ADAPTIVE PLANNING AND PREDICTION IN AGENT-SUPPORTED DISTRIBUTED COLLABORATION

Ken T. N. Hartness, M.S.

Dissertation Prepared for the Degree of

DOCTOR OF PHILOSOPHY

UNIVERSITY OF NORTH TEXAS

December 2004

APPROVED:

Kathleen Swigger, Major Professor and Associate Dean
Robert Brazile, Committee Member
Yan Huang, Committee Member
Krishna Kavi, Chair of the Department of Computer Science & Engineering
Oscar N. Garcia, Dean of the College of Engineering
Sandra L. Terrell, Dean of the Robert B. Toulouse School of Graduate Studies

Hartness, Ken T. N., Adaptive planning and prediction in agent-supported distributed collaboration. Doctor of Philosophy (Computer Science), December 2004, 113 pp., 1 table, 16 illustrations, references, 93 titles.

Agents that act as user assistants will become invaluable as the number of information sources continue to proliferate. Such agents can support the work of users by learning to automate time-consuming tasks and filter information to manageable levels. Although considerable advances have been made in this area, it remains a fertile area for further development. One application of agents under careful scrutiny is the automated negotiation of conflicts between different user's needs and desires. Many techniques require explicit user models in order to function. This dissertation explores a technique for dynamically constructing user models and the impact of using them to anticipate the need for negotiation. Negotiation is reduced by including an advising aspect to the agent that can use this anticipation of conflict to adjust user behavior.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Aims and Objectives

This chapter introduces a known but currently neglected problem – recognizing and avoiding conflicts within a computer-supported collaborative environment. Furthermore, 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 conflict resolution within a computer-supported collaborative work session and understand the scope and important contributions of this dissertation.

Overview

"Collaboration awareness" (Procter et al., 1994), that is the awareness of other users and their contributions toward a mutual goal, is relatively easy to maintain in face-to-face meetings. When speaking face to face, people learn to respond to subtle visual and auditory cues that assist them in maintaining a cohesive consensus of action. Being aware of others in a computer-supported collaborative environment, however, is much more difficult (Johnson-Lenz et al., 1991). Limitations of network bandwidth and the technology itself can drastically reduce the number and type of subtle cues that can be exchanged among participants within a computer supported collaborative environment. Without proper cues, conflicts tend to arise much more frequently. To avoid such problems, techniques must be discovered that assist humans in recovering from the loss of visual and auditory cues, without hindering the quality or the quantity of the group communication.

This research focuses specifically on the problem of group conflicts and, as a consequence, on finding automated techniques for helping users avoid conflicts within a computer-supported collaborative environment. This dissertation explores a specific machine learning technique that was programmed to anticipate and react to conflicts that occur among distributed groups. The proposed study suggests that conflicts can be avoided if an 'agent' can be trained to learn user preferences and detect where and when conflicts will occur. For the purposes of this paper, user interface agents are computer programs or components of a computer program that operate with little or no direction from a user but work to serve a client by offering advice or streamlining the interface. Such agents are designed to monitor user preferences or typical user behavior in order to offer more personalized service. The proposed study used the agent interface paradigm to try and help groups resolve conflicts with a computer-supported cooperative work environment.

Computer-supported cooperative work (CSCW) technology has been shown to be 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 must interact with other members of their team in a more spontaneous manner. Further, such multitasking systems require users to switch between concurrent or even simultaneous tasks very rapidly. This particular style of intermittent interaction causes a dramatic increase in the occurrence of conflicts, particularly when users are deprived of both voice and visual contact information. When a group of users have limited access to both visual and auditory communication cues, they are forced to infer other users' intentions, often leading to erroneous conclusions.

Thus, the idea of helping users avoid conflicts, particularly within a computer supported collaborative environment, is a complex topic and one for which we currently have very few answers. Although the current literature recognized the complexity and importance of this problem (Klein, 1994), no one has yet published design solutions for building user interfaces for systems that help users avoid conflicts. This is particularly true of researchers within computer supported collaborative environments that must deal with conflicts on a grand scale. This dissertation makes a contribution by dealing with this important topic.

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 et al., 1996) and limited in the amount of visual and auditory cues available to users. Such complexity and sensory deprivation often leads to cognitive overload, frustration, and conflicts. Awareness techniques such as different identifying colors for each user (Hill and Gutwin, 2003), interface types (Shiozawa et al., 1999), visual displays, etc., offer some degree of relief, but the need to resolve conflicts among group members continues to be a problem within computer-supported collaborative environments. For example, while the use of tools such as telepointers, radar views, and sound help groups coordinate their activities (Greenberg et al., 1996), they also seem to add to the complexity of the interface. Unfortunately, these types of interfaces have not brought substantial relief for users of collaborative systems.

The introduction of agents, programs that act as personal assistants for their users, help users perform work (Silverman, 1992), communicate with one another (Isbister et al., 2000), and

avert problems. Interest in using agents to handle negotiations among work groups has grown over the years (Woitass, 1990). Different negotiation strategies have been preprogrammed to maximize a group member's effectiveness and efficiencies. However, these negotiating agents have usually emphasized individual rather than group interests (Isbister et al., 2000). While machine learning techniques have been used to establish weights on different individual preferences, there is little work on establishing weights for using this information to respond to group conflicts. Thus, there seems to be a trade-off between when and how conflicts can be eliminated within a collaborative environment, and whether a learning agent can respond to group rather than individual preferences. This research investigated whether agents that learned individual preferences of group members could also learn responses that eliminated conflicts among group members.

Purpose of the Study

This study sought to determine whether an agent based system could help groups avoid conflicts. An agent manager was developed that allowed groups to learn users' preference and a response model. The system was developed around a calendar application aimed at helping groups schedule different types of events. As a result, the agent learning system was designed to detect and respond to events such as scheduling a meeting according to the users' preferences. The specific learning technique that was selected to drive the agent training system was Q Learning because of its ability to adjust to evolving user attitudes without having to retrain from scratch. It was postulated that the agent system could monitor all communication between members of a group in real-time, learning from the behavior of the group. At the same time, the agent system would construct a response model that would be able to recognize and handle conflict situations. Using what it had learned, the agent would then offer advice, interrupt an

undesirable action with a warning, restrict users from taking undesirable actions, etc. The combination of advice, warnings, and restrictions would guide the users to a greater understanding of each other's preferences so that, ultimately, the user himself would be able to anticipate conflicts and work towards resolving problems before they arise. Thus, the agent learning system would be able to detect users' preferences and use this information to reduce the number of conflicts.

After using the system with live subjects, it soon became apparent after that the agent system was unable to learn all of the different possible conflict patterns in real time and a single user session. As a result, the learning portion of the system was changed to an offline setting in which the agent was 'trained' to learn user preferences and conflict patterns after being fed a series of user events. For each experimental run, the system recorded the percentage of events correctly classified as compared to a model of the user's true preferences, as well as the number of conflicts detected. Performance was then judged as to how well the system was able to learn the preferences and avoid conflicts.

Significance of the Study

There is an increasing need for the development of software 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. Although researchers are working hard to provide systems that provide a more virtual environment to make collaboration easier, there are still many problems for users of this technology to know and overcome. Conflict resolution is one of the most important problems that must be solved in this area. 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 conflict resolution. Although significant progress has been made in the area of intelligent advisors, most current research focuses on either user-independent awareness tools (DiMicco and Bender, 2004) or application-support advisors (Long, 1998). This research represents one of the first steps toward interfaces that customize themselves to the needs and preferences of the group.

This research also establishes the importance and timeliness of the problem of conflict resolution during collaborative interactions. The problem is discussed from the perspective that a system cannot resolve all conflicts, but it may be able to prevent major disagreements among group members. For example, there is important research that has proposed a conflict resolution method using coordination and argumentation agents (Kraus et al., 1998). This research tries to improve upon previous agent-based studies by suggesting that machine learning techniques might be used to both recognize and respond to conflicts. Humans seem to have several levels of responses to conflict, ranging from passive acceptance to actively fighting the specific event or proposed action. This research tries to address this problem by designing an agent management system that learns individual preferences and how to respond to conflicts created by these individual preferences.

Research Design

This study sought to determine whether an agent-based system could learn user preferences and, as a consequence, use this knowledge to predict and avoid conflicts within a computer-supported collaborative environment. In order to do this, an agent management system was developed to operate in a group environment; specifically, the agent was embedded within a shared calendar application designed to help groups schedule meetings. The agent portion of the system consisted of three components: a learning program that constructed a user preference

model, a learning program that learned a response model, and a planner that selected an appropriate response and determined how it should be implemented. To measure the effectiveness of the learning system, the system recorded the percentage of events correctly classified as compared to the users' true preferences, and the number of conflicts detected.

Research Questions

The main research questions for this study were whether the agent system could learn a sufficient number of user preferences, and whether this model of user preferences could be used to develop a response model that was designed to avoid conflicts. The answer to the first question was obtained by looking at the number of events correctly classified as compared to the users' true preferences. The second question was answered by looking at the total number of conflicts detected.

Limitations and Assumptions

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

1. Due to the length of time that it took for the models to converge, the learning experiments were conducted offline.
2. The study was also limited in that initial user preferences were formulated from models rather than real user populations.

Finally, the study assumed that the agent system had completed its training session when the no major changes were observed.

Approach

This research creates one of the first generalizable approaches for addressing the problem of learning to avoid conflicts 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 group conflicts and the reason for their occurrence within a computer supported collaborative environment. The utility of these concepts was then validated, in part, with the development of an agent learning system that was able to learn how to avoid conflicts by learning user preferences.

An agent-training tool provided the agent with simulated experiences. Although the simulated users may not have behaved identically to real users, the agent system was able to learn reasonable responses as a starting point. In this way, the agent did not have to learn all reasonable responses; it just had to customize itself to its particular user's needs. The training tool also had modes that allowed certain types of experiments to be run, allowing agent performance to be examined in situations that might be difficult to guarantee in a human work group.

The following chapter contains an extensive survey of the published research on intelligent agents, computer-supported collaborative interfaces, and conflict resolution strategies. 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 intervention in the context of collaborative systems. This knowledge is then used to guide the creation of a hypothesis and its operationalization into a detailed study.

This research shows that a user interface agent can learn to adapt its responses to its user's preferences and needs. Humans working together on the Internet will likely suffer from limited sharing of information, or the system may try to correct the situation by providing too much information. Software is needed that can remind users of the rules of group interactions and summarize group activity without overloading the user.

The interface agent described here attempts to correct this problem by summarizing user activities as preferences and learning to provide this knowledge as it becomes useful to the user. Occasionally, it may act, itself, to protect the interests of its user.

CHAPTER 2

LITERATURE REVIEW

The aim of this chapter is to identify a set of theoretical constructs about how to "automatically" resolve conflicts within a computer supported collaborative system. A broad analysis of relevant existing theory serves 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 automatically resolving conflicts within 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 exists sufficient theoretical constructs about collaborative software, intelligent agents, negotiation theory, and machine learning 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: Computer-supported collaborative work (CSCW), agents, negotiation, and machine learning. The computer-supported collaborative work literature indicates that increasing awareness (that is, knowing what each member of the group is doing at any particular time) among group members may minimize conflicts. As a result, they have developed a number of tools that are designed to help groups keep track of each other. Although this seems to improve communication among group members, it is not really clear if such tools actually decrease the amount or kind of conflicts that can occur within this environment.

It has been suggested that intelligent agents may be another way to reduce the conflict that can occur among group members. These “personal assistants” have been used effectively as critics and facilitators (see Maes, 1994, for some examples). However, there is a high cost associated with using intelligent agents that is generally translated into the time that it takes to handle interruptions. This cost can be reduced somewhat if personal agents are given the responsibility of handling the negotiations between other personal agents without having to consult with users. Elaborate negotiation strategies have been proposed that might help resolve both personal and machine conflicts. These negotiation strategies have been used in a number of applications such as planning and scheduling programs (Sen et al., 1997).

However, a more powerful solution for agent negotiation strategies lies within the machine learning community. What seems apparent is that problems such as negotiation (and knowing when to interrupt users) might be learned in some systematic way, and that this method may be more effective in reducing conflicts.

As a result, 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 automatic conflict resolution within collaborative systems. Furthermore, readers should understand how this set of available theory could serve as a foundation for studying different factors that might affect group performance.

Computer Supported Cooperative Work (CSCW)

The phrase "computer-supported cooperative work" refers to an area of study that explores methods for designing, facilitating and evaluating computer-based tools for the support of work groups. In Dix (1994), the components of the phrase are analyzed individually; the word

cooperative suggests the need for some sort of communication between members of the group, and the word *work* conveys the need to use or act upon various items, some of which are shared and some of which are not. In some CSCW applications, the communication is explicit; in others, the communication occurs indirectly through shared spaces that facilitate the work of the group or represent the products of the work, itself.

The word "groupware" refers to software that supports and even enables the accomplishment of group work (Greenberg, 1991); CSCW provides the research to enable and augment groupware as well as scientific validation for its methods. CSCW applications tend to support one or more of several types of working conditions that can be classified by spatial and temporal parameters (Grudin, 1994). Groupware tools may provide electronic support for the traditional face-to-face meeting (same-time, same-place) by disseminating information, supporting voting or negotiations, or providing a shared workspace in the computer that can be used to log the progress of the group (Valacich et al., 1991).

Some of these same tools can be configured to work over a distance (same-time, different place); combined with some form of teleconferencing, they can significantly reduce the impact of spatial distribution of the participants on the productivity of the meeting. Groupware, in this context, generally refers to CSCW tools that attempt to recreate opportunities for interaction which normally only occur in face-to-face meetings (Scrivener et al., 1994). Other CSCW tools may offer an extended version of traditional e-mail and bulletin board systems that already support a form of human interaction across both space and time. Appointment calendars, project integration tools, and collaborative editors with support for side notes or annotations are all examples of tasks that can take place independently of both time and space.

One type of groupware tool briefly described in CSCW overview papers like Grudin (1994) and Scrivener et al. (1994) supports groups meeting in the same place but at different times. This type of tool is used to support work areas when all members of the group are unable to meet together at the same time. A specialized work room has computer support much like the support in the special meeting rooms described for synchronous, face-to-face meetings; essentially, these tools combine same-time/same-place tools and asynchronous, distributed tools (limited to a local-area network or even a single computer interface). Like the face-to-face meeting support tools, the asynchronous work area is superior to conventional work areas because of its ability to record and print intermediate stages of the work.

Although useful within a single-user computer environment, most CSCW tools designed for spatially distributed workgroups rely on common computer input/output devices that often require less natural modes of interaction. Traditional, face-to-face meetings are governed by social and physical rules that are familiar to most people (Johnson-Levy et al., 1991). One of the major challenges facing developers of CSCW applications is finding tools that can assist groups without requiring them to learn new modes of interaction. Finding a style of communication that helps rather than hinders groups remains a serious problem for researchers in CSCW (Gutwin et al., 2004).

Table 2-1 Types of CSCW

	Same Time	Different Time
Same Place	Computer-supported meeting room.	Computer-supported work area with messages; “white board;” version tracking; etc.
Different Place	Online chat/conference; Shared interactive tool.	Threaded discussion (newsgroup); distributed, state-tracking support tools.

One of the reasons that communication is often difficult within a distributed CSCW environment is because sensory feedback is greatly reduced. Unfortunately, humans rely on subtle cues to resolve ambiguity and regulate behavior, which are often missing in a computer-supported work environment. Spatially distributed CSCW tools, short of full immersion in a virtual workplace, are limited in sensory feedback capability. The computer interface "attenuates people's sense of social presence" (Procter et al., 1994). "Collaboration awareness," as described in Procter et al. (1994), is the tendency for a group to be focused on a common topic, goal or subtask, even if they are also working on separate subtasks. It includes an awareness of each other's actions and how each participant is contributing to that focus. Collaboration awareness is difficult to maintain in CSCW, in part because individuals are more accustomed to thinking of computer work as an isolated activity, and systems providing limited sensory feedback do little to alter this perception. Some researchers argue that conflict resolution becomes less of a problem within computer-supported communications as group awareness is increased (Gutwin et al., 2004). Group awareness tends to activate social and corporate training signals that are used to resolve conflicts similar to what occurs in face-to-face situations (Rodden, 1996).

As a result, researchers have become increasingly concerned about how to increase “awareness” among group members using their systems. As Greenberg, Gutwin and Cockburn (1996), state: “Awareness is part of the glue that allows groups to be more effective than individuals”. Awareness improves group members’ ability to make conscious decisions by keeping them up-to-date on important events (Dourish and Bellotti, 1992). The need for awareness is based on a person’s need to know past, current and future actions within a shared environment for unstructured tasks (Schlichter et al., 1998). Although developing conflict resolution strategies may be important within a computer supported collaborative environment,

some researchers (Procter et al., 1994) argue that the use of feedback (and feed through) techniques is more important, and that CSCW developers should concentrate on creating systems that enable the use of conflict resolution techniques that are already part of a group's social makeup.

In order to increase awareness among groups who use collaborative software, researchers have developed a number of 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 and 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 factor 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.

Unfortunately, awareness techniques for CSCW environments have had mixed effects. While the use of tools such as telepointers, radar views, and sound help groups coordinate their activities (Greenberg, Gutwin and Cockburn, 1996), they also seem to add to the complexity of the interface. Collaborative interfaces are already fairly complex and mentally challenging, which has caused some researchers to suggest that collaborative systems can sometimes impede group performance (Olson, Olson and Meader, 1995). Additional awareness tools may, in themselves, distract users from the work to be accomplished, causing performance to decline.

One way to increase awareness among group members without additional complexity is to force participants to take turns performing different activities. For example, groups are able to maintain a clear focus in face-to-face meetings primarily because only one person is allowed to speak or act at a time. Social protocols help govern when one participant's turn is surrendered to another. In the absence of cues that allow these types of protocols to function within a collaborative environment, the computer may enforce its own set of rules that require or, at least, permit one user to take the focus or "floor" from another. One simple type of turn management protocol allows a user to interrupt another. Other protocols require users to engage in more formal mechanisms such as requesting control or placing oneself in a queue (Dommel et al., 1997).

At first glance, forcing group members to take turns speaking or acting seems counter productive, since one of the reasons for using a CSCW system is to increase group productivity through shared work spaces. However, if the “shared work” takes longer to complete or must be redone because members lose track of group goals and potential conflict situations, then overall group performance will suffer. Turn-taking is one way of maintaining a sense of collaborative awareness without interfering with group productivity.

