions of the user interface. The dependent variable was groups' performance on the requirements definition tasks while using the different user interfaces.

1. Subjects

A total of 121 graduate students in the Department of Computer Science at the University of North Texas were successfully run as subjects in this experiment. All subjects were enrolled in either the Natural Language Processing or Database Systems Design Courses taught at the University of North Texas. All subjects were designated computer science majors. The majority of groups consisted of only three students, although there were a few four-subject groups. Due to the large number of subjects needed to recruit for the subject pool, it was necessary to select more than one class for this experiment. It was also necessary to run the experiment over a two-year period for similar reasons.

The students enrolled in the Natural Language and Database courses for Fall, 1997 comprised the control groups and the provided agent groups, while students enrolled in these same courses for Fall 1998 constituted the subjects for the customized agent groups. Students enrolled in each course were given a similar assignment and asked to use the collaborative software to specify requirements for a corresponding programming problem. Subjects self-selected themselves into three-member groups and scheduled times to complete the task. Unfortunately, the class size for the different courses varied over the two-year period. There were an uneven number of groups who participated in the experiment during the two years. Over the course of a two-year period, a total of ten

groups were assigned to the control group, eleven groups to the provided group, and twenty groups to the customized group.

This method for population sampling is less than random and therefore not optimal. However, it was judged adequate because of the exceptional problems associated with group-related studies and because of the universal motivation of a grade reward.

2. The Experimental Task

As previously stated, student groups enrolled in two classes were assigned the task of creating requirements for a specific program. The requirements specification task required subjects to use the special collaborative tool to enter a list of requirements for a specified program. There were no restrictions on the number of requirements that could be entered. The instructions also stated that members of the group had to agree upon the requirements before they concluded the exercise. Thus, the subjects were told that they must use the “voting” tool at some point to indicate their agreement/disagreement concerning each requirement created by the group. However, these instructions did not indicate the type of agreement (unanimous versus majority) that had to occur. Students were allowed to select the type of voting that worked best for their group.

Because the experiment spanned over two classes and two years, special care was taken to make all the assignments very similar to each other. For example, instructors of both classes agreed to make instructions comparable to each other and to create tasks in which the optimal number of requirements was equal to each other. Examples of the assignments given to the various classes can be found in Appendix B. A pilot study was

conducted to set the instructions at an appropriate level. Through testing with pilot subjects it was discovered that subjects also required training on how to specify requirements for a given program. Thus, this type of instruction was also included in the experimental study.

3. Treatments

Each of three treatment conditions employed a different form of the agent interface for performing the requirements task described above. The control groups implemented the requirements task in isolation. Subjects were asked to perform the requirements specification task without interruption from any agents. The “provided agent” treatment condition gave users control over when they would handle the agents. Whenever the system detected a problem with the group, an announcement was displayed on their task list. This announcement notified the group of the existence of a waiting agent message. Subjects could then click on the display and bring the matching message to the foreground or ignore it. If more than one agent message was queued, users did not have to perform all the queued tasks together, but instead could bring up any or all messages at will. The “customized agent” treatment condition acted similarly to the provided agent system except that the group’s own customized agents were loaded into the system at the beginning of the session. The group’s, and not the provided agents, were invoked if the system detected any problems with the group.

4. Apparatus and Procedure

All subjects performed the requirements task on workstations running the Unix operating system. Each subject was given their own computer and was isolated from the

rest of their team members. The experimental software was written partially in C++ and TCL/TK using GroupKit.

Groups were run usually two or three at a time at scheduled time intervals during a one-week period during the semester. The experimenter adhered to the following procedure to ensure that the treatments were administered to each subject in a consistent manner. One week prior to the scheduled experimental sessions, each instructor gave a two-hour lecture on how to create meaningful requirements and then distributed an assignment requiring the groups to create requirements for a specific programming task.

For the Customized Agent groups that participated in the experiment, a special one-hour session was provided in which the groups learned how to use the Agent Management Interface to enter their customized agents. The training involved an explanation and demonstration of how to use the tool to enter agents and a description of current agents. The groups were encouraged to ask questions throughout the period and were given numerous examples of agents that might be useful for this task. The groups were then instructed to create customized agents that might be helpful to their group when defining requirements for a programming task.

All groups were then scheduled for a two-hour experimental session. During the first part of the session, subjects were provided instruction on how to use the requirements specification software. During this training session, they also were given a practice problem that required them to use the system to define ten words to ensure they were comfortable with the software before they began the assignment for their course. Once this practice problem was completed, the groups were told to begin working on

their assignment. Groups were allowed to continue to use the system until they had completed the assignment.

CHAPTER 4

DATA ANALYSIS AND RESULTS

Introduction

Observations recorded during this experiment were used to empirically compare the three known solutions to the problem of providing agent advice. This experiment recorded observations of how people behave when given agent advice. These data can now be analyzed to determine whether reality supports the hypothesis.

The main hypothesis for this experiment predicts that “the particular type of advisory style agent that is implemented in a user interface affects a group’s performance on a collaborative task” (see chapter 1). A single measure of a group’s performance was chosen as appropriate for testing this hypothesis: A grade on the requirements specification task was used as the measure. The individual instructors of the Natural Language and Database courses determined the groups’ grades on the requirements tasks. Each instructor compared the student groups’ list of requirements with a pre-prepared instructor’s list to determine a score for the group. These scores were then entered into the data analysis program and used to compare the performance among the three groups. Other measures such as time to complete task and number of times agents were accessed were also recorded by the computer system and used to compare the groups for the subhypotheses. In addition, an audit trail was used to identify agent execution and users' actions after the agents provided advice.

There are many other interesting performance measures that could have been analyzed. However, these other categories of data were judged to be secondary to the main purpose of this experiment. The specific performance measures chosen represent the tasks that are most obviously appropriate for testing the main hypothesis and its subhypotheses. Analysis of other performance measures, although interesting, is left for future work. Only data collected for the experimental trials (and not practice trials) are included in these analyses.

Sample

The sample was a convenience sample drawn from computer science graduate students enrolled at the University of North Texas. The sample included graduate students taking courses in Natural Language Processing and Database Design. The sample was collected over the course of the following semesters: Fall 1997 and Fall 1998. Forty-one usable samples were collected for this experiment.

The Main Hypothesis

The following results were used to analyze the main hypothesis and lead to H_1 being rejected. Although the Customized Agent group performed better than either the Provided Agent or No Agent group it was not significantly better according to the ANOVA performed on this data. Therefore, significant evidence was not found for the influence of agent intervention style on task performance.

Two kinds of analysis are reported for this particular measure of performance (1) a table of descriptive statistics and (2) an ANOVA. An alpha level of .05 is used to make decisions of significance.

The descriptive statistics report the mean, standard deviation, standard error, and count. The descriptive statistics are reported in Table 1. The ANOVA uses a simple one-factor model as defined in Hinkle, Wiersma and Jurs (1994). This analysis is reported in Table 2.

Table 1. Descriptive Statistics for Average Grades on the Task Assignment for Each Agent Group

Treatment	N	Mean	Std. Deviation	Std. Error Mean
No agents	10	80.30	5.85	1.85
Provided Agents	11	82.00	9.89	2.98
Customized Agents	20	85.30	7.45	1.67

Table 2. Simple ANOVA for Grades on Group Assignment

	Sum of Squares	df	Mean Square	F	Sig.
Between Groups	188.139	2	94.070	1.527	.230
Within Groups	2340.300	38	61.587		
Total	2528.439	40			

Analysis of the Subhypotheses

The subhypotheses make predictions of differences between the individual treatment conditions. Tests of these subhypotheses are made with the results of the

analysis reported in the previous sections for the main measure of performance.

The following table summarizes these results to facilitate testing the subhypotheses. The previous analyses revealed no differences between pairs of treatments. Although there were no significant differences between the three groups (i.e., customized agents, provided agents, and no agents), it is possible to rank order the three treatments in terms of the mean grades on the experimental tasks. The following table summarizes the “best” to “worst” ranking of the three different user-interface design solutions. Ranking is done from 1 (best) to 3 (worst). Ranking is based on the results of the mean grades achieved on the requirements specification task.

Table 3. Ranking from Best to Worst for the Three Treatment Groups

Treatment	N	Mean	St. Dev
Customized Agents	20	85.30	7.45
Provided Agents	11	82.00	9.89
No Agents	10	80.30	5.85

1. Subhypothesis 1

Subhypothesis 1 says "the total time to complete the task will be affected by which method of agent advice system implemented by the user interface." This subhypothesis anticipated that users who ignored their agents would actually spend more time recovering from problems associated with this inattention. Descriptive statistics for the times are reported in Table 4. The ANOVA for this data is reported in Table 5.

Table 4. Descriptive Statistics for Average Times for Each Agent Group

Treatment	N	Mean	Std. Deviation	Std. Error Mean
No agents	10	1:06:06	0:19:28	0:06:09
Provided Agents	11	1:06:50	0:15:31	0:04:40
User Agents	20	0:56:56	0:20:22	0:04:33

Table 5: One-Factor ANOVA of Time on Task for Agent Advisory Interfaces

	Sum of Squares	df	Mean Square	F	Sig.
Between Groups	3373270.898	2	1686635.449	1.298	.285
Within Groups	49376760.077	38	1299388.423		
Total	52750030.976	40			

The ANOVA found no significant main effect. Therefore, these data can be said to reject subhypothesis 1. Although the groups with Customized Agents spent less time on the task than either the Provided or No Agent groups, it was not significantly less time.

2. Subhypothesis 2

Subhypothesis 2 says “groups using a customized agent interface will access their agent advice system more frequently than groups using other agent advice systems.”

This subhypothesis is based on the assertion that people are unreliable in their ability to perform situational awareness during multitasks, and that groups will forget to access

their advice systems if allowed to postpone the activity. However, if given the opportunity to negotiate the timing of their agents' advice, groups should become more responsible in accessing their advice system. The results for these data reject this view. The mean times the customized groups accessed their help systems were 1.4 as opposed to 2.0 for the provided groups (see Table 6).

Table 6. Descriptive Statistics for Number of Accesses of Agent Advice for Each Agent

	Group				
	N	Minimum	Maximum	Mean	Std Deviation
Customized Agents	20	0	9	1.4	2.1126
Provided Agents	11	0	6	2.0	1.9494

Subhypothesis 2 is only relevant to this experiment because of its indirect statement about the two user interface solutions for coordinating group collaborations. The implication is that if users access their advice systems more frequently, then they will perform better. Although the groups with the customized interface had the best performance, they actually accessed their agents less than the provided agent groups (see Table 6). The customized situation was intended to allow people to exercise their strength in dynamic negotiation. It, however, also seems to have allowed people to exercise their weakness in handling interruptions in a timely way, particularly when they realized the nature of the advice. When people are given control of choosing convenient

times to handle interruptions, they sometimes decide it is convenient to put them off, particularly if they know the content of the interruption.