CSCW research will continue to explore methods for enabling users to work together in spatially or temporally distributed groups even after total immersion technology becomes commonplace. The recent growth of collaborative architectures or frameworks has enabled programmers to create collaborative applications more quickly and easily (Winnett et al., 1994). Answers to questions about how to share both the application and the users’ inputs have enabled researchers to understand how to build more effective distributed applications. However, issues concerning awareness continue to pose problems for developers of collaborative software. The

literature on awareness within computer supported collaborative systems continues to show that it is difficult to develop tools that facilitate both coordinated and individual tasks at the same time (Gutwin and Greenberg, 1998). Although the use of multi-colored telepointers, radar views, and sound are helpful, they do not alleviate all the problems related to coordinating group activities. Conflicts continue to occur within CSCW environments, even when there are elaborate awareness tools available to users.

Agents

Dynamic and complex environments, such as those that support group work, need "user interfaces that are active and adaptive personal assistants." According to Huhns and Singh (1998), "Agents are active, persistent (software) components that perceive, reason, act, and communicate." One of the most important functions of intelligent agents seems to be their ability to accept requests and perform certain laborious tasks on behalf of users (Selker, 1994). For the purposes of this research, the word "agent" is used to indicate auxiliary software that supports the user and adapts to user behavior. The agent gradually becomes more adept at handling some set of chores for the user and keeps the user informed of important situations.

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, called InformationLens, to manage electronic messages (find, filter, sort and prioritize). The follow-on system, called 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).

In the last few years, researchers have introduced user interface agents or personal assistants to help users deal with a complex computer environment. As the number of lay people using computers increases, it is important for the user interface to take a more active role in the dialog between the user and the computer, responding to and even anticipating user needs. Agents that are programmed with user preferences or goals can handle simple situations or even act as proxies for their human users. Some chat room 'bots and MUD characters are programmed to interact with users in relatively sophisticated ways, providing them with the illusion of dealing with an intelligent being within a limited domain (Foner, 1997).

Another type of agent is one that "critiques human-generated solutions" (Silverman, 1992). Ideally, a “critical” agent will force the user to make unambiguous design choices that can be proven correct or incorrect (Silverman, 1992). The critic first generates its own solution using an expert system or case-based reasoner, and then compares the user's solution with its own. Differences beyond some maximum delta are recorded, and biases and missed opportunities are recognized. Silverman tests a user’s design against a set of standards in four specific categories: clarity, coherence, correspondence, and workability. The agent uses a dialog generator to translate specific errors into human-presentable form. Silverman’s design critic can operate in either an active or passive mode in order to assist both novice and experienced users.

A different type of critical agent can be designed to look at only the user’s communication rather than his or her work (Isbister et al., 2000). Communication agents follow the conversation and make suggestions to keep the communication moving. A helper agent

detects contextual cues, provides help, and then fades back into the background. It is designed to interact with users only when conversation lags. Isbister's conversational critic operates within a 3-dimensional virtual world; this world reveals the group's orientation to each other and shows the agent's presence. As different group members address each other or ask a question of one another, their 3-D representation moves accordingly. The agent is able to intrude gradually on users by moving its 3-D representation towards them.

Conversational agents have also been used to modify cultural differences among different groups. For example, Isbister et al. (2000) designed a conversational agent that would suggest culturally appropriate topics to be discussed between Japanese and American groups. Drawing on previous studies, the article suggested that Japanese prefer concise and calm exchanges over noisy and talkative ones, while Americans tend to prefer more emotional conversations. A conversational agent was programmed to either suggest safe or unsafe topics. A mix of Japanese and Americans were selected to participate in a discussion. Whenever the conversational agent encouraged "safe" topics, the Japanese who were participating in the study ranked Americans closer to their own culture, whereas when they were given "unsafe" topics, the Japanese ranked Americans further from their culture. Through this technique, Isbister et al. were able to alter user behavior without explicitly instructing them about different cultural biases.

Such unobtrusive agents engaging in gentle reinforcement of desired behavior may be preferable to forcing humans to accept a computer-generated compromise in cases of conflict. Humans can be encouraged to utilize social protocols they have already learned, or they can be instructed in protocols appropriate to the new communication environment. If the agent can predict conflict before it occurs, then an agent may be able to avert conflicts, eliminating the need for compromise.

Negotiation

Whenever several individuals work together, conflicts arise. Conflicts can arise over incompatible goals or subgoals or even conflicting priorities. In the case of distributed CSCW applications, conflicts can also arise because two or more individuals try to modify the same object at the same time. In many cases, standard protocols for distributed computing can handle these later types of conflicts. The more challenging area of study involves guiding or automating the processes by which people, not systems, resolve conflicts in traditional work groups.

Users, in general, are unaccustomed to working in a distributed work environment with limited bandwidth. Intensive communication within such an environment can significantly impact the work effort, especially during busy periods of the day when the network's bandwidth may be approaching or even exceeding the recommended maximum load. At the very least, an intelligent interface agent within a CSCW environment should anticipate the need for additional information or make suggestions about the current work situation so that users can focus on the source of the conflict. Ideally, agents can be programmed to automatically handle conflicts for their users.

Automated conflict resolution can cause certain types of problems to occur, as suggested in Woitass (1990). For example, some implementations of meeting schedulers require users to share their personal calendars, which can result in a violation of privacy. Also, users may be forced to list phantom meetings on their calendars to prevent the automated meeting scheduler from allocating all their time for meetings. In order to avoid some of these problems, Woitass (1990) used a centralized arbiter to broadcast proposed meetings to users. The agent could be programmed to accept or reject conflicting meeting times, but the final decision to accept a meeting time was left to the human user.

A great deal of work on different negotiation strategies has been done in the area of distributed artificial intelligence (Laassri et al., 1990; Decker et al., 1995). These types of systems tend to utilize a collection of agents to solve a problem. Negotiation is simplified because the agents are designed to work together and are programmed with the same primary goals. Thus, many agents within distributed artificial intelligence environments are benevolent (Sen et al., 1994).

There is also interesting work concerning negotiation among adversarial entities in multi-agent environments where the only common factor between agents is the protocol used to communicate. In this system, an agent conveys data and accepts compromises only if it can "improve" its position. As a result, the data transmitted to other agents may actually be inaccurate (Rosenschein et al., 1994). A number of techniques have been proposed that encourage honesty and cooperation by means of protocol selection or ensuring that the price of dishonesty and selfishness exceeds the benefits of "unsocial" strategies. These systems tend to rely on different "bidding" strategies to determine the benefits of a specific action. For example, Rosenschein and Zlotkin (1994) propose a technique that encourages honesty in bid calculations. They describe this method using the idea that long distance service selection can be based on bidding rather than customer choice. A company only has to underbid its competition to win the customer. However, competitive systems may be encouraged to provide inaccurate information in order to win negotiations. An agent that remains aware of other bids can always set its bid lower; alternatively, the agent may be able to offer a service at a much lower cost but try to maximize its profit. Rosenschein and Zlotkin suggest a more sophisticated bidding policy called "Vickrey's mechanism" that awards the bid to the lowest bidder but uses the second lowest bid to make the agreement. In the telephone company example, a company has no reason to overbid

because it will be awarded the next highest bid if it wins and is more likely to lose. Underbidding is only successful as long as all competitors bid more than the cost of offering the service. The point of this research is that heterogeneous agents can be encouraged to honestly portray the importance of winning a negotiation if some type of economy of resources constrains them.

All of these transactions, however, require a great deal of communication among agents. Communication is very expensive compared to computation, so techniques are often explored to minimize the need for explicitly sharing information among agents. Some believe that economic techniques are ideal for reaching consensus while minimizing the need for additional communication. For example, Wellman (1995) describes a negotiation strategy in which agents bid on resources until there is a balance between supply and demand. The messages are compact compared to detailed information about preferences. The assumption is that equilibrium is reached eventually when resources are assigned and agent actions are determined in a way that is acceptable to the other agents. Unfortunately, such a system is not guaranteed to achieve an optimal allocation.

Agents can also be instructed to make "deals" in order to maximize profits. Rosenschein et al. (1994) describe deals as joint plans that satisfy the goals of all involved agents. The utility of a deal is defined as the amount the agent is willing to pay for the deal minus the cost of the deal to the agent. The set of acceptable deals are those with a positive utility for all agents. Conflict resolution is handled by proposing a trade or by factoring in the cost of doing nothing.

The problem with systems that use utilities to negotiate conflicts is that the agents will try to maximize utility, resulting in selfish behavior. Enforcing social protocols by levying stiff penalties can help prevent some of the selfish behavior that can occur. For example, an agent that agrees to share resources and then decides that another plan is more advantageous should have to

pay a high price for abandoning the original plan unless the agent can convince other involved agents that it is advantageous for them as well (Sandholm et al., 1995). Failure to offer up flexibility or resources in exchange for betrayal is difficult to enforce without a centralized arbiter, but other agents can refuse to share resources with an antisocial agent.

Bidding and deal-making strategies, however, still require some communication among the agents. With enough knowledge, most if not all of the negotiation process can take place internally with perhaps just a little communication to verify that everyone is in agreement.

Fenster et al. (1995) describes a coordination process known as "focal points." Focal points are solutions that “tend to be picked.” For example, if asked to pick a number between one and ten, people are less likely to pick numbers at the extreme ends of the range. Focal point solutions tend to be unique, symmetrical where possible, and extreme in some sense. One problem with focal points is that joint plans are selected based on the likelihood that other agents will select them rather than on their worth or utility. If enough information exists to evaluate plans from the perspective of other agents, then this information can be used to pick the optimal plan.

Banerjee et al. (1999) try to eliminate communication costs by using a Bayesian network to model agent responses to various situations. The topology of a network determines what aspects of a situation help determine agent response. Prior and conditional probability values capture the contribution of each characteristic of a situation that leads to a final decision. Assuming that reasonable estimates of relative importance and likelihood of choices can be produced, it should be possible to adjust the probabilities to create a better fit to actual agent behavior.

Sen et al. (1997) proposes a very complicated negotiation framework for planning meetings. The framework depends on users explicitly stating their preferences and on agents repeatedly broadcasting proposals and counter-proposals until a solution is found. This negotiation framework involves setting preferences for each attribute of a meeting, represented by a real number between 0 and 1. Values close to the threshold represent weak preferences, and values closer to 0 or 1 represent strong preferences. Also, different attributes can be weighted (e.g., afternoon meetings may be more important than on what day the meeting is held).

A user starts the process by requesting a meeting for a particular group, length, priority, suggested time(s), deadline, etc. The user's agent then goes around and tries to find the times that are open for other group members, ranks these times according to preference, and transmits one or more of the best times to the other agents. The agent determines which times are acceptable by counting votes. If no time is acceptable, then the agent begins a new round of negotiation.

This particular framework uses an elaborate ranking method that assigns weights to different preferences and attributes. The system also tries to give more weight to times that were previously selected. For the meeting scheduler application, this means that times will be selected that meet the most requirements, or at least the more important requirements, for each agent.

Negotiation techniques range from methods involving the transmission of complete data about a user's preferences and the user's offers or counteroffers to methods that, in a sense, anticipate the negotiation process and arrive at an equitable compromise with little or no communication. These techniques rely on knowledge of the other competitors, knowledge that is provided explicitly, learned, or logically deduced. Some negotiation strategies rely on the accurate sharing of information, while others require feedback concerning a plan's or resource's

relative worth. In either case, the benefits of learning this knowledge indirectly rather than relying on possibly inaccurate transmissions should be explored.

Adaptive Learning

A number of machine learning techniques have been used for prediction and control. These techniques tend to be categorized as either symbolic or non-symbolic (Malcolm et al., 1990). Symbolic learning techniques generate human-readable rules that are adjusted or replaced over time (Luger, 2002), while non-symbolic systems tend to categorize the desirability of a classification or event using numeric weights or probability values (Haykin, 1994). While useful for capturing reflexive responses or categorizations, non-symbolic systems do not usually have the ability to explain their reasoning.

Neural networks are a popular form of non-symbolic learning technique. Neural networks are an attempt to emulate nature's efficient learning mechanisms (Haykin, 1994). Neurobiologists may be interested in precisely modeling the biological neuron, but most practical implementations of neural networks use a much coarser model based on different optimization theories or statistics. Neural networks are usually implemented as a collection of nodes, called neurons, which are connected to a set of inputs. A function is then used to convert these inputs into an output. Most neural network implementations have some method for adjusting the function based on a feedback mechanism that may be external or self-generated. Many implementations of neural networks use an array of values to represent the inputs into the network, an array of adaptable weights that help define each input's contribution to the function's value, and an array of node outputs.

One of the most well-known neural network types is the back-propagation network. Many practical solutions to problems of prediction or classification have been solved with this

type of network (Wasserman, 1989). In general, the various nodes within a neural network are assigned different weights based on their contribution to the eventual output. These weights represent the parameters of the system that are adapted so that the network can learn proper responses (outputs) to various situations (inputs). During the course of the learning process, the weights are adjusted to minimize the difference between correct outputs and actual outputs. In more complicated networks, such as two-layer networks, a back-propagation algorithm allows the network to learn more complicated response patterns because it introduces a degree of non-linearity in the output to ensure that each layer's weights have their own unique contribution to the final result. The sigmoid function $out = \frac{1}{1+e^{-net}}$ is often used for this non-linearity function because it forces all outputs to be between zero and one, eliminating the possibility of infinitely growing output while at the same time providing adequate discrimination for values near zero. It also solves the "credit assignment" problem for multiple layers (Wasserman, 1989) as it has a simple derivative for determining the relative contribution of each node to any error in the output.

Traditional back propagation involves adjusting the weights of the neural network in order to minimize the error of the network's response to a population of inputs. This type of machine learning technique, unfortunately, can quickly forget older exemplars if repeatedly presented with different ones; one way of dealing with this problem is to calculate the error on a training set of exemplars, then back-propagate this overall error (called epoch training). The training occurs off-line by repeatedly applying the network to the entire training set and recalculating an overall measure of its accuracy.

Probabilistic neural networks attempt to approximate Baye's method of classification using a Bayes-Parzen classifier converted to resemble a traditional neural network architecture

(Masters, 1995). Given a training set of exemplars that provides a full and representative coverage of different classes, a PNN estimates class membership by approximating the Bayesian probability that the input belongs in that class. The PNN is often designed to simply select the class to which the input most likely belongs, but its ability to report its selection criteria in terms of probabilities makes it attractive for use in hybrid systems where, for example, a more symbolic A.I. technique like a fuzzy rule-based system can use these estimates in its reasoning process (Luger, 2002).

Another feature of the PNN is that it can be retrained to incorporate new training exemplars relatively quickly. Unfortunately, this feature is balanced by longer classification times. The PNN must compare the input value against every exemplar in each class in the training set in order to classify it (every training example is represented by a neuron). For this type of dynamic system, many of the same questions that plagued training set selection for back propagation neural networks also plague the PNN. One clear advantage that the PNN has over the BPN is that new exemplars can be integrated into the training set more quickly.

In addition to simple classification techniques, there are a number of machine learning approaches that attempt to predict appropriate responses to a *sequence of actions,* either by predicting a user response or directly predicting when the negotiator should intervene. One approach to this problem uses a spatial mapping of time to overcome this problem (Haykin, 1994). Finite sets of past inputs are saved in a sliding window; the order of inputs within the window indicates the relative order of the events that caused them. The neural network is provided with all past inputs simultaneously and learns to assign an appropriate significance to more recent events versus older events. The implementation described in Haykin (1994) is

identical to the traditional BPN except for this technique that maps all past inputs in the window as if they were occurring at one moment.

Real-time recurrent networks support a less explicit method for determining the impact of a past event on the current behavior of a network. Like the binary flip-flop used to implement computer memories, this type of network uses feedback from the previous output to determine the current value of the output. Both hidden units as well as explicit output units provide input to the network. Supervised training of a recurrent network involves calculating the difference between the desired and actual outputs and multiplying the result by an error gradient calculated across each weight and adjusting the corresponding weight by the result (reduced by a learning rate parameter). The error gradient is estimated over time to be the contribution of a weight to the values of the visible outputs.

Induction is the process of creating generalizations from specific examples. Like scientific theories, rules or classifications created by induction can be subjectively evaluated for how well they explain or describe some object, process, etc., based on current evidence. Inductive learning techniques produce symbolic output, as opposed to machine learning methods that produce numeric values or weights.

A well-known induction method is the ID3 algorithm (Jackson, 1999). ID3 creates a decision tree from a set of examples. Each node of the decision tree represents an attribute of the object to be classified. The paths that lead from each node are labeled with values for that attribute. The leaf nodes of the decision tree represent possible classifications of the object. These decision trees can be mechanically translated into production rules for an expert system (Luger, 2002).

The previous learning methods are similar in that they learn from examples. That is, the program is provided with a set of input/output pairs, and the task is to learn a function that covers those pairs. However, there are other types of learning techniques that begin without training examples. The adaptive-critic, and other related methods, represents a variation on reinforcement learning. Reinforcement learning is supervised only in the sense that an occasional positive or negative reinforcement is applied to situations where performance is acceptable or unacceptable. Behavior is represented as a series of states or state transitions. Explicit reinforcement occurs only for states where there is a clear positive or negative evaluation; the system may be exposed to a number of states with no clear evaluation. Presumably, continuous reinforcement (traditional supervised learning) is impractical for these intermediate states, so in cases where a sequence of actions leads ultimately to a reinforcement signal, some method must be employed to distribute the feedback to the appropriate states that led to correct behavior. While some systems, such as drive reinforcement learning (Barto et al., 1983), distribute feedback by using techniques similar to the temporal-to-spatial mapping mentioned in the previous section, others use a dynamic programming approach to propagate the reinforcement values from the point of actual reinforcement back to earlier states. This adaptive critic then provides a pseudo-reinforcement signal to the actual reinforcement network, increasing the speed and degree of learning possible for states earlier in the sequence.