Table 7. T-Test for Number of Times Each Agent Group Viewed Advice

Viewed Agent Advice	t	df	Sig. (2-tailed)	Mean Difference	Std. Error Difference	95% Confidence Interval of the Difference	
						Lower	Upper
Equal variances assumed	.777	29	.444	.6000	.7724	-.9798	2.1798
Equal variances not assumed	.796	22.214	.435	.6000	.7541	-.9630	2.1630

3. Subhypothesis 3

Subhypothesis 3 says “groups using a customized agent system will use less time to complete their tasks than groups using other methods of advisory style agent interfaces.” This subhypothesis is based on findings that people regard the predictability of response times an important influence on their behavior. Time-management is extremely important in completing tasks. It can be argued that users who are allowed to control their interruptions will perform tasks faster than users with less control. When allowed to program agents to interrupt at specific times, groups should outperform agent systems that either delay their interventions or are non-existent. Thus, this view predicts

that the customized user interface design solution is the “best” solution, and that the pre-programmed interface will rank second best.

The discovery of no significant effects for times among the three groups (Table 4 & 5) is adequate evidence to disconfirm this prediction. Again, it should be noted that the customized groups spent less time on the task, but that it was not significantly less time.

Summary and Conflicting Results

Both the main and subhypotheses were rejected. Although, in all cases, the customized agents groups did better and in less time, their scores were not significantly higher than the provided or no agent groups. The negative results from this experiment seem to highlight the complexity of the user-definition and interruption problem. Many of the theories and findings contained in the literature that have predictive power have only been tried in tightly constrained contexts. Their use for general interdisciplinary investigation and within complex environments is therefore inappropriate. There are many useful and relevant works that only investigate single user and single task domains in isolation. These can be used to predict the relative appropriateness of alternative user interface design solutions for portions of single-user agent systems. However, good performance on one task and with one user does not necessarily insure good performance on all tasks and with multiple users. The added complexity of group work makes it more difficult to isolate individual elements of a good design. A single user can influence the performance of a group and thus lessen the impact of a particular feature within a user interface. A more complete discussion of these findings and future research now follows.

CHAPTER 5

CONCLUSION

Introduction

The major goal of this research was to determine whether groups could collaborate more effectively using their own, customized agents as opposed to no or preprogrammed agents. The motivation for such a goal was to determine whether groups could develop advisory-style agents that were both useful and used. Although previous research found that system provided agents were deemed effective for improving group collaboration, groups complained that the agents' interruptions were intrusive and distracting. In order to correct this problem, the agents' method of interruption was changed so that groups, not the program, controlled when the agent's advice was displayed. Although users were more satisfied with this new design, performance declined. This previous work formed the basis of this dissertation by revealing the relationship between agent interruption and group performance. As a result, a Collaborative Agent Management system was built that allowed users to specify their own collaborative agents to support a requirements elicitation task. The Agent Management system is equipped with a shared interface tool that allows groups to specify various aspects of an advisory style agent. It also contains a mechanism that integrates the customized agents into the main collaborative interface. Following the completion of this task, a study was performed to test three different versions of the agent intervention

in the collaborative interface. That is, one variation of the system was created that would allow groups to customize their agents; a second version supplied users with system provided agents, but allowed users to schedule the agents' advice; a third version offered no agent advice. It was initially thought that those groups who were allowed to customize their agents would be less disturbed by the interruption of their agents and, therefore, more likely to read and follow their agents' advice. Therefore, the hypothesis for this study was that the three different versions of the agent interfaces would support different degrees of group performance on a collaborative task.

In order to test this hypothesis, students were separated into three sets of groups; a control set that had no agents, a second set that used system provided agents, and a third set that customized agents for their own use. Each group was provided training on the collaborative application itself. Groups assigned to the "customized agent" group were also given training on how to use the special agent-definition software. All three groups used the collaborative software to complete a requirements definition assignment. The collaborative software system collected data for all group interactions. Instructors were asked to assign grades to each group's output from the collaborative task. Thus, both grades on the requirements elicitation task along with data collected by the collaborative software system were used to evaluate different performance variables.

Findings

Although the average scores on the collaborative task were higher for groups that customized agents versus groups that had system provided or no agents, they were not statistically higher. Therefore, results suggest that experimental groups that are able to

customize their own agents do not experience an improvement in their performance as a result of this specific treatment. The average of grades attained by groups that used system provided agents was also higher than the average of the groups that had no agents. The levels of difference in all cases, however, were not statistically significant. The main hypothesis can be rejected statistically. Thus, the study was unable to demonstrate improved performance by groups who are able to define or customize their agents.

At the same time, three secondary questions were also investigated.

1. Will the success of the experimental groups be improved by the availability of customized agents?
2. Will groups view the advice of their agents more often after they have defined the agents?
3. Will groups using a customized agent system use less time to complete their tasks than groups using other methods of advisory style agent interfaces?

Data obtained from log files were analyzed, and it was concluded that none of the three subhypotheses was supported.

Conclusions

In previous research, it was found that groups are often frustrated by an interface's constant interruptions. On the other hand, groups tend to ignore the offerings of a passive advisor and, as a result, overall performance declines. These results prompted this researcher to build a Collaborative Agent Management system that could facilitate the creation of a user-developed help system that could support computer-supported cooperative tasks. The results of this study are mixed. Although their scores

were not significantly higher, groups that customized their agent system performed better than groups with no agents and groups that used system provided agents. When one is able to add the scores from Swigger's original study (Swigger, 1997), there appears to be a gradual increase in performance – with scores from the no-agent system at the bottom, followed by the provided-agent system, the customized agent system, and lastly, the fixed system provided agents at the top. While this finding is clearly not an argument for the main hypothesis, it does tend to suggest that customized agents are more helpful than nothing or system provided agents.