Q Learning, a reinforcement learning technique, completely eliminates the distinction between weights that indicate a state/action pair's worth and the estimate of future reinforcement. Instead, it assigns a Q value to each state/action pair. These Q values, hopefully, converge to some optimal level as actions are performed over time. Assuming that Q' represents the mapping of state/action pairs to an ideal measure of expected reinforcement, the optimal plan would

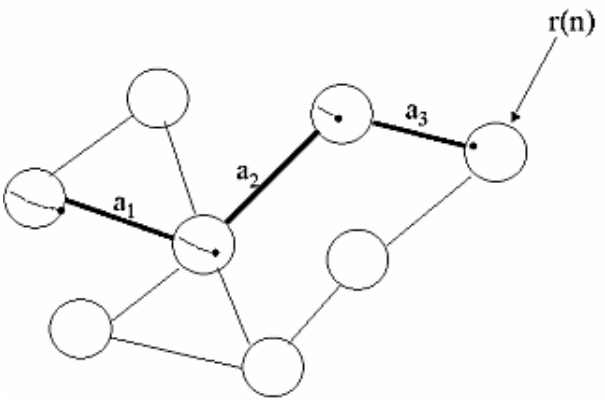


Figure 2 - 1 Q Learning (with decaying eligibility)

involve following the action with the highest Q' value for a given state; as a perfect measure of future reinforcement, each action selected thereafter in the context of succeeding resulting states would represent the same maximum Q' value.

In Q Learning, each state maps every action to a Q value that estimates the expected reinforcement to be received by following an optimal policy after that action. A variation of Q Learning also associates an eligibility value with each action. Whenever an action is selected, the eligibility factors of all other actions for that state are set to zero while the eligibility factor for the selected action is set to one. The eligibility value on unreinforced actions decays over time, decreasing the likelihood that these actions are pivotal in achieving the current reinforcement (Figure 2-1). When reinforcement is ultimately received, the Q values are adjusted for each state/action pair based on the level of the reinforcement and the eligibility of the pair. Thus, a sequence of states and action decisions made over time are rewarded or punished, with the greatest change occurring to actions representing the most recent activities.

Q Learning has been successfully used in the creation of controllers that learn how to manipulate a system over time. However, Q Learning is not as effective as a classifier system in those cases that require the learning of classifications as opposed to policies. Considerably more training is required in situations where a chain of events is not responsible for ultimate

reinforcement. Where supervised training is possible, a more traditional classifier system can use optimization techniques to increase learning.

Summary

This chapter presents an overview of the important ideas that were relevant for this particular study. These ideas were synthesized from the significant 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., agents and machine learning), human-computer interaction (negotiation), and computer-supported cooperative work.

The general question that was faced is how to make a computer-supported collaborative environment in which conflicts are handled automatically, without constant human/human negotiation. A related question asked whether it is possible to avoid the need for advanced conflict resolution by increasing awareness of potential conflict without overburdening the user with additional information. The four main examples of how to approach this problem were discussed in great detail. These four areas have contributed in greater or lesser degrees to the design of this study.

CHAPTER 3

DESIGN

The previous literature guided this study on the use of a hybrid artificial intelligence approach to minimize conflict in a shared CSCW environment. After determining that group conflict occurs in collaborative environments, a special computer-supported collaborative application was developed to examine methods for reducing group conflict through the use of interaction-support agents. Each user's agent "learns" to predict potential conflict situations. The agent then responds by directing the user's attention to the problem, offering advice, or, in some instances, interceding for the user. This software was then used in a special project. The project was divided into two phases: data collection without the agent, and a simulation of long-term interaction with the agent. The data collection phase used several small groups of humans to create a realistic model of human interaction and to determine the nature of conflict in this type of application. The agent phase verified the capacity of the agents to adapt to unique users over time. Unfortunately, long-term trials with humans were not possible at this time, so a simulation of user activity was used to verify the adaptability of the agents. The results of these two phases of the project will be reported in Chapter 4.

Chapter 3 describes the collaborative application system in greater detail. It considers the various intervention and learning components of an agent. After reading this chapter, the reader should have some knowledge of the collaborative application used in this study. The reader should also be familiar with the interaction support system used to detect and respond to conflicts.

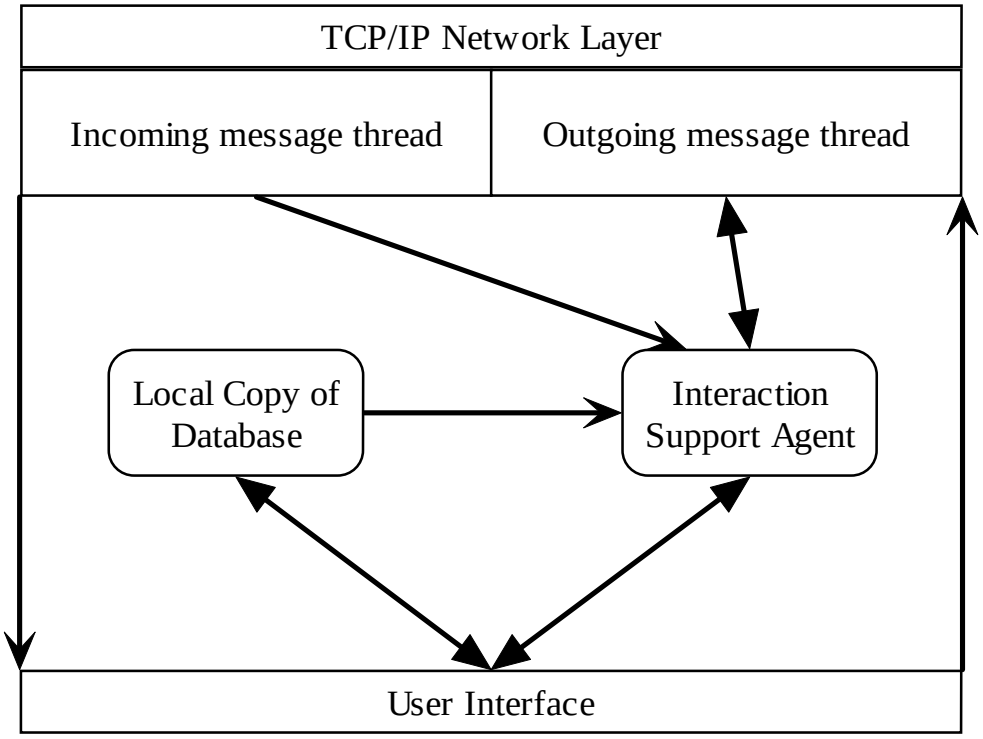


Figure 3 - 1 Collaborative Application Design Overview

System Overview

The major objective of this research was to determine how to minimize conflict within a computer supported collaborative application. In order to address this problem, a special computer supported collaborative application was developed. The specific context of this collaborative application was a shared calendar system that allows group members to schedule events (i.e., meetings) for specific time periods and on certain days. The collaborative application, therefore, consists of a graphical user interface, Internet communications support, a database management system, and an interaction support agent (Figure 3-1). The graphical interface contains a number of shared windows that allow groups to access each other's calendars and schedule meetings. Groups can schedule events by selecting a day and time on the calendar as well as enter additional information about the event. A simple distributed database maintains local copies of the events scheduled by each user. In addition, the collaborative

application contains an agent system that monitors activities related to both local and communicated user actions. The agents are designed to detect conflicts that occur while users are trying to schedule events with one another and to respond to the recognized conflicts in an appropriate manner. A more detailed description of the system components now follows.

User Interface

The first step in this research was to create a computer supported collaborative application that could be used to study group conflicts within a CSCW setting. A group calendar application, resembling an appointment book, was created to help members of a group schedule events. The group calendar software differs from a single-user calendar system in that all events are automatically broadcast to every member in the group, effectively creating a shared workspace. An integrated message service (Figure 3-3) allows both private and group communication among users. The main window takes the form of a monthly calendar (Figure 3-2). The calendar contributes to each user's awareness of group activity through a simple color-coding scheme. The days of the month turn yellow whenever a change is occurring. Once a change is reviewed, the day button for that date changes from yellow to white to continue to remind users that an event has been scheduled on a specific day. Users may view, modify, and add entries by selecting and clicking on a specific day in the calendar window. The interface works similar to a daily appointment book (Figure 3-4) in that users can switch back-and-forth between the month and day views. Every time an entry is added or altered, the change is automatically broadcast to all active members in the group.

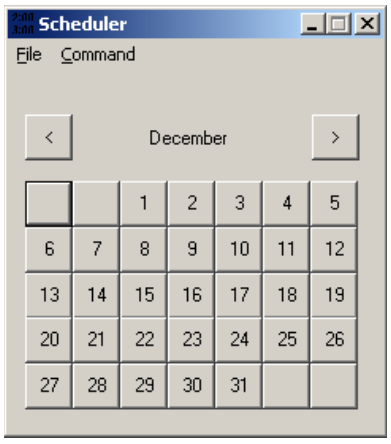


Figure 3 - 2 Main Calendar Window

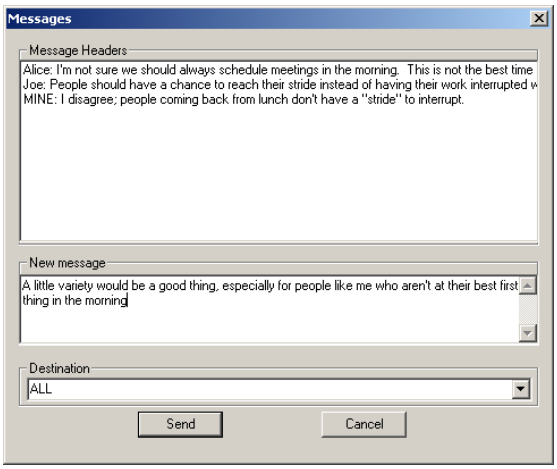


Figure 3 - 3 Message Window

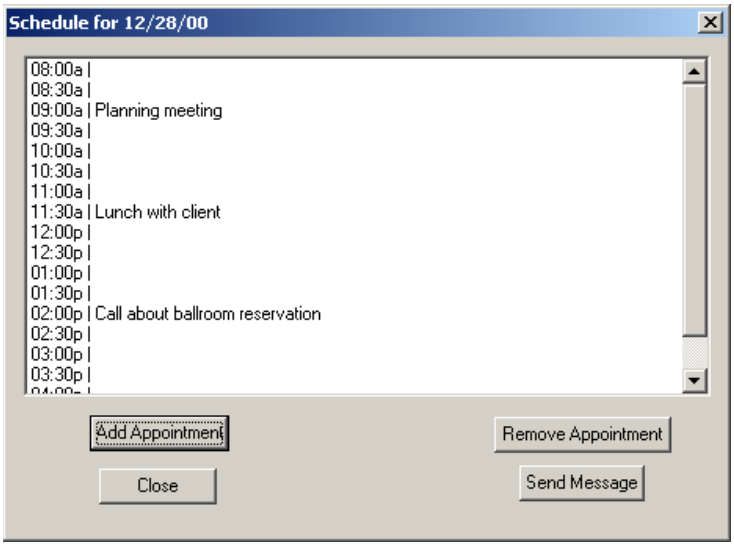


Figure 3 - 4 Day Schedule

There are several features within the shared calendar application that are designed to help users remain aware of one another's activities. For example, a log window records any changes that occur and appears automatically whenever someone in the group alters an existing event. A button corresponding to a day changes color whenever another user modifies an event on that day. Finally, there is a summary window (Figure 3-5) that provides users with an overview of all of the events that have been scheduled by the group. These features were designed to improve user awareness of group activities. Such features are considered important for successful collaborative communication (Procter et al., 1994).

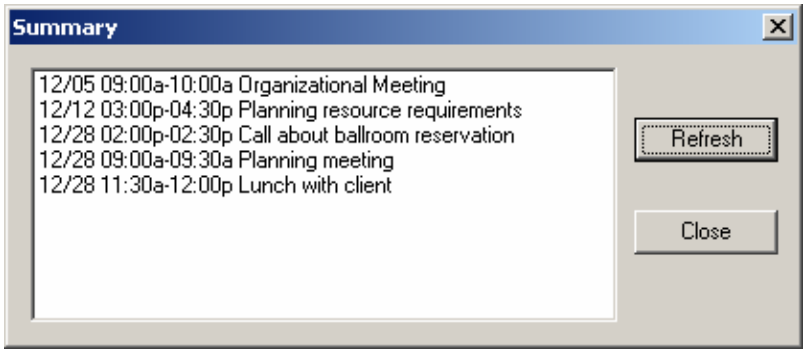


Figure 3 - 5 Summary Window

Distributed problem-solving is assisted by a central server that stores the names and IP addresses of every member in the group. Normally, all communication takes place directly between the members of the group via peer-to-peer connections. Rather than hardwire each IP address into the system, the application requires users to connect to a common name server at the beginning of a session. Each user configures the software to connect to the server's IP address and enters their name and the name of their group. This information is sent to the server; the server, in turn, responds with the names and IP addresses of the other active members of the group. In addition, the server informs the other active members that a new member has joined the group and provides them with the user's IP address. The server plays no other role in group communication or database management.

Once the group is connected, users communicate directly with the distributed application over TCP sockets. Actions taken by different users are summarized into message packets that are broadcast to other members of the group. The message types can be separated into three categories: entry-related, informative, and negotiation messages. Entry-related messages are sent whenever a user enters a new item or modifies an existing item. An entry can be modified by removing it entirely or by performing any combination of modification acts such as changing the description of the entry and changing the entry's start and end times. Informative messages are sent whenever a user opens and closes a day's entry window; these informative messages are

intended to notify the group about the focus of each user, allowing the user or agent to anticipate future entry-related messages concerning a particular date. Informative messages are also sent whenever a user joins or leaves the active group. Negotiation messages are described in the next two sections.

Low-Level Conflict Handling

In addition to the shared calendar interface, the application contains an agent system that is designed to detect conflict and offer advice on how to resolve group conflicts. Initially, the agents were designed to detect low-level conflicts such as those that occur whenever two users schedule different events for the same time period. In the first version of the application, a simple "agent" was developed to negotiate low-level, mutual exclusion conflicts caused by two or more users' making changes at the same time (Figure 3-6). The negotiation process uses a simple tie-breaking scheme inspired by the work of Lamport (1978) and Rabin (1982). Each agent involved in the conflict generates a pseudo-random number seeded by a user's identification number to generate a ticket; tickets were exchanged in conflict situations, and the highest ticket won the conflict. Initially, a conflict was said to occur whenever two users tried to add an event in the same slot or change the same event simultaneously; the loser of the conflict was required to rescind their changes.

The intelligent agent would try to improve its behavior by learning how to predict this form of low-level conflict. However, after analyzing several interactions among users of the application, it became apparent that too few conflicts occurred during the test sessions. Even when users were restricted to scheduling events within a single month, they rarely encountered conflicts. This was due, in part, to the large number of non-overlapping times that were available during the month (that is, potentially 12 x 30 = 360 non-overlapping times, which assumes, on

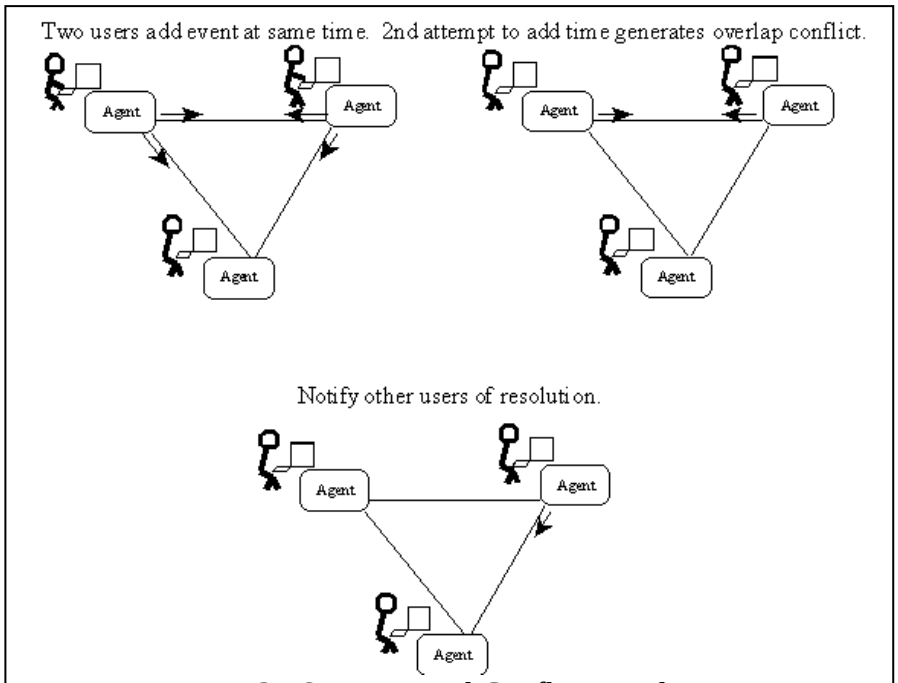


Figure 3 - 6 Low-Level Conflict Resolution

average, one hour meetings with only about 12 useful hours a day for meetings). Further, since the calendar application used a simple FIFO method to handle scheduled events, true conflicts occurred only during the few milliseconds when a scheduled event was being transmitted to the rest of the group. After considering other types of shared applications in which conflicts might occur (such as a shared whiteboard with a single resource), the definition of conflict was expanded to include other types of conflicting situations such as when one user schedules an event at a time that is “undesirable” for another user.

The change in the definition of “conflict” suggested that an “intelligent agent” might be used to learn a user’s preferences to aid in determining when a conflict had occurred or might occur. Rather than force the user to enter their preferences directly, a method was sought to have an agent “learn” preferences by observing the user’s actions. Since user’s actions were already being shared whenever the calendar was updated, it seemed logical to assume that an agent could be programmed to learn all users’ preferences with minimal effort or effect on the rest of the communication system.

Therefore, a second version of the intelligent agent system was developed and was programmed to "learn" user preferences in order to offer assistance to groups whenever different types of scheduling conflicts occurred. In the current version of the shared calendar application, a preference conflict is said to occur whenever one user schedules an event at a time that is "unacceptable" to another member of the group. Possible responses to the different types of conflict range from providing the user with advice about how to resolve a conflict to automatically negotiating a resolution of the conflict with the other user's agents.

Several different learning algorithms were investigated to determine if they could be used to detect the different types of conflict and select appropriate responses. User preferences, in particular, are unpredictable. In addition, appropriate responses may vary somewhat from one user to another. Reinforcement learning algorithms were of particular interest, since they have the ability to learn about unknown dynamic systems in real time (Littman et al., 1991, for example).

In order to use the reinforcement-learning algorithm for the calendar application, the event data had to be classified to fit the system's learning component. For example, events were represented as items with specific dates and times, covering a specific time period. To promote more general learning, times were classified as morning, lunch, afternoon, supper, or evening times. In addition, several user actions were characterized as being either constructive (such as scheduling a new event) or destructive (such as removing an event or otherwise changing it so that it no longer reflected the original user's preferences). This type of classification made it easier to recognize general trends in the practical preferences or actions of a user. Resources were also conserved. Since there are, on average, 360 time slots per month in the calendar

application, a learning algorithm must try and contend with a minimum of 720 states. The simplest implementation of Q learning uses a lookup table to track the estimate of future reinforcement (Q value) for each possible state/action pair. Assuming five possible actions, the table needs 3600 Q values. If the state is defined by the last two user actions, this number grows to 2,592,000. In general, if each piece of data used to define a state has *a* different possible values, the space requirements for a number of machine learning methods would be $O(a^n)$ for n data items. The more information used to recognize or predict conflict, the more space is required, which also impacts learning time.

A hybrid learning system, on the other hand, is able to combine the strengths of several different learning methods. One of the weaknesses of using a single machine learning method is that most problems are too complex for a single approach (Goonatilake et al., 1995). Another reason for rejecting a single learning system is that it is often resource intensive. Since Q Learning methods tend to require a large amount of resources (Baird, 1995), it is important to find ways to reduce the processing time required to compute the various Q values.

The particular hybrid learning system that was used in the Calendar application was designed as a two-step hierarchical system in which simple inferences are passed to the next, more general level. The classifications computed at each stage of the process provide input to the next level. This was done, in large part, to minimize processing time and is similar to what occurs in an expert system that uses rule sets and hierarchies to accelerate inferencing. Also, by characterizing states as different layers of knowledge, going from specific to general, it is possible to eliminate some of the thrashing that occurs in some learning algorithms because conflicts tend to disappear as the more general layers are invoked. As a result, a hybrid learning

systems seemed to be uniquely suited for learning conflict situations that occur when scheduling events.

One of the major issues that needed to be addressed for this particular hybrid learning system was the makeup of the different layers. There are several different ways to define the layers, and it is interesting to review the different approaches. For example, one layer might be designed to learn user preferences, while the second is used to learn the user's next most likely action. A third layer could then reason about which actions lead to different types of conflict and infer an appropriate response. An alternative solution is to have the agents classify the next action from a chain of previous actions, bypassing explicit user preferences entirely. Yet a third solution is to connect user preferences and recent actions directly to a learned anticipation of conflict, bypassing the third layer described in the first example.

The latter solution has two benefits. Encoding specific user actions in terms of general user preferences creates a noticeable reduction in the number of distinct states that have to be processed in the second layer. On the other hand, the very nature of a dynamic learning system requires that it is initially and occasionally wrong. Any layer that responds incorrectly to an event will probably cause an incorrect response to occur in the inferencing that takes place in some future layer. However, if the system can "anticipate" a conflict during one of the early stages of learning, then the action prediction layer may be unnecessary, and fewer layers will be required to get reasonable responses from the system. Thus, a learned "anticipated" layer approach was designed and implemented for this particular calendar application.

The current agent system contains three modules: a preference learning module, an action selector, and an action planner (Figure 3-7). The agent first learns preferences from the actions of the users. Preference learning can be done independently of the other two layers

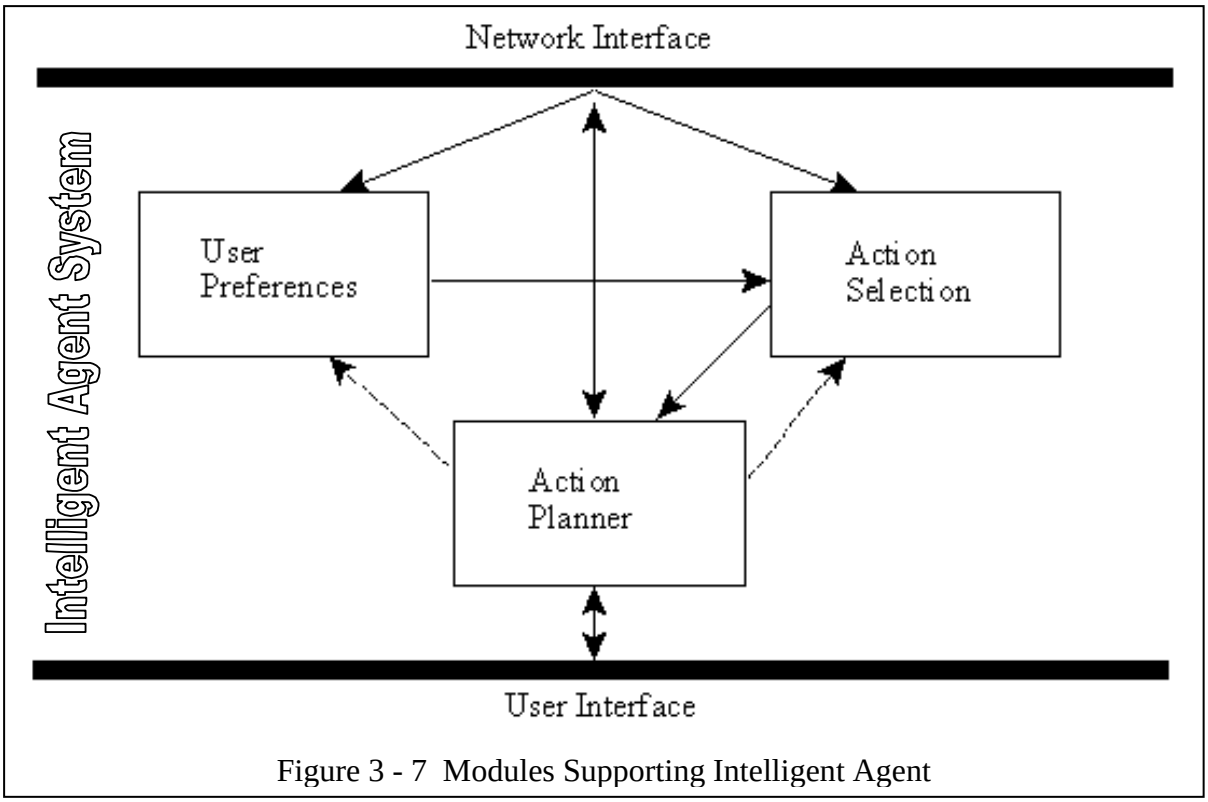


Figure 3 - 7 Modules Supporting Intelligent Agent

because it is really a separate learning problem. Preference learning is essentially a method of summarizing past actions and recognizing more general patterns.

The action selection module examines the user's preferences and the last action taken by a user and translates these into an appropriate agent action. The agent actions at this layer are encoded as a simple value representing a type of response. In addition to recognizing and identifying a response to different conflicts, the action selector "learns" to recognize events leading to conflict.

The action planner module consists of a set of plans that are associated with the different action types proposed by the action selection module. Once the action selection module has chosen an action type, the action planner tries to prescribe a series of steps and actions that might be appropriate to overcome the current or potential type of conflict. For example, one proposed action of "INTERRUPT TO WARN USER" generates the following sequence of steps: DETERMINE STRENGTH OF USER PREFERENCES, SELECT MESSAGE OF SIMILAR

STRENGTH, and IMPOSE MESSAGE BOX WITH MESSAGE ON USER'S DISPLAY. By separating the agent's responses into action type and action steps, other machine learning algorithms can be used in this system with minimal modification. The action planner can be replaced with other types of Artificial Intelligence tools such as case-based reasoners (Kolodner, 1993) that can customize responses to individual users. For the present, this layer is simply part of an overall software design module. The software analyzes the situation that matches an action type and infers a fixed sequence of steps.

User Preferences

As previously stated, the user preference module is designed to learn various characteristics about the user. More specifically, this module tries to determine the days and times a user prefers to schedule certain types of events. The intelligent agent makes these decisions by examining the user's repetitive selections for different dates and times within the Calendar application. As previously stated, user preferences are defined as time elements such as a day of the week, a time of the month (early or late), and a time of day (morning, afternoon, evening, etc.). To classify user preferences, the intelligent agent matches the user's specific actions (e.g., adding a date with Joe for July 3 at 8 p.m.) with the user's preference values (e.g., the user prefers Tuesday evening, early in the month). The user preference module is also able to make certain generalizations about previously classified events such as UserA prefers to schedule events in the evenings.

The agent currently uses a simple reinforcement technique along with a generalization algorithm to learn user preferences. User actions are classified as neutral, constructive, or destructive. An example of a neutral action is a user opening a window on a particular day. A constructive action consists of an event such as adding a new item to the schedule. Destructive

actions include events such as removing an item from the schedule. Modification actions may be broken up into both constructive and destructive actions that involve changing the time of an event. Each time a user performs a constructive action, the user preferences module records this as evidence of a positive user preference; likewise, a destructive action is recorded as evidence of a negative user preference. Anomalous actions, such as unrepeated actions that contradict past behavior, are handled by computing a weighted average of past and new evidence. A strong preference requires as many as three consecutive observations that the preference has changed before this is reflected in the new preference data.

The reinforcement algorithm used in the calendar application takes the weighted average over time of a reinforcement signal determined by user actions. This is a common technique used in other reinforcement learning systems (e.g., Watkins et al., 1992). Because this layer is concerned with classification rather than prediction and assumes that user actions occur independently of each other, only instantaneous reinforcement is considered. Reinforcement is calculated entirely from the type of user action inferred: 0 for a neutral action, -1 for removing another user's event, +1 for adding a new event, between 0 and +1 for more ambiguous constructive actions, and between 0 and –1 for more ambiguous destructive actions. Ambiguous actions include modifying the time of an event so that it overlaps with the original scheduled event. Such an adjustment may be less a reflection of preference than the need to make adjustments to an already crowded schedule. The new preference is calculated by combining the previous measure of preference with the reinforcement signal:

$$p_i(w,d,t) = \alpha\, p_{i-1}(w,d,t) + (1-\alpha)\, r$$

where r is the reinforcement signal and α is a forgetting factor. Values of α close to 1 indicate past experience is more important than new experiences, and values close to 0 indicate that new

experiences are more important than past experience. Two types of α values are used, α_e for encouraging an existing preference or establishing a new one ($p_{i-1}(w,d,t) = 0$ or has the same sign as r), and α_c for contradicting an existing preference.

Two "forgetting" factors are included in the calculation for user preferences to insure that the confidence levels grow slowly and, at the same time, that any contradictory evidence is accurately reflected by a change in preference. Three strong user actions (r=1.0) are required to confirm a user preference with confidence 0.5 if α_e is 0.7937; six confirming actions will achieve a confidence of 0.75. A value of 0.756 for α_c will not allow a single contradictory experience to undo a strong preference measure, but three such contradictory actions will turn a preference level of 1.0 into –0.136. Preference strengths below 0.75 will change if less evidence is present.

A user action, translated into a constructive or destructive action for a specific time of month, day of the week, and time slot, provides direct evidence for a user's preference for a particular part of a month. A generalization algorithm is then used to try and capture preferences of a broader nature. For example, a user action may indicate that a person prefers to meet on Monday evenings and at the beginning of the month. However, this simple conclusion might actually be part of a broader preference for meeting at night. Further evidence is needed to verify these more general preferences; the generalization algorithm starts with a low-confidence measure that such a preference might exist and increases the confidence measure as additional evidence is found.

Since the system is concerned with learning user preferences for different dates and times in a calendar application, the term "dimension" is used to refer to any single measure of a particular action's place in time, such as the day of the week, the week within the month, and the general time of day (morning, afternoon, evening, etc.). A specific user action may indicate a

preference for an intersection of any subset of these dimensions. Instead of simply indicating a preference for a specific day and time, a user may also indicate a preference for a day of the week or a combination of time of day and time of month, etc.

Whenever a user action occurs, the system tries to generalize this event by adjusting the weights of all neighboring times and days. The greatest adjustment occurs for those neighboring entries that share the same values along one or more dimensions (e.g., yesterday at the same time, or $p_{i-1}(w,d-1,t)$). More specifically, the algorithm works in the following manner:

Let d_i represent the distance of a preference datum from the observed event, measured along only one of n dimensions. The positive or negative impact of the observed event on the preference datum degrades both with overall distance and the smallest distance to an item that matches along one dimension.

$$\iota = 2 \cdot 0.05^{\left(\sqrt{\sum_{i=1}^{n} d_i^2} + \min_{i=1}^{n} d_i \right)}$$

The small decay factor of 0.05 causes the impact to fall off quickly towards zero as the distance between dimensions increases. Combining overall distance with the minimum distance along a single dimension causes the results to fall off far more quickly for data that do not share a common set of attributes with the new event.

The combination of specific evidence with the generalization algorithm means that the system delivers a response based on past behavior, regardless of whether the time of interest has been referenced previously. Previous literature (see Chapter 2) discusses a number of different negotiation methods that can be used to exchange user preferences among agents. This particular problem was avoided in the Calendar application because every user action is transmitted to all the members in the group. Since the user's agent has access to group information at all times, it

can estimate another user's preferences for a particular event without requesting additional information from any other user.

Preference information for a specific time slot is obtained by looking at the preference level, or weighted average, calculated for a specific week, day, and time, $p_i(w,d,t)$. The preference levels can be classified as "highly desirable," "moderately desirable," "no opinion," "moderately undesirable," or "highly undesirable." Preference levels are maintained between –1 and 1. Because the generalization algorithm ensures that the entire set of preference levels will be affected, if only slightly, by every constructive and destructive user action, the preference category of "no opinion" is applied to all magnitudes of less than 10^{-6}. Moderate preferences are defined as those preference levels whose magnitudes are between 10^{-6} and 0.2, inclusive. Strong preferences refer to any levels above 0.2. These preference classifications are the results of calculations from the rules that reside in the user preferences layer of the system.

Action Selection

Once user preferences are determined, the system can respond to a user action if it represents a conflict situation or a precursor of a conflict. System responses include doing nothing, drawing attention to an activity that might be a possible source of conflict, warning or advising the user concerning a potential conflict situation, automatically notifying other users of undesirable behavior or otherwise acting on behalf of the user, and taking steps to prevent undesirable actions. A Q learning algorithm is used to obtain the appropriate response that is associated with a particular state, as represented by the user's preferences, involvement, and recent activity.

Recent activity includes items such as the type of activity that occurred and whether it was constructive, destructive or informative.

Involvement is a weighted average that allows users to be ranked as very active, somewhat active, or largely inactive; the average is calculated as the time between messages generated by a user's behavior divided into 30 seconds (an assumed "minimum" response time), so involvement, or "activity level" as it is called in the algorithm, is a value between 0 and 1, approaching 1 for users who are adding or changing scheduled events as quickly as possible. The measure of involvement is reduced to "virtually inactive" (average participation every 30 minutes or more), "low activity" (participates every 10-30 minutes), and "high activity" (participates more than once every 10 minutes).

Preferences are provided by the preference-learning system after being classified as one of the five categories mentioned in the previous section. Thus, real-valued inputs like the preference weights are combined with fine-grained inputs to create a finite number of discrete states.

For example, UserA may use the Calendar application to schedule a meeting with UserB without consulting UserB's calendar. If UserA schedules a meeting on a Monday and UserB does not like Monday meetings, then a conflict is said to have occurred. The action selection system should eventually learn to respond to the scheduling of Monday meetings between UserA and UserB. UserB's agent may learn to notify UserB of the undesirable event, and UserA's agent should begin advising UserA to stop scheduling meetings on Mondays. Eventually, the Q learning algorithm can associate UserA's viewing of a Monday date with a negative response, allowing both users' agents to anticipate the creation of undesirable events before they happen. In this way, an informative action, such as a notification of a user exploring a certain date, becomes a signal for an undesirable constructive action that could lead to conflict.

In order to learn appropriate responses to situations, the Q learning algorithm requires inputs from the system in the form of rewards and punishments. The action selection module drives the Q learning software and also determines when to punish inappropriate system responses and reward appropriate system responses. An example of an inappropriate system response is the decision to do nothing at a point of recognizable conflict. The action selection module defines a “hard” conflict as any modification to the time or existence of a scheduled event by a user other than its creator. A conflict can also occur whenever a negative message from a user causes the creator to modify an event, although this is more difficult to detect than other types of conflicts. Potential conflicts are any actions that create or change events that cause those events to be incompatible with user preferences; these markers of potential hard conflicts are referred to as “preference conflicts”. Whenever a conflict is detected, the module sends out a small negative reinforcement ($r = -0.1$) to the Q learning system. This negative reinforcement is then propagated to the various state/response pairs, reducing the Q values that determine response selection for responses leading to the conflict; the “eligible” state/response pairs are reduced relative to their proximity to the conflict event in time. The negative reinforcement is increased ($r = -1.0$) if the Q learning system responds to an obvious conflict by proposing that the agent do nothing. If the Q learning system recommends an inappropriately strong response whenever the user’s preferences do not significantly outweigh the preferences of others, such as blocking a change to the schedule, then a negative reinforcement ($r = -1.0$) will be applied again.

In addition, positive and negative reinforcement may be directed toward a state/response pair depending on the user’s settings. If the module determines that strong responses are required, it will ask the user for permission to automatically intercede for the user. These types of strong responses do not only inform or advise the user; they actually block or cause changes to

occur without user interaction. For example, a user with an inflexible schedule may choose to accept a strong response from the agent to prevent any possibility that any other user will make a change to the schedule. The user allows the system to make changes automatically or to restrict the actions of an agent. These user-defined settings extend the definition of appropriate and inappropriate responses to include more feedback from the user. Strong responses are only rewarded (r = 1.0) when explicit user approval is given. They are punished as inappropriate (r = -1.0) if the user explicitly rejects them.

The Q learning system ranks the four different types of responses to further determine their appropriateness. Warnings or explicit advice, say via a pop-up window, are considered stronger than the action of notifying the user of the activities of other users (subtly through the interface or by unobtrusive messages). If the Q learning system advocates a strong response that is inconsistent with the current state (e.g., warn the user about scheduling an event on a day that is acceptable to everyone), then the state/response pair that is responsible is punished (r = -0.5) and the response is weakened.

The responses of other users' agents can also be used to calculate reinforcement. If another user's agent automatically rejects a previously entered event, then a conflict occurs and any response to the original user's action, other than rejecting it, is inadequate. It should be noted that punishment of inadequate responses allows the agent to avoid unnecessary communication overhead by responding to the offending action immediately rather than waiting to be notified by another agent.