The study, however, failed to provide any evidence that groups who customize their agents are more productive; that is spend less time completing the specific task. There was no statistical difference among the three groups with respect to time to complete the task, nor in the number of times groups invoked their agents' advice. Part of the issue with respect to this specific data may be related to the fact that all of the groups were relatively inexperienced with the interface, and that this lack of experience may have affected group productivity. There is also some evidence to suggest that groups that had customized their agents were already aware of the agents' help messages. Thus, whenever an agent notified group members that a message was waiting in the queue, they did not have to look at the message because they were already familiar with its content. Their overall performance improved, however, because they were "reminded" to do a specific task (e.g., vote, add more items, etc.).

This dissertation makes a contribution by conducting basic research on the problem of agent interruption. The contribution of this particular experiment is also a first step toward discovering general design guidelines for the agent interruption problem.

Implications for Future Research

While this dissertation contributes to the problem of agent interruption within a computer supported collaborative environment, it does not provide a solution. The uncovered complexity of this problem implies that there is a vast amount of work that must be done before a final solution will be available. Some potential future areas of work were considered during the creation of this dissertation, but they were deemed outside the scope of the current work. These additional efforts are now outlined below.

The experiment reported in Chapter 3 had to fix on only a few variables and outcomes in order to isolate the comparison of interest. Only three subhypotheses were tested for this experiment, only 3 measures of performance were analyzed from the data recorded. There are several other possibly important subhypotheses and different measures of performance that could be tested and analyzed in future works. For example, it can be hypothesized (or “subhypothesized”) there is a relationship between performance and the groups’ activities either before or after agent intervention. For example, agent advice may be viewed less interruptive if a group is inactive. Since the collaborative system can monitor these types of activities, it should be possible to add a timer to the agent intervention system that detects inactivity. Another subhypotheses could determine the relationship between a group’s subjective preference of interruption and their objective performance. Do people like to customize their agents? Another

subhypotheses could state that there are correlations between individual differences of groups and their performance. Some possibly important individual differences that might be recorded in a questionnaire are: sex, age, and degree of knowledge of software. Another future work could examine the correlation between the agent interface and errors. This result might partially explain why there is no difference in productivity among the three different types of agent interfaces.

An interesting topic that was beyond the scope of this dissertation is a more in-depth investigation of a group's collaboration style. Styles observed during these experiments were: (1) Discussion of every item by every member before it is entered by a member, (2) Division of labor (i.e. each student enters a certain number of items), and (3) One or two dominant students who entered all of the items. An important future work might be to examine the correlation between various collaboration styles, productivity and the agent interfaces. The impact of agent advice should be different for the different collaboration styles.

General design guidelines for agent interruption have not been solved. Future work is needed to finish the job. The definitive solution may be some mixture of the three known individual design solutions. Some of the data collected for this experiment support speculation that such a mixed solution may be more useful than any one solution in isolation. Subjects interviewed after the study remarked that they preferred a system that would interrupt them while they were doing a task. Others claimed that they favored a system that would allow them to customize their advice. Finally, there were some

subjects who liked “unscheduled” interruptions but wanted to be able to take control and switch to a “scheduled” mode whenever necessary.

The creation of design guidelines that would be generally useful across different domains and environments is an ambitious future task. However, this is what is needed in order to make agents truly useful and used. In the absence of real design guidelines, systems designers will continue to rely on their own instincts to engineer usable agent advice systems. Hopefully, this research will provide insight and a software tool to help engineers deal with the agent interruption problem.

Summary

This multi-year research offered insight into group interactions supported by advisory style agents. It showed some difference in success but not enough statistically to declare improvement based on the presence of customized agents. It investigated productivity with similar results.

Because there was no statistically significant change in results for the different groups, this study can conclude that the interruptions by agents, both provided and customized, did not hamper the work of the group and may have helped.

APPENDIX A

FIGURES

FIGURE 1: Item Summary Window

Group: demo

File Collaboration Help

Task List
Awareness
Agent Definitions
Agents Advice

Update Vote Chat **letatia in Group: demo** Change Display Columns

Free For All Floor Management = FreeForAll

Voting Type = Majority

Number	Vote	Author	Creation Date	Item Content
1010	pending	letatia	Thu Nov 9 16:18:10 200	Count the number of departments in the company
1020	pending	letatia	Thu Nov 9 16:22:36 200	Add an employee to a department
2000	pending	dshin	Thu Nov 9 13:42:37 200	List all the children of each employee by their name and age
2010	pending	letatia	Thu Nov 9 16:19:03 200	List the budget of each department
3000	pending	dshin	Thu Nov 9 13:42:38 200	Count the number of children that each employee has
3010	pending	letatia	Thu Nov 9 16:19:56 200	List the employees in a department