The Q learning algorithm in the Calendar application updates states through a combination of direct feedback and a variation of a dynamic programming technique. The dynamic programming technique allows Q values to impact more recent states *before* a conflict

is recognized and direct reinforcement provided. If a user consistently moves through a state before reaching a conflict situation, a response to the conflict will gradually precede the conflict itself, as previous negative reinforcement propagates back to notify the system that doing nothing during this earlier state is a bad thing. This can happen even in the absence of repeated reinforcement because the Q values of the next anticipated state represent an estimate of future reinforcement that is used in the training of previous Q values. Any response that is effective at eliminating conflict will be selected as soon as the conflict can be unambiguously detected.

Action Implementation

There are several different types of responses that are generated by the action selection module: notifications, warnings, automatic restriction of certain user behaviors, and automatic response generation. Depending on the circumstances surrounding an event, the actual steps that the agent takes may differ from one invocation to the next. A simple rule base system establishes the appropriate plan or sequence of actions for a given situation. Once a plan is generated, the system automatically performs those actions. A description of the different types of actions now follows.

Notification actions are made in response to another user's actions or a response to an action performed by the agent's user. The agent may respond to another user's actions by displaying or flashing a window that shows the user’s current and past changes. A marker can be placed on the calendar to indicate times of possible preference conflict.

A warning type of action requires the system to use a more forceful intervention strategy such as the display of a pop-up window that interrupts normal user behavior. One of two buttons is used to dismiss the window, one of which allows the user to indicate that the warnings are

annoying. A small negative reinforcement is sent to the action selection system if warnings are not appreciated.

A restrictive action is designed to initiate negotiation between two or more agents. In rare cases, two agents may add or modify an event, simultaneously. Whenever this occurs, a restrictive action is used to force a negotiation. The same negotiation function that handles conflicts between agents is used in this situation. The only difference between the two types of restrictive behavior is that in cases of restrictive responses, the system takes action after anticipating a possible user conflict as opposed to after a conflict has occurred. Users are given the option of receiving notification of restrictive actions with or without being asked first. In cases where the agent's own user triggers a restrictive response, the agent acts automatically but gives the user the opportunity to override the responses.

Automatic responses are similar to restrictive actions except that they are done automatically without any negotiation with other agents. Modification actions include adding events removed by another user, removing events added by another, or restricting the user’s actions that create conflict. The system responds to modification actions which involve its user by sending a message back to the user that the item should not be changed and why. Since modification actions are deemed extreme, the user must give permission to the agent to make these types of actions. In cases where the system starts to anticipate restrictive or modification actions, the action planner may elect to send a warning message to the suspected user indicating the strength of an agent's determination to protect the time slot.

Conservative actions are designed to increase user awareness of the activities and preferences of other users, particularly in those cases where there is limited bandwidth and/or the complexity of the task reduces awareness. Active responses are designed to protect the agent's

user from actions that will cause a conflict. Active responses prevent a determined user from scheduling intolerable events or changing mandatory events directly.

Agent-Training Tool

Research has shown that the slow, incremental learning that usually takes place with Q learning systems can be improved with the aid of a simulation (Kuvayev, 1997). This particular approach was used to decrease the amount of experience required by the Q Learning portion of the system before it could be useful within the Calendar application. It was determined that a significant period of time would be required for the system to acquire a reasonable response model without the aid of a simulation. The simulator also enabled the creation of controlled experiments for the evaluation of agent learning.

The simulator uses a variation of the shared calendar intelligent agent to train the three modules used in the Intelligent Agent portion of the system. More specifically, it is designed as a separate program that can be used to train the action selection module to adjust to different user personalities. For example, the agent trainer can prepare two types of intervention style that can be used to respond to conflict. In order to do this, different reinforcement patterns are used: one representing a conservative approach and the second representing an aggressive approach. The more conservative model is used in the initial stages of the simulation, while the more aggressive model is used whenever a user's activity level and rate of conflict exceed specified thresholds. The conservative model is trained to intervene with minimal intrusion by using only two of the four active responses. The aggressive model is trained to use the automatic responses in the presence of extremely strong user preferences. A user can save time if he/she allows the agent to automatically block events that are scheduled at undesirable times. A user is considered

aggressive if he or she alters the events of other users or re-schedules an event altered by another user.

Each virtual user in the simulated system is represented by an activity level, a selfishness factor, a response probability value, a response delay value, a set of explicit preferences, and a list of warnings received from other users or agents. The activity level denotes how frequently the user schedules events and is used to calculate other values such as the response probability value. The selfishness factor represents the tendency of the user to ignore warnings and focus on his/her own agenda. The response probability value defines a user's aggressive behavior such as the tendency to remove an undesirable event created by another. The response delay value represents the user's ability to recognize undesirable activities by other users. The simulation draws on the user's explicit preferences to determine when to schedule events and which events are considered undesirable; this information is not available to the intelligent agent, which must learn these items through observation. The agent's advice is encoded as a list of warnings that alter user behavior. For example, the lower the degree of a user's selfishness, the more likely a user will "remember" past advice about other users' preferences. Warnings gradually reduce the selfishness value of a user to simulate the user's growing awareness of other users' preferences.

The simulation uses an action queue, which is prioritized according to the time that the actions are scheduled to occur. As each action is processed by the event-driven simulation, new actions may be generated. Virtual users decide to schedule a new event based on their activity level. An event is randomly scheduled based on the user's explicit preferences; any warnings received from other users are also considered, depending on the "selfishness" factor of the user. Specifically, a time is rejected at a rate of (1 – preference_level) or, whenever the system matches a warning from another user, at a rate of (1 – selfishness), where preference_level is in

the range [-1,+1] and selfishness is in the range [0,1]. Negative preferences are always rejected, and perfect selfishness (1) ignores all warnings. A new event may be added to the shared schedule every 30 seconds for a particular user. New events are generated at a rate of every 60 seconds for 75% of the events, 5 seconds for 20% of the events, and 10 minutes for 5% of the events. However, the rate that events are generated depends on a user's activity level. For example, if a user's activity level indicates that events should be added frequently to the shared schedule, then such additions may be made every 30-60 seconds, extending to 10 minutes on rare occasions. The action queue is a priority queue that delivers the events in the appropriate order so that the simulated time never decreases.

The virtual users are given an opportunity to respond to these actions, which, in turn, can lead to further actions, etc. If one virtual user generates an action by adding an event at an undesirable time for another user, the system simulates the other user deciding whether to delete the offensive event in direct proportion to the level of dislike recorded in the offended user's explicit preferences ($- p_{w,d,t}$ is between 0 and 1). Removal of the undesirable event is scheduled to take place between 30 seconds and one hour after an event occurs, simulating the user's activity level, aggressiveness, and awareness of the actions of others. A base time of $3600^{1 - 2\ response\_factor}$ is added to a pseudo-random number in the range [0, response_delay). The response probability value (i.e., response_factor) is used in the computation only if the virtual user has a low level of aggressiveness, resulting in a significant delay in reaction time.

Virtual users respond to the removal of an event that they created according to their response probability values. If users choose to respond, they reply by recreating the event. The response delay is calculated using a ratio between ½ to 1½ of the user's response delay, simulating a time lag between the actual removal of the event and the user's awareness of and

response to that removal. This delay, like all other delays, assumes that a user needs a minimum of 30 seconds to become aware of a situation, decide what to do, and use the user interface to respond to the situation.

The simulation is designed to respond to the agents' actions on behalf of their user. Advice and subtle cues either reduce response time or are represented as warnings. Of course, a "selfish" virtual user may ignore warnings, but the simulation assumes that the users are not entirely self-absorbed. Each warning reduces the selfishness component of the virtual user, forcing the user to gradually consider the interests of others.

One of the system's responses to a conflict or potential conflict is to make the user aware of the fact that the selected action or current situation could lead to conflicts. If this response is the result of the agent's user's actions, it takes the form of a light warning. If the response is the result of another user's actions, then the offending user's simulated response time is temporarily reduced and its responsiveness is temporarily increased to reflect an increased awareness of the activities of others.

Stronger warnings are handled in much the same way, except that the user who is warned against an action by its own agent is given an opportunity to cancel the action. The system simulates this behavior by adding to the user's list of warnings, then reevaluating the acceptability of the user's action in light of the new information.

The simulation attempts to simulate human-like responses. The major reason for developing the simulator was to exercise the learning agent and verify that it could respond to events in a reasonable manner. Also, the simulator was used to pre-train the agents to increase system performance.

CHAPTER 4

DATA ANALYSIS AND RESULTS

Introduction

The data collection and analysis was performed in two stages. During the first data collection stage, groups of student volunteers were asked to engage in a scheduling task using the collaborative calendar application. As students performed the task, the system recorded the different patterns of behavior and captured individual scheduling preferences. This data was then analyzed to determine the correlation between user preferences and scheduling behavior. The system also captured the different behaviors that characterized conflict. Once this phase of the project was complete, an agent-based system was designed to 'learn' user preferences and respond to behaviors that might lead to conflict; the agent system was developed using information gained from the human subjects' study. A series of experiments were then developed to test the accuracy and adequacy of the learning component of the agent system. Chapter 4 describes the data obtained from the human subjects as well as the experiments with the learning system.

Human Preference Studies

A number of students were asked to use the group calendar application to help develop different user models that could be used to initialize the intelligent agent system. A small group of undergraduate students were recruited for the project. All student volunteers were enrolled in computer science courses at Sam Houston State University between Spring semester 1997 and Summer, 2000. It became obvious after Spring 2000 that a record of the user's scheduling preferences was necessary for the creation of the different user models. While a total of thirty-

two volunteers used the calendar application, only the last eight volunteers (i.e., those students enrolled in Summer 2000) were asked to explicitly record their scheduling preferences. Of these eight volunteers, four were relatively naïve users who were enrolled in a computer literacy course; while four were enrolled in one or more advanced computer science courses and had presumably more computer experience. All volunteers were told that they were participating in a study of a shared calendar application, and that their responses were being recorded for analysis. Unfortunately, one group of volunteers seemed to know significantly more than others, so only the data collected from six volunteers were used to inform the development of the user models for the agent system. In cases where preference data was not analyzed, data from up to thirty volunteers is still available.

Each student who participated in the study received a copy of a scenario (see Appendix A) in which students were asked to use the calendar software to arrange for a series of contrived events. Groups were further instructed to use their real schedules in determining when a particular event could be scheduled. One-hour sessions were then arranged in which groups of two or three students were asked to use the software to complete the assignment. A log was kept of computer sessions for later analysis.

Before each experimental session, subjects were asked to enter their preferences for different days and times. The system tracked the number of constructive and destructive actions that occurred during the experimental sessions for different types of days and times. Constructive actions (such as scheduling an event) were seen as evidence of a positive user preference, and destructive actions (such as deleting another's event or changing it to a different day or time) were seen as evidence of a negative user preference.

The initial plan was to use the preferences specified during the human preference studies to direct the input into the agent system. However, an analysis of the data from the human subjects' sessions showed individual users ignored their initial scheduling preferences between 7% to 66% of the time, with an overall average of 56%; these statistics indicate that there was a great deal of variation between a user's actions and their stated preferences. After talking with subjects, it became clear that the system was not adequately identifying contradictory user preferences. For example, a user might list a preference for afternoon meetings and, at the same time, indicate an aversion to Monday meetings. If group members scheduled a meeting for Monday afternoons, the user might reject the Monday meeting, even though it was scheduled at an afternoon time. The discovery of this problem led to the development of a new user model that captured user preferences for specific combinations of weeks, days, and times. Furthermore, a generalization algorithm (described in Chapter 3) was developed to help 'infer' a user's more general preferences (e.g., a user who likes Monday, Tuesday, Thursday, and Saturday afternoons can probably be said to like afternoons, in general). The preference learning algorithm, including the generalization component, is called the "generalizer."

Figures 4-1 and 4-2 show the number of actions that contradicted user preferences for the six users who provided explicit preference data. As mentioned earlier, the preference data collected was not sufficiently specific, so Figure 4-1 reports the number of actions that contradict all applicable preferences, whereas Figure 4-2 reports the number of actions that contradict *at least one* of the applicable preferences. Both figures compare stated user preferences with the preferences learned by the generalizer from actual behavior data. The percentages reflect the degree to which stated and learned preferences can predict user behavior. Note that the agent

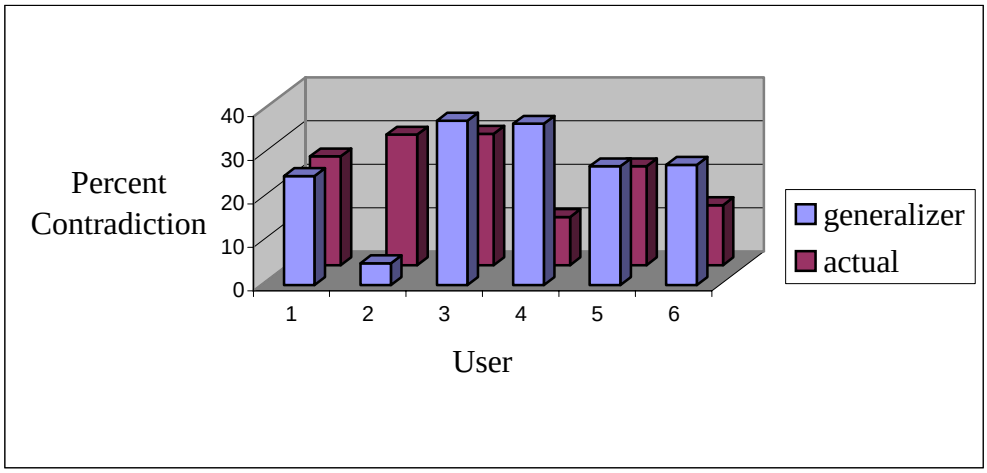


Figure 4 - 1 Percentage of Time that Behavior Contradicts Preferences
Learned Preferences (Generalizer) vs. Actual Stated Preferences

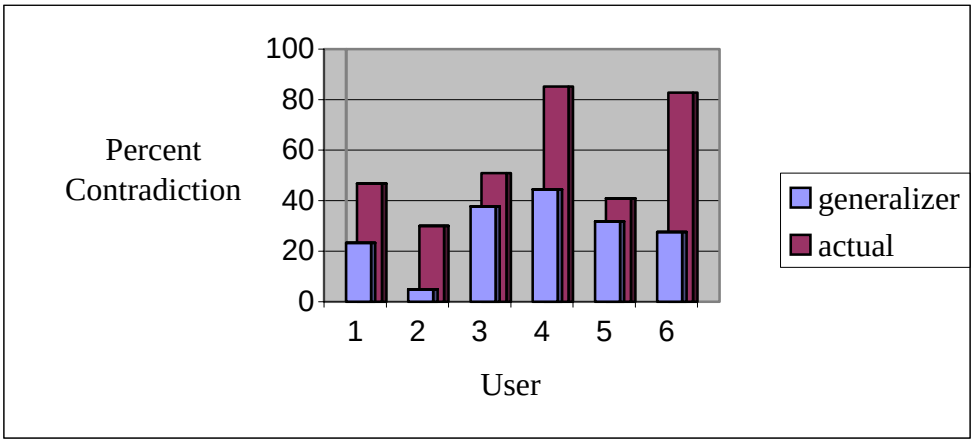


Figure 4 - 2 Percentage of Time that Behavior Contradicts Preferences
(using more conservative preference matching algorithm)

learning system with the addition of the generalizer is competitive with a system that has knowledge of the user's true preferences.

The data gathered from the human user experiments also indicated that groups were either dominated by a single, active user or were distributed in both their work and communication. After observing this phenomenon in the preliminary studies, hard data was collected from the last twelve groups that participated in the study, of which data from ten groups was found useful (see Appendix C). For example, the most active user in one group scheduled eight times more events than any other user in the group, and was 1.75 times more

active than the least active user in the group. Four of the groups had a single user who scheduled twice as many events as other members. Each of the four groups showed a drop in activity level similar to that described in Stasser and Taylor (1991). On the other hand, the most active users in two of the groups were only 1.43 times more active than the least active user. Given this data, a "hierarchical" group was defined as one in which a single user in the group was at least 1.75 times more active than other users in the group. All other groups were considered to have uniform distributions of activity, although most observed groups had at least one member who was noticeably less active than the others. Since all other members appeared to have similar activity levels, they were said to exhibit "democratic" behavior.

Figure 4-3 compares the 'activity levels' of the various groups. Activity level was defined as the percentage of schedule-changing events made by a user during a single session (the ratio of a user's actions, both positive and negative, to the total number of actions taken by all members in the group during a session). Figure 4-3 seems to indicate that all groups should be characterized as being "hierarchical." Upon further inspection, however, only two groups showed differences in all three categories of actions (positive, negative, and message communication), indicating that a single leader was responsible for most of the activities for that group. Overall, the most active user in these two groups was almost twice as active as other users in the group. On the other hand, five of the groups had less significant differences in their activity levels, indicating that they demonstrated more "democratic" behavior. Messages were assumed to mean that someone within the group was requesting or communicating information (when the student subjects took them seriously).

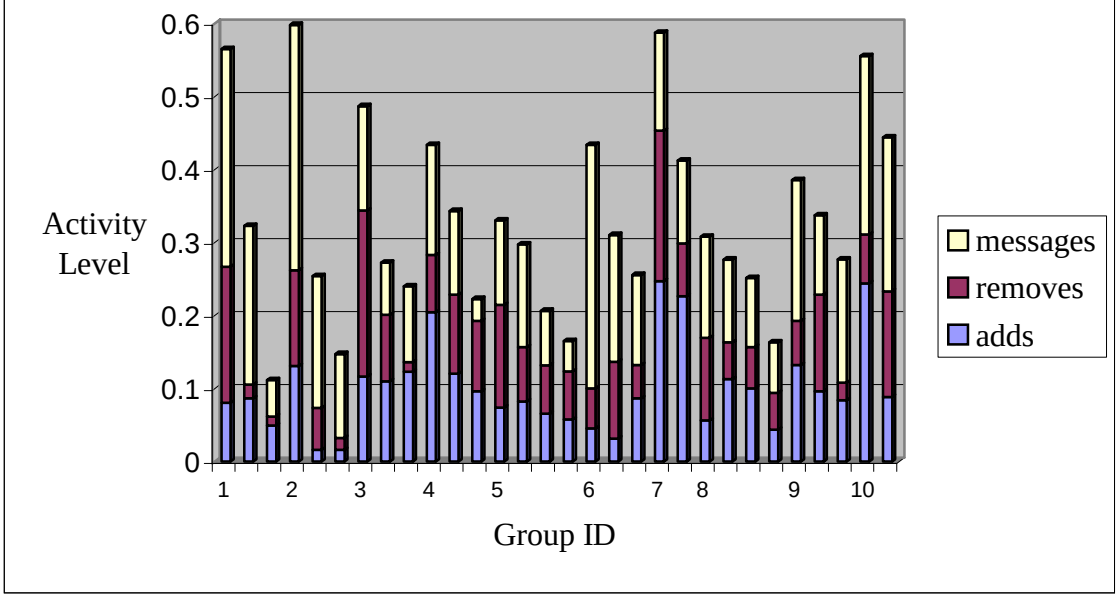


Figure 4 - 3 Relative activity levels within ten groups

A desire to support the different needs of these two types of groups led to the development of two idealized response models for the different types of activities in the groups and was used to guide some of the experiments conducted with the agent learning system at the end of the study. These experiments are discussed at the end of this chapter.

Although the agent seemed to adapt quickly to user preferences, it soon became obvious that testing the effectiveness of the agent's ability to learn a conflict response model could not be done using 'real subjects.' Thus, a series of off-line experiments were conducted to determine the effectiveness of the agent system in learning preferences and avoiding conflicts. Another question raised during the development of the agent system was whether the agent should learn 'only' from user interactions, or whether it should be provided an initial model that would be adjusted during the learning sequence. Several experiments were conducted to determine the best approach and are described below.

Experimental Procedures

An agent learning system was developed and tested with user models informed by the data gathered in the previous study. The following experiments were performed to verify that the agent program could anticipate user preferences, given a preference model, and could use this model in combination with immediate state information to select an agent response that helps reduce conflict among users. The user models within the agent-learning module were adjusted to provide for a variety of interactions. The agent-learning program was set to run at either 80 or 160 simulated hours. A simulated hour consisted of between 150-200 responses to messages (corresponding to the sharing of information between users in a group), a message being any activity including the addition, deletion or modification of a scheduled event within the calendar application. The reason for selecting 80 and 160 ‘hours’ as the two time periods for the learning module was because initial observations indicated that most ‘learning’ took around 40 simulated hours to stabilize, and twice that amount of time whenever an intervention was injected within the middle of the experiment. Measures were then developed to determine the accuracy of the agent-learning module to detect user preferences and provide appropriate intervention strategies. The setups for the different agent-learning experiments are now discussed.

For each experimental run, the system recorded the percentage of events correctly classified as compared to the user model’s true preferences, and the number of conflicts detected. Conflict rates generally stabilized within forty hours; that is, whenever there were no changes in the users’ behaviors. Whenever a change in user behavior was introduced, conflict rates tended to double (because the system had to re-adapt).

Preference Learning

Preference learning was important in all experiments. Each experiment was provided with a set of three different types of user models. Each user model represented the unique preferences and attitudes of an individual user. Three user models were used in each preference training experiment because this was the typical size of a group in the human experiments, and it allowed for easier comparison among the different experiments. Each user preference model consisted of a table of individual preferences, all of which were assigned a number between –1 and 1. For easier analysis, a series of three categories of preferences were created to represent the different levels of user preferences; a positive number indicated a positive preference for that category, and a negative number indicated a negative preference for that category. For example, *extreme* preferences were encoded as either a +1 (i.e., highly desirable) or a –1 (i.e., highly undesirable). *Medium* preferences were assigned either a +.75 (i.e., 'strongly' preferred but somewhat flexible) or -.75 (i.e., 'strongly' undesirable but flexible). The preference levels of 0.01 or -.01 were used to represent *insignificant* preferences (i.e., slight preference for time but 'didn't care').

Since the major objective of the agent-training program was designed to test the effectiveness of the agent in learning to adjust to conflict, it was necessary to create user models that contained preferences that would lead to conflicts among the users. As a result, the user models were initially programmed with the following preferences:

- User 0 loves Wednesday afternoons (+1), likes Tuesday or Thursday afternoons (+0.75), tolerates other times between Tuesday and Thursday (+0.01) more so than Monday or Friday (0), is uncomfortable with weekends (-0.01),dislikes Friday after lunch (-0.75), and hates Saturday, Sunday or Monday mornings (-1).

- User 1 loves Friday after lunch (+1), loves weekend meetings (+0.75), especially before supper (+1), and hates evenings during the week from Monday through Thursday (-1).
- User 2 loves (+1) mornings and Saturday afternoons and hates (-1) meetings during lunch or supper.

In addition to the explicit user model, the agent-learning system was tested with a number of different types of user models in which the preference values were randomly generated. The first random model type was created by uniformly generating different preference levels for each possible situation. The second model type also contained a uniform distribution of positive and negative preferences, but 55% of all preferences were of medium (±0.75) strength, 35% were maximum (±1) strength, and only 10% were weak; this preference pattern models individuals with strong preferences, allowing for more conflict. Other model types were created with a set number of positive, extreme preferences (+1). Each extreme preference set in one user's model caused the opposite preference (-1) to be set in the models for other users if no preference was already assigned; this practice guarantees conflict. An algorithm for averaging neighborhoods of preferences replaced unassigned preferences to create a smooth fading of preference (e.g., a user who loves Monday afternoon meetings will like Monday lunches and Tuesday afternoon meetings unless otherwise programmed).

Once the user preferences were defined, four sets of experiments were run to determine the agent's effectiveness at learning preferences from user actions. The first set of experiments was run without any initial data (that is, no initial knowledge of users' preferences). A second set of experiments was run using the learned preferences from a previous session. A variation on this set of experiments introduced a modified portion of preferences in the user model between runs

in order to test the agent's ability to recover after receiving new data. For some of these experiments, the preferences were automatically altered during the simulation run.

For each experiment, four different measures were used to analyze the quality of the agent's performance: (1) the percentage of correctly classified preferences for each of the three user's across all events over time, (2) the percentage of correctly classified negative or positive preferences across all observed events over time, (3) the number (or percentage) of hard conflicts per hour, and (4) the number (or percentage) of preference-related conflicts per hour. Each measure was recorded for every 'simulated' hour.

The first measure was designed to test the quality of the generalization algorithm (see above) by comparing both experienced and anticipated preferences of the agent to actual user model preferences. The second measure tried to capture the correctness of the agent-learning program and determine its ability to adapt to new information (e.g., how long does it take the system to approach 100% after a change in user preferences?). The third and fourth measures, discussed in the next section, were developed to determine the effectiveness of the agent in reducing conflict. A "hard conflict" was defined as an observed user action that contradicts a previous action initiated by another user, whereas a 'preference conflict' referred to actions that are in conflict with another user's preferences. Preference conflicts were a common precursor to hard conflicts. Because the system had a complete record of the targeted users' preferences, preference conflicts were easy to detect and use in the evaluation of the agent learning system.

The agent-learning program was run for at least 40 'simulated' hours. Positive and negative events were gradually introduced into the environment in a random fashion. As each new event was introduced, the agent-learning program adjusted its preference weights in a manner described in Chapter 3. The learned preferences represented weighted averages of

observed positive and negative events combined with the generalizing algorithm and some probability of being selected. Analysis of several test runs with the preference-learning tool indicated that weights of 0.0001 and 0.25 corresponded well with the actual preferences used to drive the user models (again, described in Chapter 3). User model numbers of +1 usually produced learned model weights of at least 0.25, and user model preference numbers of 0.75 usually produced learned weights above 0.0001. The relatively small values of the latter weights were due to the limited number of events experienced by the 'simulated users' in a particular category. Just as humans require several stimuli before a response is learned, the agent-learning program assigns preference levels that increase as repeated events are encountered.

All the statistics described below assume that a period of stabilization has already occurred. The period of stabilization varied depending upon the difficulty of the learning environment, but was never less than five simulated hours.

Overall, the intelligent agent was able to learn user preferences. In each preference-learning experiment, the agent system learned user preferences as a result of some user action (the second measure) between 82% and 100% of the times, with an overall average of 97%.

Exact matches of medium preferences were made between 66% and 100% of the time, with an overall average rate of 92.5%; fifty percent of all the measures taken during the experiments were above 90%, and 28% reached the 100% accuracy levels. The minimum accuracy level for all but the most random preference models was 80%.

Extreme preferences (that is, preferences with values of +1 or –1) were learned by the agent learning system for all possible events between 49% and 100% of the time, with an overall average of 72%. The extreme range for extreme preference learning is due to the variation in performance among the different preference models rather than a variation in performance

during a single session. Although it took the agent longer to learn extreme preferences, recognition accuracy always stabilized before the end of the 80 simulated hours. Only user preference models that were initialized with random preferences required additional simulated hours to stabilize. (They stabilized after 115, but the user model with the worst performance maintained at least 60% accuracy after 116 simulated hours).

User preference models that were seeded with random preference values required between 32 and 124 simulated hours to reach a point where the agent-learning module performed consistently well. Training sessions that included some patterns in the preferences, reflecting general preferences, stabilized within ten simulated hours. The generalization algorithm is currently designed to anticipate user preferences that do not exist in the more randomly generated set of preference values. The lower performance and longer training times may indicate that the generalization algorithm is not an ideal algorithm for capturing preferences created by artificial constraints (e.g., you hate afternoons but job requirements and work schedule require that you schedule a meeting every Monday afternoon, as opposed to, you love Wednesday afternoon meetings but other afternoons are okay).

As mentioned earlier, experiments were also run with examples of user preferences that changed over time. One experiment involved changing the user's preference table to a series of random numbers in the middle of the experiment. Preferences were first initialized to the three preference models described earlier. After the fortieth 'simulated' hour, one user's preferences were set to uniformly random settings. To encourage conflict among the group participants, strong user preferences were changed to reflect either an opposite preference (40% probability) or an insignificant preference (60% probability). The other users' preferences that did not match the first user's strong preference were either set to an opposite or random number.

As expected, all performance levels dropped significantly following the fortieth 'simulated' hour and continued to be low until the forty-first 'simulated' hour (medium preference, exact match: 100% to 59%; extreme preference, exact match: 75% to 47%; observed events, general match: 93% to 66%). However, the preference-learning tool soon recovered and then stabilized within three simulated hours. An exact match of extreme preferences took around 18 simulated hours to stabilize. All other measures appeared to fluctuate between the 58^{th} and 79^{th} hours.

The overall results for the 'adaptive' version of the agent-learning program were as follows: Matching of general preferences averaged 80% (ranging between 67%-91%); Matching of medium preferences averaged 86% (ranging between 77%-100%); Matching of extreme preferences averaged 74% (ranging between 67%-81%). Matching of extreme preferences actually improved for some agents after the preferences were changed in the middle of the experiment. Although overall performance decreased, the system was able to demonstrate its ability to adapt to behavioral changes.

Figure 4-4 provides a view of how the agent discovers the users' preferences over the course of an experiment. This figure shows how the system learns the initial preference models (previously described) and then adapts to the randomized preferences introduced midway through the experiment. In Figure 4-4, the 'Seen' label refers to the second performance measure; that is, the general preference assignment based on actual observed actions. The Moderate and Extreme labels indicate how well generalization algorithm predicts "user" preferences of medium and extreme. The data for Figure 4-4 was obtained from an experiment in which the user models were initialized with explicit preferences consistent with those of different human users.

As Figure 4-4 indicates, the agent learning program's performance in matching extreme preferences appears to be less effective than matching medium preferences. This is due, in part, to the fact that the generalization algorithm does not assume strong preference levels for times it has never experienced. Therefore, any times that lie outside the agent's experience (no one has scheduled events at that point in time, yet) will be assigned medium preference weights at best.

Random preference models were also tested to ensure that the learning agent's performance was consistent for different situations. Preference models consisting of random but recognizable patterns of behavior performed at least as well as those reported in the previous experiments. However, uniformly random preferences with no relationship among preferences for similar events did not perform as well. The average performance of the learning agents in these cases was above 70% for medium preferences and above 85% for extreme preferences. However, the experiment performance varied substantially more than what occurred in non-random preference experiments. The last ten simulated hours were all above average, so one can conclude that learning is simply slower when preferences are seeded with random values as opposed to more real-world values.

Many of the experiments, including the one represented by Figure 4-4, involved a change of preferences after forty simulated hours. Small changes were captured almost immediately; large numbers of changed preferences took a number of simulated hours to fully explore and unlearn old preferences. In Figure 4-4, the preference learning system was tested with a radical alteration of all preferences. This radical change produces a preference pattern not unlike the uniformly random pattern described earlier. The more random preferences lead to the lower level of accuracy noted in the graph.

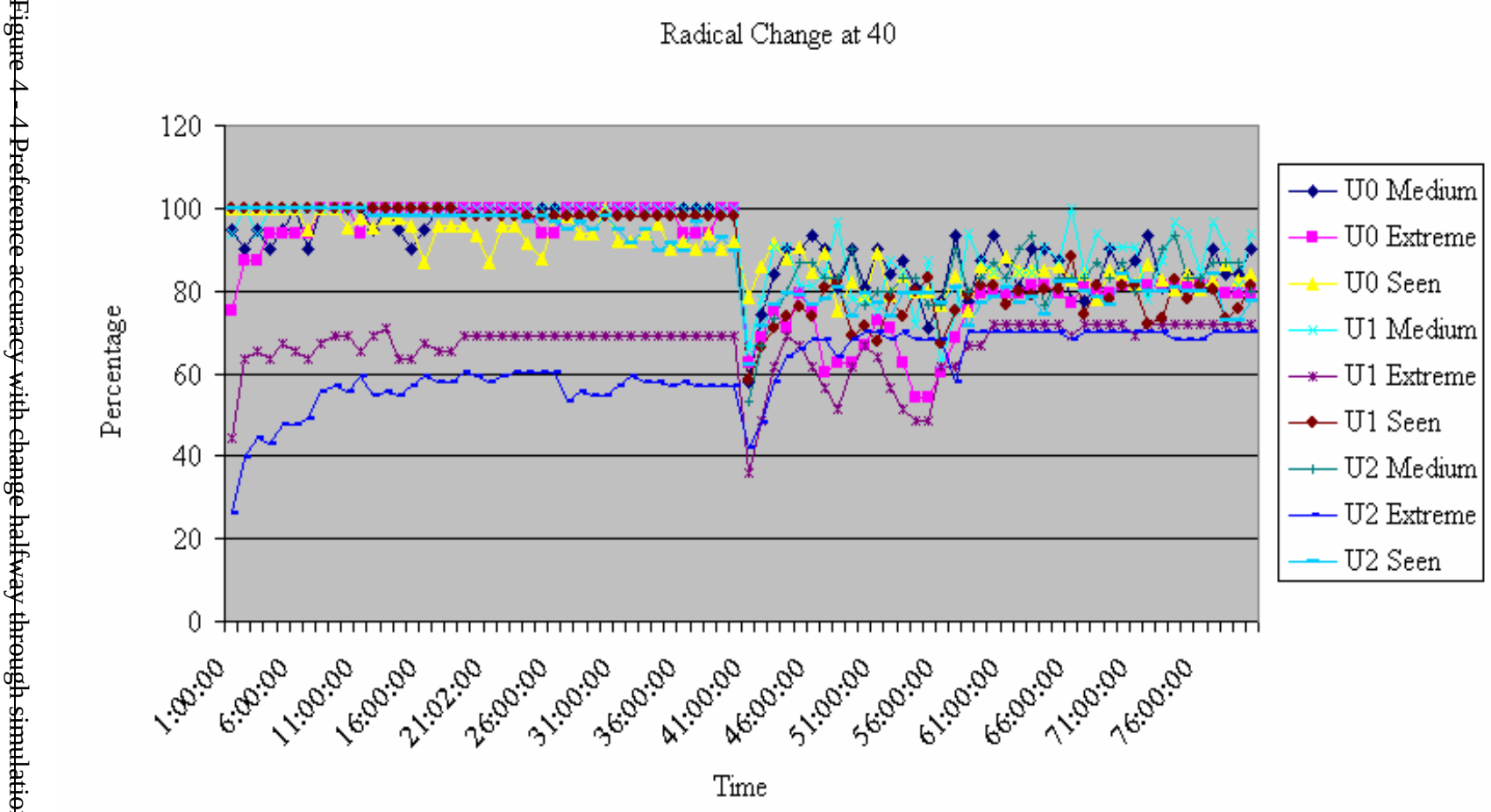


Figure 4 – 4 Preference accuracy with change halfway through simulation

Learning Conflict Avoidance

Learning user preferences is only part of the agent's learning activities. The second part involves learning the user preferences that are associated with responses that anticipate or avoid conflict. The agent's effectiveness in reducing conflict was measured by examining the total number of conflicts that occurred per hour (TC), and the total number of actions that violated users' preferences (TPC). Figure 4-5 compares the conflict measures for two different experiments. The first experiment, labeled NA, shows the number of conflicting events that occurred 'without' agent intervention, whereas the second experiment shows the same information 'with' agent intervention (ANI). Since our 'simulated' users are incapable of adapting to conflicts without some feedback, the results may be misleading. Nevertheless, the data clearly shows that agent intervention can reduce conflict.