New Item Content

Add


Print refresh  letatia Ned dshin

Figure 2. Customized Agent System

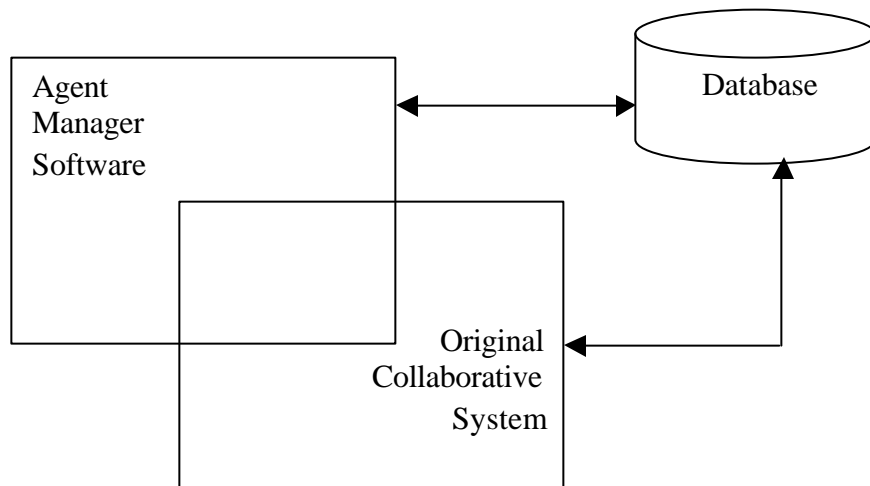
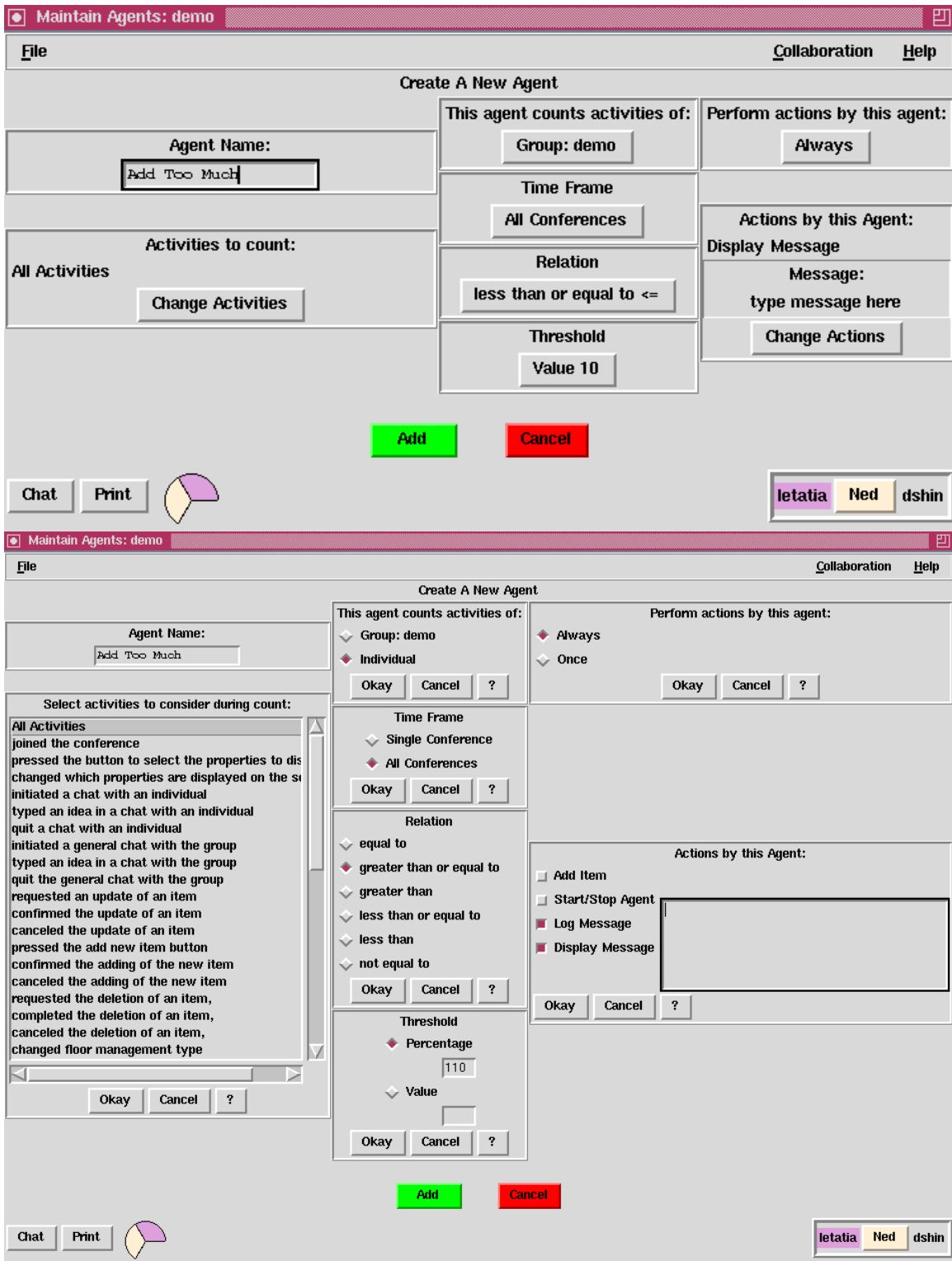


FIGURE 3: Agent Creation Window



APPENDIX B
ASSIGNMENTS

Project 5290 (Fall 1997)

I am the president of a travel agency. We have just created a web site for our clients. On this web site, we would like our customers to be able to enter English like statements and receive information about places, things, etc. that they would like to see or visit.

My staff has completed the task of putting all the information into the database.

Your group's project is to write the requirements for the natural language interface to this database.

The program will need to take English sentences and convert them to *sql* (a query language). The program should be able to handle the following types of sentences.

- a. Which cities have trains.** This sentence would be parsed. It would then produce a predicate in the form has (cities, trains). It would then produce a *sql* statement that would create a table of the form:

Has(cities char(15), trains char(15)).

- b. If a city has trains and I like trains then I should visit the city.**

Again, the sentence would need to be parsed to create a view that is the form:

Create view likes (<NAME>,train) as
Select <NAME>, train
From has, likes
Where has.train = like.train

The format for these two English statement is:

- a. S = <WHICH> or <WHAT> attribute predicate attribute
or
S = <WHICH> or <WHAT> attribute <IS> predicate
(The words in the brackets are constants.)

- b. S = <If> A1 <and> A2 <then> P
A1, A2, P = [a, an, the] attribute predicate [a,an,the] attribute
Or
[a,an,the] attribute <is> predicate

(The "a" or "an" indicates the first use of the attribute and the "the" indicates the attribute should be the same as a previous use of the attribute.

The program should not be case sensitive.

If the system cannot parse the user's sentence, then the system should print out "enter another question."

Specific Tasks

1. Use the Collaborative System to create a requirements document for this interface.
Do not panic if your group doesn't know anything about SQL or databases.
Remember that your job is to simply indicate the requirements for such a system. The above examples indicate what happens at each stage of the system, so you need to indicate what is required for that step.
2. Assign the tasks (i.e.) requirements to the people in your group.
3. Hand in a copy of your activity.

Grading:

Your project will be graded by looking at the list of requirements that your group generated and comparing it to the list of requirements generated by the TEACHER (i.e., ME).....

We will also use the requirements "later in the course" and determine if your requirements are adequate!

Project I CSCI 5350 Fall 1998

Your assignment is to create ten functional requirements for the database project described below. Each person in the group must contribute at least two requirements. The set of ten requirements will be the requirements you must satisfy for your programming project. The second part of the project is to implement the database system using Object Store and write a program that demonstrates that you have satisfied the functional requirements you created.

Airport Database

Keep track of airplanes at an airport. Every airplane has a registration number, and each airplane is of a specific model. The airport accommodates a number of airplane models and each model is identified by a model number (e.g. DC-10) and has a capacity and weight. A number of technicians work at the airport. You need to store the name, SSN, address, phone number and salary of each technician. Traffic controllers must have an annual medical examination. For each traffic controller, you must store the date of the most recent exam. All airport employees (including technicians) belong to a union. You must store the union membership number of each employee. You can assume that each employee is uniquely identified by social security number. The airport has a number of tests that are used periodically to ensure that airplanes are still airworthy. Each test has a Federal Aviation Authority (FAA) test number, a name and a maximum possible score. The FAA requires the airport to keep track of each time that a given airplane is tested by a given technician using a given test. For each testing event, the information needed is the date, the number of hours the technician spent doing the test, and the score the airplane received on the test.

REFERENCES

- Adams, M.J. & Pew, R. W. (1990). Situational awareness in the commercial aircraft cockpit: a cognitive perspective. Proceedings of IEEE/AIAA/NASA 9th Digital Avionics Systems Conference, 519-524.
- Baker, W. J. & Gloster, A. S., II. (1994), Moving towards the virtual university: A vision of technology in higher education. Cause/Effect, 17(2), 4-11.
- Barron, A. Hoffman, D., Ivers, K, & Sherry, L (1994). Telecommunications Florida Center for Instructional Technology, University of South Florida, Tampa, Florida.
- Bentley, R., Rodden, T., Sawyer, P, & Sommerville, I. (1994). Architectural support for cooperative multiuser interfaces. Computer, 27(5), 37-47.
- Bly, S., Harrison, S., & Irwin, S. (1993). Media spaces: Bringing people together in a video, audio, and computing environment, Communications of the ACM, 36(1), 28-47.
- Boicu, M., Tecuci, G., Marcu, D., Bowman, M., Shyr, P., Ciucu, F., Levcovici, C. (2000). Disciple-COA: From agent programming to agent teaching. LALAB Research Report, January.
- Bowers, J. (1994). The work to make a network work: studying CSCW in action. Proceedings of the ACM 1994 Conference on Computer-Supported Cooperative Work, 287-297.

Bowman, M., Tecuci, G., and Boicu, M. (2000). A methodology for modeling and representing expert knowledge that supports teaching-based intelligent agent development. LALAB Research Note, January.

Boy, G. A. (1997). Software agents for cooperative learning. In J. M Bradshaw (Ed.), Software agents (pp. 223-245). Menlo Park, CA: AAAI Press.

Bradshaw, J. M. (Ed.). (1997). Software agents. Menlo Park, CA: AAAI Press.

Card, S. K. & Robertson, G.G. (1996). The WebBook & Web Forager. CHI '96 – Video Program. New York, NY: ACM.

Carroll, J. M. (1987). Minimalist design for active users. In R. M. Baecker & W. A. S. Buxton (Eds.), Readings in human-computer interaction: a multidisciplinary approach (pp. 621-626). Los Altos, CA: Morgan Kaufmann.

Carroll, J. M. & McKendree, J. (1987). Interface design issues for advice-giving expert systems. Communications of the ACM 30(1), 14-32.

Dourish, P. & Bellotti, V. (1992). Awareness and coordination in shared workspaces. Proceedings of the 1992 ACM Conference on Computer-supported Cooperative Work, 107-114

Duffy, T. M., Palmer, J. E. & Mehlenbacher, B. (1992). Online help: design and evaluation. Norwood, NJ: Ablex Publishing.

Ellis, C., Gibbs, S. & Rein, G. (1991). Groupware: Some issues and experiences. Communications of the ACM, 34(1), 680-689.

Eveland, J. D., Blanchard, A., Brown, W., & Mattocks, J. (1995). The role of "Help Networks" in facilitating use of CSCW tools. The Information Society, 11(2), 113-129.