As previously mentioned, the Q Learning algorithm learns by adjusting weights, called Q values, assigned to situation-response pairs. The Q values also represent an estimate of long-term reinforcement (both reward and punishment) likely to result from choosing a specific type of response to a situation. Several experiments were conducted, which were designed to test the agent's ability to avoid conflict given different initial conditions. In one experiment, the Q value weights were initialized to zero (TC none), causing all actions to be equal until they received some type of reinforcement (either positively or negatively). In a second experiment, the Q values were adjusted using information from the best performer in the group (TC Best). Finally, a fabricated "ideal" response model was used to initialize the weights (TC Ideal). The ideal response model was developed with weights designed to respond with reasonable, albeit uninspired, actions to various situations. For example, the agent does nothing whenever user preferences are insignificant (with weight 0.1), a user sends a message (0.5), a user views a date

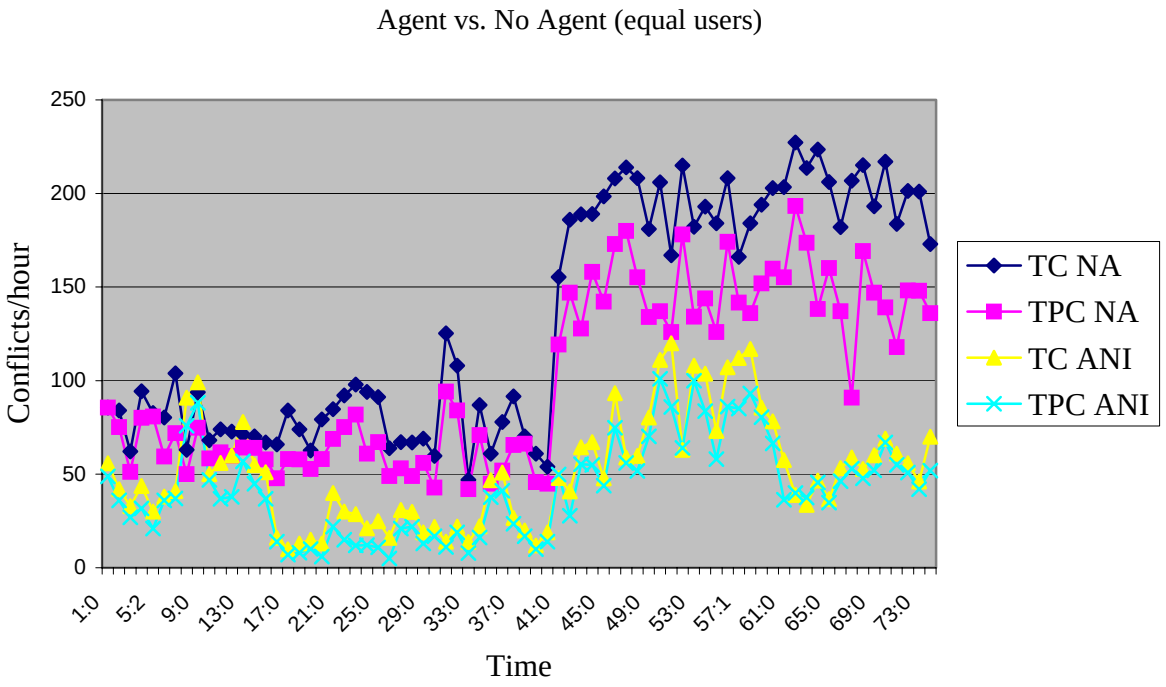


Figure 4 - 5 Agent vs. No Agent

on the calendar (0.01), or users' activity levels are low (0.05). The agent is programmed to issue a warning message if the user tries to schedule an event at a time when others object, or delete an event at a time strongly preferred by another user. Users are also given a warning when other users try to add or delete events at times that are not preferred (0.1). In addition to these notification activities, agents can select a more extreme response where they either automatically anticipate user responses and generate those for the user or block undesirable user actions. The action planner automatically filters inappropriate extreme responses suggested by the Q Learning system and provides negative reinforcement to discourage future selection of such responses. These inappropriate responses are blocking users from sending a message or viewing a date (–0.5), blocking its own user where the user's preferences are stronger than others (–0.1 to –0.5, depending on difference in preference), and taking action against other users where its user does not possess a strong preference (–0.5) or other users have equally strong objections (–0.1).

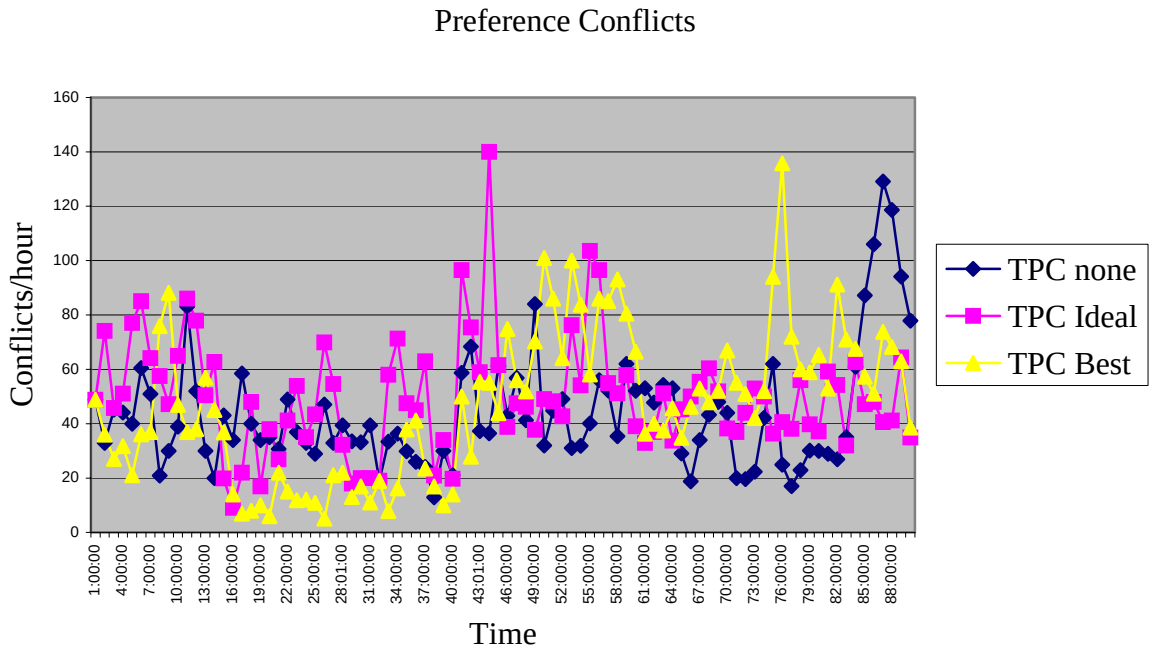


Figure 4 - 6 Comparison of Conflicts with Q values from Different Starting Weights

Figure 4-6 shows how the agent performs when given the three different initial Q value states. Interestingly, the "ideal" condition appears to be less effective at reducing conflicts than the other two conditions. The ideal response model was designed to give 'reasonable' responses to different activities. If these responses are inappropriate, then additional time is required to 'unlearn' and adjust the weights to reflect the correct stimulus-response patterns.

However, all three initialized conditions seemed to perform well and unwell at different times. The agent experiment in which all weights were initialized to zero (TC none) learned how to eliminate 43% of the hard conflicts and 28% of the preference conflicts that occurred among users who had default preferences. When User 0's preferences were changed in the middle of the experiment, conflicts were introduced, but the agent was still able to eliminate 78% of the hard conflicts and 70% of the preference conflicts. The agent experiment in which the weights were seeded with a prefabricated response model (TPC Ideal) eliminated 30% of the hard conflicts and 26% of the preference conflicts before the fortieth hour; 67% and 62%, respectively, after the

fortieth hour. The agent experiment in which the weights were seeded with values from the agent who had the fewest conflicts (TPC Best) eliminated 55% of the hard conflicts (64% after hour 40) and 58% of the preference conflicts (59% after hour 40). The TPC Best experiments seemed to show that the agent learned how to avoid conflicts better than in other experiments.

Experiments were also conducted to determine if prior knowledge of a group's style or group dynamics might affect the agent's ability to reduce conflict. As mentioned previously, groups tend to operate as either a collection of equal individuals or as a single entity, dominated by an active leader. If a single person dominates the group, then the less active members tend to react slower. The learning program allows the researcher to set activity levels for the scheduling of events by different users. Therefore, the single-active group dynamic condition was modeled by setting one user's activity level (user 0) at 90% and the others at 30%(UnEq). The more democratic dynamic condition was modeled by setting activity levels at 80% for all users (EQ). Figures 4-7 and 4-8 illustrate conflict rates for experiments in which the user models were seeded with weights from the 'best' response model. The figures show the impact of mixing the response models of different groups. Unfortunately, the results did not indicate that users encounter fewer conflicts when the system tries to adjust for the different types of group activity levels. The experiments in which all the users were equally active (Figure 4.7) seemed to have fewer conflicts. One interpretation of this result is that the uniformly active individuals have more interactions, which produces more opportunities for the system to learn how to adjust to conflicts.

A more interesting part of the experiment involved examining the actual agent's recommendations for the different behaviors that were simulated in the different experiments. Some of the responses seemed reasonable while others did not. A partial listing of these

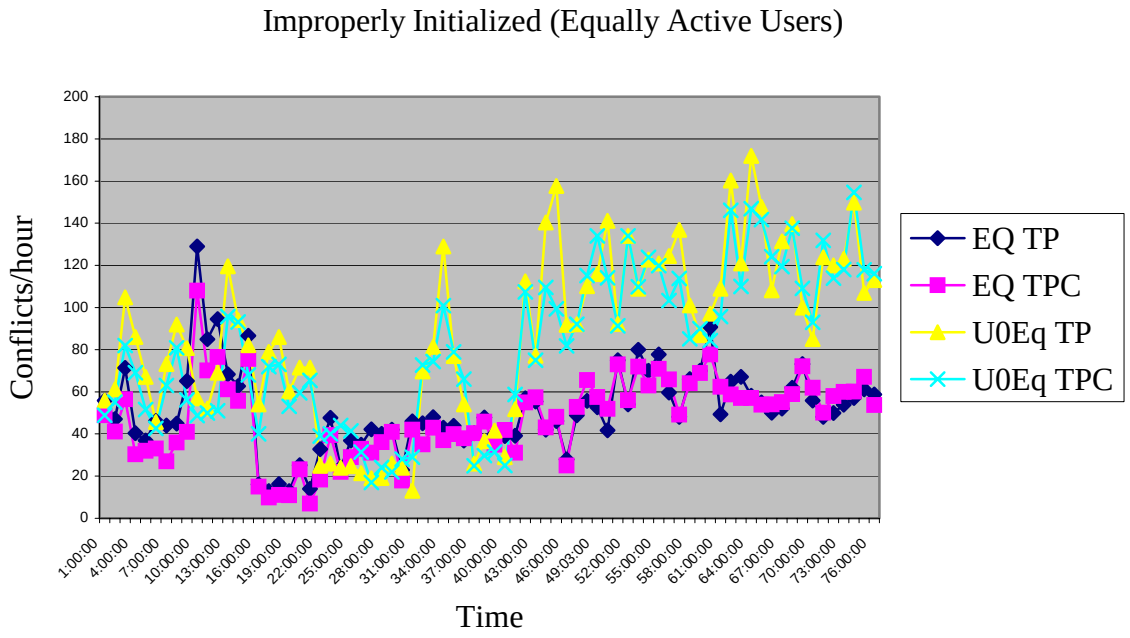


Figure 4 - 7 Comparison of Response Models for Equally Active Users

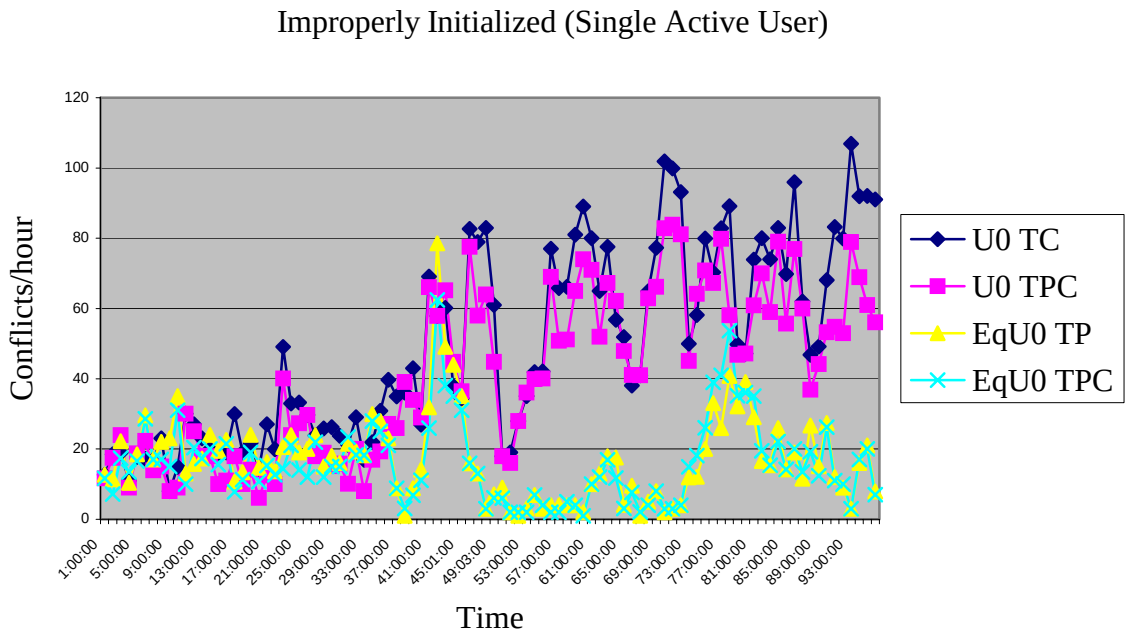


Figure 4 - 8 Comparison of Response Models for Hierarchical Group

responses from a sample experiment can be found in Appendix D. The most common oddity in agent responses takes the form of an inappropriately extreme response. For example, in 3.1 of Appendix D, the Q learning system advocates preventing its user from viewing or changing

events for a particular date. The agent is pre-programmed to consider this behavior to be unreasonable, so the response is converted into a warning. If the response was simply punished and ignored, the Q learning system would eventually learn not to select that response. By converting the response into an informative warning about potential conflicts about a particular date, the agent reduced the possibility of future conflict and, thus, future punishment.

An agent's list of possible responses included some examples that did not make sense for every situation. The current implementation translates these inappropriate responses into more appropriate responses, which allows the agent to take some suitable action. This research considered the alternative of negatively reinforcing inappropriate actions, which would have had the effect of making the system gradually learn the appropriate action. However, this was rejected for the following reason. Q learning spreads its negative reinforcement to those choices leading up to the point of inappropriate response or conflict. While such a strategy might work for a conflict situation, it may not be correct to punish prior choices for a poorly learned and probably unrelated response.

The agents for the individual users produced reasonable responses most of the time, once some of the responses suggested by the Q learning system were altered to reflect more generic messages. For example, if a user selects a date for viewing or adding a new entry, the user's agent may display a message warning him/her of potential problems. Thus, the user is encouraged to think about avoiding conflict before scheduling any new event. In Appendix D, after message 20, response 20.1 warns its user that removing an event that was scheduled by another user at a preferred time is inappropriate. In response 20.3, the agent for the user who scheduled the event is warned that the event is being deleted. The likelihood of a response from

the original scheduler of the event should increase users' sensitivity to each other's desires over time.

Overall, the agent seemed to perform well, developing a reasonably accurate model of different user’s preferences. It was also able to reduce conflict-related behavior. The agent-learning program tended to need more training time in those experiments where it had no prior knowledge of user preferences. Furthermore, the agent appeared to be able to recover from sudden changes in the middle of an experiment and adapt to the new data.

CHAPTER 5

CONCLUSIONS

The major goal of this research was to examine whether a software agent could 'learn' a group's preferences well enough to assist members in avoiding conflict. The agent was defined as a software component embedded within a collaborative interface. Although previous research found that advisory style agents were able to learn individual preferences (Mitchell et al., 1994), little work has been done in developing specialized agents that are capable of recognizing conflicting preferences that arise within group work. As a result, this researcher developed a preference-learning program that was designed to learn user preferences specifically for the purpose of overcoming group conflict. The agent learning tools were designed around a group calendar application that allowed users to schedule meetings at preferred times. The system's ability to learn a conflict response policy and preference model was implemented using a Q Learning technique and a related reinforcement-based learning algorithm. Q Learning was selected because it does not require a prior model of either learning or action selection. After developing the software, students were asked to use the system to schedule meetings for different types of events.

The initial plan was to embed the learning system within the calendar application and have the agents detect and resolve conflicts in real-time. However, it soon became apparent that this strategy was not going to work. Since the initial volunteer groups used the software without stating their preferences, the agent system was forced to learn both user preferences as well as the responses to conflicts that arose because of the differences among users' preferences. Since

the response module used the users' preferences as input, it had to unlearn any inappropriate responses before it learned new responses. In all cases, the learning system had an insufficient number of events and time to build the appropriate response model in real time.

To accelerate the learning process, a program was created to expose the agents to group-like events in a training environment. The goal was not to duplicate every aspect of human interaction but to provide the agent system with a sufficient number of training examples that would allow it to learn a "standard" set of responses. Because the trainer uses an explicit preference model to generate simulated user responses, it can compare this model to the agent's learned preference model to verify system performance. The trainer generates user activities faster than human users, allowing the trainer to model the use of the application over extended periods. The trainer was used to run a series of experiments and collect conflict rates over a period of simulated time (typically, from 80-160 hours, or 2-4 weeks of continuous use).

The trainer was also used to validate the agent's preference model. In later versions of the human subject sessions, users were asked to list their stated preferences. In these cases, the agent's preference model did not always match the user's stated preference; however, the user's stated preferences did not always serve as an adequate predictor of user behavior. The discovery that explicitly stated preferences were often erroneous or changed during actual practice suggested that preferences learned from observation were more useful than explicitly stated preferences.

The agent training system was used in this research to conduct a series of off-line experiments to determine the effectiveness of the agent system in learning user preferences and developing a response policy for avoiding conflicts. Three different user models were created as the 'target' group, each reflecting moderate degrees of conflict. Later, other models were created

with randomly generated preference values representing different levels of conflict. As the system produced different messages, similar to what occurred in the original calendar application, the three user agents processed the messages and gradually adapted their user preference model to whatever was observed. The agents also adapted their response policies so that they could 'better' reply to the various conflicts that were encountered during the experiment. For each experimental run, the system recorded the percentage of events correctly classified as compared to the users' true preferences, and the number of conflicts detected. Conflict rates usually stabilized within forty hours; that is, whenever there were no changes in the users' preferences.

Findings

As previously mentioned, the data from the human subjects' experiments indicated that users often ignored their initial scheduling preferences about 35% of the time. Moreover, there was a great deal of variation among different users concerning how often they deviated from their preferences. Some did it often (56% of the time), while others rarely took actions that were different from their stated preferences. Because of the results of these data, offline experiments were run to determine the effectiveness of the learning component of the system.

In general, the offline experiments showed that the agents were able to learn user preferences, averaging 97% accuracy rates. The agents were less successful at classifying the strength of preferences, averaging only an 82% accuracy rate. However, trying to correctly classify the different strengths of a preference is much more difficult than classifying something as being either positive or negative because the agent needs to experience all combinations of the different categories of 'strengths' (i.e., strong, moderate, and weak). The agents also tended to have difficulty recognizing random preferences, averaging only 72% classification accuracy.