Foster, A. (1995). Virtual online university. [On-Line]. Available: <http://www.vou.org>

Gaver, W.W., Smith, R.B., & O'Shea, T. (1991). Effective sounds in complex systems: The ARKola simulation. Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '91), 85-90.

Greenberg, S., Gutwin, C. & Cockburn, A. (1996). Using distortion-oriented displays to support workspace awareness. In A. Sasse, R.J. Cunningham & R. Winder (Eds.), People and Computers XI (Proceedings of the HCI '96) (pp. 299-314). London: Springer-Verlag.

Greenberg, S., Hayne, S. & Rada, R. (Ed.) (1995). Groupware for real time drawing: A designer's guide. New York: McGraw-Hill.

Gruber, T. (1991). The role of common ontology in achieving sharable, reusable knowledge bases. In J. A. Allen, R. Fikes & E. Sandewall (Eds.), Principles of Knowledge Representation and Reasoning: Proceedings of the Second International Conference (pp. 601-602). Cambridge, MA: Morgan Kaufmann.

Grudin, J. (1994). CSCW: History and focus. Computer 27(5), 19-26.

Gutwin, C., Roseman, M. & Greenberg, S. (1996). A usability study of awareness widgets in a shared workspace groupware system. Proceedings of the ACM 1996 Conference on Computer Supported Cooperative Work, 258-267.

Hall, B.L. & Hirsch, D.E. (1982). An evaluation of the effects of a time management training program on work efficiency. Journal of Organizational Behavior Management 3(4), 73-96.

Hackos, J. T. (1997). Online documentation: The next generation. Proceedings of SIGDOC 97: 14th Annual ACM Conference on Systems Documentation, 99-130.

Henin, I. 1998. Evaluation of On-line Help. J.UCS(Journal of Universal Computer Science), 4(4). [Online] Available:
http://www.iicm.edu/jucs_4_4/evaluation_of_on_line/paper.html (accessed 6/13/2000)

Hinkle, D., Wiersma, W. & Jurs, S. (1994). Applied statistics for the behavioral sciences. Houghton Mifflin: Boston.

Honda, S., Tomioka, H., Kimura, T., Ohsawa, T., Okada, K. & Matsushita, Y. (1997). A virtual office environment based on a shared room realizing awareness space and transmitting awareness information. Proceedings of UIST 97: 10th Annual ACM Symposium on User Interface Software and Technology, 199-207.

Huhns, M. N., Jacobs, N., Ksiezyk, T., Shen, W-M., Singh, M. P. & Cannata, P. E. (1993). Integrating enterprise information models in Carnot. Proceedings of International Conference on Intelligent and Cooperative Information Systems, 32-42.

Javenpaa, S. & Leidner, D. (1998). Communication and trust in global virtual teams. Journal of Computer-mediated Communication, 3(4). [On-Line]. Available: <http://mc.huji.ac.il>.

Johnson, B. M. & Rice, R. E. (1987). Managing organizational innovation: The evolution from word processing to office information systems. New York: Columbia University Press.

Jones, H. K. (1997). Familiar contexts, new technologies: Adapting online help to simulate an expert system. Proceedings of SIGDOC 97: 14th Annual ACM Conference on Systems Documentation, 145-151.

Katz, R. (1995). Automatic versus user-controlled methods of briefly interrupting telephone calls. Human Factors 37(2), 321-334.

Kirlik, A. (1993). Modeling strategic behavior in human-automation interaction – Why an aid can (and should) go unused, Human Factors 35(2), 221-242.

Lashkari, Y. (1995). Feature guided automated collaborative filtering. MS Thesis, Massachusetts Institute of Technology, Media Arts and Sciences.

Lashkari, Y., Metral, M. & Maes, P. (1994). Collaborative interface agents. Proceedings of the 12th National Conference on Artificial Intelligence, 444-449.

Liang, T., Lai, H., Chen, N., Wei, H. & Chen, M. (1994). When client/server isn't enough: coordinating multiple distributed tasks. Computer, 27(5), 73-79.

Lieberman, H., van Dyke, N., & Vivacqua, A. (1999). Let's browse. Knowledge-based Systems, 12(8), 427-431.

- Maes, P. (1994). Agents that reduce work and information overload. Communications of the ACM, 37(7), 31-40.
- Maglio, P.P., Barrett, R., Campbell, C.S. & Selker, T. (2000). SUITOR: an attentive information system. Proceedings of the 2000 International Conference on Intelligent user interfaces, 169-176
- Malone, T. W., Lai, Kum-Yew, & Grant, K. R. (1997). Agents for Information Sharing and Coordination: A History and Some Reflections. In J. M Bradshaw (Ed.), Software agents (pp. 109-143). Menlo Park, CA: AAAI Press.
- Mandviwalla, M. & Olfman, L. (1994). What do groups need? A proposed set of generic GroupWare requirements. ACM Transactions on Computer-Human Interaction, 1(3), 245-268.
- Manteo, M., Baecker, R., Sellen, A., Buxton, W. A. & Milligan, T. (1991). Experiences in the use of media space. Proceedings of the ACM 1991 Conference on Human Factors in Computing Systems, 203-208.
- Martin, G. L. & Corl, K. G. (1995). System response time effects on user productivity. Behavior and Information Technology, 5(1), 3-13.
- Okamura, K., Fujimoto, M., Orlikowski, W., & Yates, J. (1995). Helping CSCW applications succeed: The role of mediators in the context of use. The Information Society, 11(3), 157.

Olson, J.S., Olson, G. M., & Meader, D. (1995). What mix of video and audio is useful for small groups doing remote real-time design work? Proceedings of the ACM 1995 Conference on Human Factors in Computing Systems, 362-368.

Palmer, J. & Fields, N. (1994). Guest editors' introduction: Computer-supported cooperative work. Computer 27(5), 15-17.

Pedersen, E. R. & Sokoler, T. (1997). AROMA: abstract representation of presence supporting mutual awareness. Proceedings of the ACM 1997 Conference on Human Factors in Computing Systems, 51-58.

Pilkington, R. (1992). Question-Answering for intelligent on-line help: The process of intelligent responding. Cognitive Science, 16(4), 455-490.

Rao, R., Pederson, J.O., Hearst, M.A., Mackinlay, J.D., Card, S.K., Masinter, L., Halvorsen, P.K. & Robertson, G.C. (1995). Rich interaction in the digital library. Communications of the ACM, 38(4), 29-39.

Reinhard, W., Schweitzer, J., Volksen, G. & Weber, M. (1994). CSCW tools: Concepts and architectures. Computer, 27(5), 28-36.

Rieman, J. (1996). A field study of exploratory learning strategies. ACM Transactions on Computer-Human Interaction, 3(3), 189-218.

Robertson, G.G., Card, K.K. & Mackinlay, J.D. (1993). Information visualization using 3d interactive animation, Communications of the ACM, 36(4), 57-69.

Roseman, M. & Greenberg, S. (1996). Building real time groupware with GroupKit, a groupware toolkit. ACM Transactions on Computer-Human Interaction, 3(1), 66-106.

Rubens, P. & Krull, R. (1988). Designing online information. In E. Barrett (Ed.), Text, context, and hypertext: Writing with and for the computer (pp. 291-309). Cambridge, MA: MIT Press.

Schlicter, J., Koch, M., & Burger, M. (1998). Workspace awareness for distributed teams. In W. Conen, & G. Neumann, (Eds.), Coordination technology for collaborative applications: organizations, processes, and agents (pp199-218) Berlin: Springer-Verlag.

Selker, T. (1994). Coach: A teaching agent that learns. Communications of the ACM, 37(7), 92-99.

Shneiderman, B. (1997a). Direct manipulation for comprehensible, predictable and controllable user interfaces. Proceedings of the 1997 International Conference on Intelligent User Interfaces, 33-39.

Shneiderman, B. (1997b). Direct manipulation versus agents: Paths to predictable, controllable, and comprehensible interfaces. In J. M Bradshaw (Ed.), Software agents (pp. 97-106). Menlo Park, CA: AAAI Press.

Shneiderman, B. (1998). Designing the user interface: Strategies for effective human-computer interaction (3rd ed.). Reading, MA: Addison-Wesley.

Sheridan, T.B. (1988). Task allocation and supervisory control. In M.Helander (Ed.), Handbook of human computer interaction (pp. 159-173). New York, NY: Elsevier Science Publishers.

Smith, D.C., Cypher, A. & Spohrer, J. (1994). KIDSIM: Programming agents without a programming language, Communications of the ACM, 37(7), 55-67.

Strassman, P. A. (1985). Information payoff: The transformation of work in the electronic age. New York: Free Press.

Strauss, F. (1993). Context sensitive help-facilities in GUIs through situations. Proceedings Human Computer Interaction, Vienna Conference, 79-90.

Swigger, K. & Brazile, R. (1995). Evaluating group effectiveness through a computer-supported cooperative problem solving environment. International Journal of Human-Computer Studies, 43, 523-538.

Swigger, K., Brazile, R., Depew, T. (1994). A computer-supported cooperative environment for requirements elicitation, Proceedings Infrastructure for Collaborative Enterprises, 165-170.

Swigger, K., Brazile, R., Shin, D. (1995). Teaching computer science students how to work together. Proceedings of the Conference on Computer Supported Collaborative Learning, 356-361.

Swigger, K., Brazile, R., Shin, D., & Ducksworth, L. (1997). Intelligent agents for teaching people how to cooperate. Proceedings of ED-MEDIA/ED-TELECOM 97 World

Conferences on Educational Multimedia and Hypermedia and on Educational Telecommunications. [CD_ROM]

Terveen, L. G. & Murray, L. (1996). Helping users program their personal agents. Proceedings of the ACM 1996 Conference on Human Factors in Computing Systems, 355-361.

Tou, I., Berson, S., Estrin, G., Eterovic, Y. & Wu, E. (1994). Prototyping synchronous group applications. Computer, 27(5), 48-57.

Turk, K. L. & Nichols, M. C. (1996). Online help systems: Technological evolution or revolution. Proceedings of the SIGDOC '96 Conference, 239-242. [On-line] Available: <http://www.acm.org/pubs/citations/proceeding/doc/238215/p239-turk>

Ummelen, N. 1997. Declarative information in software manuals: What's the use? Proceedings of the SIGDOC 97 Conference, 283-296.

Virtanen, M. T., Gleiss, N. & Goldstein, M. (1995). On the use of evaluative category scales in telecommunications. Proceedings of Human Factors in Telecommunications.

Voorhees, E.M. (1994). Software agents for information retrieval. ECAI-94: European Conference on Artificial Intelligence, 269-273.

Whitaker, S., Frohlich, D., & Daly-Jones, O. (1994). Informal workplace communication: What is it like and how might we support it? Proceedings of the 1994 ACM Conference on Human Factors in Computing Systems, 131-137.

Wooldridge, M & Jennings, N.R. (1995). Intelligent agents: Theory and practice. The Knowledge Engineering Review 10(2), 115-152.

Yetim, F. (1993). User-Adapted hypertext explanations. Proceedings of Human Computer Interaction, Vienna Conference, VCHCI '93, 91-102.