The agent-learning system was able to eliminate a significant amount of conflicts compared to user models whose agents were disabled. The average conflict rates that occurred while experienced agents were active were 59% less than when they were disabled.

The conflict rates for a well-trained response model were approximately 41% lower than response models that learned from no previous model. Improvements were noticed even in cases where an agent was initialized with different user response models. One experiment indicated that an agent trained with data that was meant to simulate a democratic group of active individuals was better at eliminating conflict, even when configured with a different group of user styles. This latter experiment was intended to test the hypothesis that different response models should be selected based on group type. The hypothesis assumed that a democratic group of users who were characterized as being equally engaged would require a different set of responses than a hierarchical group who was dominated by a single user. The hypothesis was rejected when agents that were originally trained with “democratic” group data performed even better when used to advise a hierarchical group; in one case, agents trained for “democratic” groups had 42% of the conflicts encountered by agents actually trained for the hierarchical group. This seems to indicate that new agents should use response models from experienced agents involved with active groups to reduce training time.

Conclusion

In previous research, it was found that groups often encounter conflict, particularly in a computer supported collaborative environment. More conflicts occur within this environment because there is an absence of familiar cues that can cause users to become confused about the expectations of others (Johnson-Lenz et al., 1991). These problems prompted this researcher to develop a calendar application system that could detect or predict conflict and then use this

information to help groups overcome conflict. The system does this by observing how each user schedules events and then detects whether attempts at scheduling new events conflict with a user's preference model. The agent-learning system that was built as a component within the calendar application appears to accomplish its goal. The system is able to learn user preferences and is able to respond to conflicts or potential conflicts in a way that reduces overall conflict.

The agent was effective at reducing conflicts compared to when no agent was present. The agent was also effective in learning users' preferences, particularly when the users' preferences did not contradict anticipated patterns. Even when this occurred, the generalization algorithm performed poorly only when it anticipated preferences outside the agent's experience.

Although this research failed to show that different types of groups require different response models, the research did indicate that an agent experienced at adapting to conflict can benefit users whose preferences and working style are different from the original user. This phenomenon suggests that learning can be greatly enhanced by providing new users with an experienced user's agent. The lack of a consistent pattern in the response models of different agents suggests that either a variety of responses may be helpful or appropriate responses vary from user to user.

In any case, the evidence indicated that Q learning was effective at creating response models that reduced conflict. Although somewhat slow, the learning system was able to learn user preferences and response policies that reduced conflict.

There were also several other conclusions obtained that were peripherally related to the above findings. For example, many of the volunteers who assisted in the data collection phase of the project seemed to encounter difficulty working with each other exclusively by computer. The volunteers often found it problematic to maintain aware of the actions of others; even when their

preferences were shared among the group. Presumably, some of this behavior arose from the students being unaccustomed to using a computer for group work, as opposed to individual work. Someone inexperienced with collaborative software can certainly benefit from tools that track the activities of other members. One of the sub-goals of this work was to develop a system that could 'learn' to provide this type of assistance and, at the same time, remain unobtrusive.

The agent- learning module tracks user preferences sufficiently well to aid the user in maintaining awareness of the other group members and in avoiding unnecessary conflicts due to inattention or forgetfulness. The question of whether it provides the appropriate level of unobtrusiveness, enough to maintain awareness without distracting the user, is a question that remains unanswered. Since the software was designed originally to be part of a calendar application tool, further studies with actual human users should be performed. However, the data obtained from this study clearly shows that the agent software was successful in reducing conflicts.

Some means of tracking user preferences is absolutely essential for any intelligent system that is intended to help users avoid conflict. Furthermore, a system that can learn user preferences dynamically is superior to a system that requires users to explicitly enter their preferences. Ideally, users can avoid listing preferences by having a system that can automatically adjust to changes in users' preferences as they are working with their group. Unfortunately, the user studies performed during phase one of this study indicated that group members often contradicted their own stated preferences. However, the experiments in this study also indicate that a preference learning system can learn preferences for users even when that behavior is inconsistent, but it may require additional time.

Future Research

Preference Learning

A number of different types of learning methods were considered for the preference learning system before selecting a system based on weighted evidence averaging combined with an explicit generalization algorithm. These other methods were ultimately rejected because they had problems recognizing preferences that changed over time. Using a static associative technique such as a traditional back-propagation neural network to learn user preferences between sessions might be effective, however, if an appropriate method for “forgetting” and generalizing could be used to prevent the training set from growing too large. A Bayesian network might also be used to learn appropriate weights for preferences, with each weight corresponding to the strength of the user's preferences. The strength of a preference would represent the network’s estimate of the probability that the user possesses a particular preference.

A probabilistic neural network might approach these same values if it is trained with a representative set of examples. However, probabilistic neural networks have some of the same problems that other classification systems have in that they assume that everything is classifiable, and the goal is to learn that classification. Human preferences require a more dynamic mapping of objects to items; so instead of simply learning a mapping, the classification system must also track changes to the mappings.

One technique for handling a dynamic system is to add a new exemplar to the training set whenever a misclassified event occurs. Whenever an exemplar is misclassified, the exemplar’s class can be changed. However, one misclassified event can have major effects on the system. As a result, the program needs to be able to be trained to recognize and forget obsolete exemplars after it has seen repeated evidence that an action or event is obsolete.

Many popular machine-learning techniques use a long training period to create a set of static associations. Further research is needed to incorporate these methods into a dynamic learning environment.

Improvements to the Response Model

Presently, the response model for the agent-based learning system developed for this project uses a technique that is similar to that used to train automated control systems. Control systems learn from experience and can, over time, unlearn old responses as new responses become more appropriate. The need to adapt to new stimuli made Q Learning and related reinforcement-based systems a very attractive choice for tracking dynamic human attitudes and needs. However, this type of reinforcement learning represents a type of control system that has a definite and recognizable impact on the environment. Some of the actions taken by the agent described in this work, such as warning the user of another user's preferences, require more subtle responses. Thus, one of the problems that need to be examined is how to make the system more responsive to subtle changes or actions.

The system currently learns from experience. At the very least, the system is effective at discouraging agents from doing nothing while conflicts exit. The system is very good at training itself not to take extreme actions in areas of questionable user preference. Some fine-tuning of response selection does take place. For example, the user can explicitly discourage the system from issuing intrusive warnings too often. However, additional work needs to be done on personalizing the response selection process and encouraging subtle hints or stronger warnings at the appropriate times.

Also, learning a response model that is effective in reducing conflict appears to take some time. The performance data indicated that it took between 10 and 20 simulated hours for the

system's conflict rates to stabilize at relatively low levels. Some combination of off-line and real-time training may be necessary to get more reasonable response rates.

The Q learning algorithm is designed to learn a weighted average estimate of future reinforcement by being exposed to a variety of experiences. If users experience few conflicts, then the system may never learn an adequate number of responses to conflict, since conflict is not part of its immediate experience. Of course, since the agent's job is to reduce conflict, then this is an acceptable problem as long as the group remains relatively conflict-free. However, as more data is collected from real-live groups, it may be more desirable to develop user models for the learning system that reflect both low- and high-conflict-level groups and use these models to initialize the agent. Later, user models based on the preferences and activity levels of actual members of the group can be used to train the system off-line.

The response model might also be improved by using some type of genetic algorithm. Genetic algorithms do not necessarily require training examples. A set of response weights could be created, and a measure of conflict and "user satisfaction" could be used as the evaluation function. If parallel processors were added, the genetic algorithm could experiment with numerous response models simultaneously, keeping only the ones that worked best. When a new active session begins, the best response model would be used and validated with the real users.

Other Areas of Activity

Another possible future activity is to require students to use the scheduler to coordinate meetings for a group project. Currently, the system only functions with a group of simultaneously active users. The system could be modified so that changes can be made to a schedule, even when the user was off-line. In addition to making it more useful, this modification

would make computer assistance even more important, since users would need to know about the events that occurred while they were away from the system.

Finally, the system could be extended to include additional collaborative tools such as document sharing to enhance the ability of users to work together on-line. A more powerful system may be the only way to justify requiring extended use of the software so that it can have a chance to adapt to the needs of the users in the group.

Final Comments

The idea for this study began with a question about mutual exclusion in distributed software for support of groups. The work evolved into automated support for user preferences and using this information to reduce conflicts. Along the way, the research uncovered an interesting problem for any application that supports cooperative group work: maintaining the impression that a user is part of a group that is working together. This goal can be achieved by helping the user remain aware of others while, at the same time, facilitating the user's own work. Although it can be argued that technology will eventually make this concern obsolete because users will be able to interact with each other through virtual environments that simulate face-to-face meetings, users will continue to benefit from intelligent advisors that augment their understanding of the group's work. The foundation laid by this work has the potential to evolve into a new type of intelligent assistant that adapts to best facilitate the work of the group as a whole rather than that of a single individual.

APPENDIX A

SCENARIO USED FOR JULY, 2000, GROUP TESTING AND DATA COLLECTION FOR USER PREFERENCE MODELING

You are planning a conference of computer science lectures (perhaps on computer gaming or something similar which may attract a relatively large number of students) with the other members of your group. The conference will be held during the last full week of the month at the time which best suits the members of the group and their opinion of when the most students will be able to attend. Naturally, you consider your own attendance a priority.

Assume that scheduling can be done on-line (what we're doing now), but most other activities require meeting together at times which fit the majority, if not all, of the personal schedules of the members of the group. Consider the following a set of minimum requirements:

Description	Min. number of meetings	Order
Major topics of conference	2	at start
Securing presenters	2	near start, early
Advertising	2	early, after schedule
Acceptance and scheduling	2	as soon as presenters known
Conference administration	2	prefer at least week before event
Last minute stuff	1	day before event
Conference event, itself	4	day of event

The "number of meetings" corresponds to roughly a measure in hours. Thus, 2 meetings could be 4 half-hour meetings, 2 one-hour meetings, or 1 two-hour meeting. Of course, in some cases, time will be needed for members to think about matters before continuing the discussion. For example, it might be better to make sure "Acceptance and scheduling" consists of at least two separate meetings: discussion of whether a presentation is appropriate, reconvene to make sure everyone is in agreement still and to discuss how to schedule resources (rooms, computers, times, etc.), then let everyone think about it and return for a short meeting to vote it official. Or, you could just do it all at once without letting people put much thought into it at all. Note that this is a minimum set of requirements. If you think of other things or want more time, feel free to allocate more meeting times.

The ultimate purpose of this exercise is to gather information about your preferences, so use the calendar itself to express your preferences by adding events at times which are convenient for you. Feel free to change the events created by others if you cannot attend or prefer to hold the meeting at another time. Some coordination and discussion can be held using the message facility. Please do not speak to each other even if you wind up in the same room as I need a record of these discussions. The content of what you type will not be used in my research, but if user preferences are not obvious from your actions with the calendar, I may have to use the messages for additional clues. In other words, "Dr. Burris is evil incarnate" will not be used, but a message to the effect "I'd rather not meet on Monday evenings" may.

Major topics of conference: Meeting to prepare a list of general subject areas related to the point of the conference. This list will be given to possible presenters as a guide of acceptable presentation material.

Securing presenters: Discussion of ways to advertise for presentations or who to contact directly as an invited presenter. Delegate tasks, then meet again to verify that everything is going smoothly and a response is being generated.

Advertising: Determine who is invited and the best ways of informing these people of the conference. Delegate tasks such as contacting radio or newspaper or posting flyers with initial data about conference. Meet again after schedule of presentations has been fixed to discuss creation and dissemination of more detailed information about conference; delegate appropriate tasks to accomplish this dissemination.

Acceptance and scheduling: Discuss acceptability of presentations and any changes which need to be made. Meet again when final list of presentations are finalized to plan resource allocation: which rooms, do they need to be computer labs, and what times. Additional meetings/hours may be required to leave time to verify availability of presenters and resources.

Conference administration: Discuss and delegate the tasks necessary to make the conference operate smoothly. Make sure someone provides refreshments, determine who is going to work the registration desk, what does "registration" entail, etc.

Last minute stuff: signs, registration materials, guest packets, etc. May want to allocate several hours for group to work together to bringing all the pieces together in preparation for the conference on the following day.

APPENDIX B

EXPLICIT USER PREFERENCES ELICITED FROM VOLUNTEERS

1 = strong dislike for meetings, 2 = prefer not to meet, 3 = don't care,
4 = prefer to meet, 5 = strong preference for meetings

The left column represents preferences regarding short meetings (one hour or less) and the right column represents preferences regarding longer meetings.

Group 1			Group 2		
User 1:			User 1:		
Early in month:	3	3	Early in month:	4	3
Late in month:	3	3	Late in month:	2	2
Early in week:	4	4	Early in week:	3	1
Late in week:	2	2	Late in week:	3	2
Morning:	3	3	Morning:	5	3
Lunch: 2	2		Lunch: 1	3	
Afternoon:	3	3	Afternoon:	2	4
Supper: 2	2		Supper: 4	5	
Evening: 1	1		Evening: 5	1	
User 2:			User 2:		
Early in month:	3	4	Early in month:	4	3
Late in month:	4	2	Late in month:	2	3
Early in week:	2	2	Early in week:	4	4
Late in week:	5	4	Late in week:	2	2
Morning:	3	3	Morning:	2	1
Lunch: 2	2		Lunch: 3	3	
Afternoon:	4	4	Afternoon:	2	1
Supper: 2	2		Supper: 4	3	
Evening: 1	1		Evening: 3	4	
User 3:			User 3:		
Early in month:	4	2	Early in month:	2	5
Late in month:	2	4	Late in month:	4	2
Early in week:	3	3	Early in week:	1	3
Late in week:	3	3	Late in week:	5	5
Morning:	1	1	Morning:	1	2
Lunch: 1	1		Lunch: 3	2	
Afternoon:	3	3	Afternoon:	2	5
Supper: 3	3		Supper: 1	1	
Evening: 3	3		Evening: 1	1	

APPENDIX C

HIERARCHICAL VS. UNIFORM INDICATORS IN HUMAN DATA

The following table is a reproduction of a spreadsheet used to analyze and compare the relative activity of users in different groups. This data was collected by logging all user activity during a 1-2 hour group session.

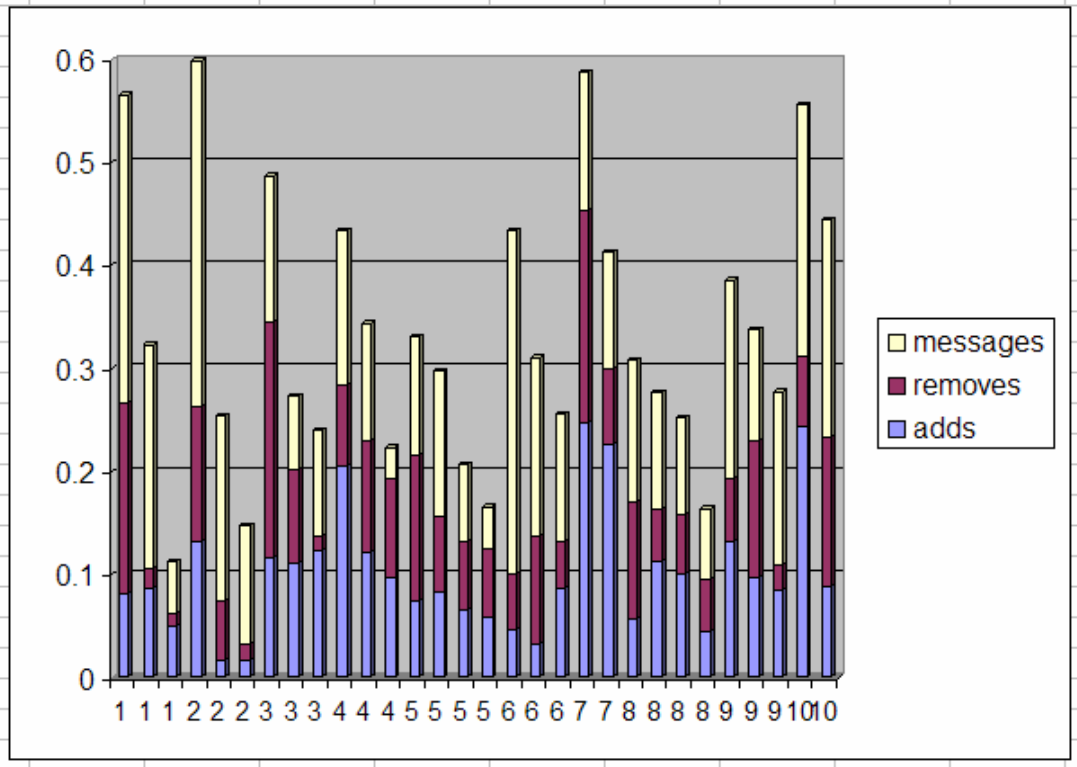
The columns on the left contain action counts. User actions are categorized as ADD, user schedules new event, MOD, user changes or removes an event, and MSG, user sends a message to another user. The columns in the middle represent relative participation by each user. Each percentage represents the responsibility of each user for the actions of a particular type during that group session. Because actions that affect the shared schedule were considered separately from message passing activity, a column marked “Events” summarizes both actions that add to the schedule and that modify scheduled events. Users within a group are listed from the most active user to the least active user. The right two columns indicate the differences in activity levels.

The final page of the table contains columns of fractions that are related to the percentages described above with one significant difference. The fractions indicate the ratio of one user’s performance of a type of action compared to the total actions taken by all users in the group. This data was used to create fig. 4-3.

COUNTS FOR ALL RECORDED SESSIONS													User/next	First/user
group3.txt														
user	ADD	MOD	TotEvts	MSG	Total		%ADD	%MOD	%Events	%Msgs	%active		X more active	
2	18	35	53	22	75		0.333333	0.686275	0.504762	0.44898	0.487013		1.785714	
1	17	14	31	11	42		0.314815	0.27451	0.295238	0.22449	0.272727		1.135135	1.785714
0	19	2	21	16	37		0.351852	0.039216	0.2	0.326531	0.24026			2.027027
	54	51	105	49	154									
group6.txt														
user	ADD	MOD	TotEvts	MSG	Total		%ADD	%MOD	%Events	%Msgs	%active		X more active	
0	10	12	22	73	95		0.277778	0.266667	0.271605	0.528986	0.43379		1.397059	
1	7	23	30	38	68		0.194444	0.511111	0.37037	0.275362	0.310502		1.214286	1.397059
2	19	10	29	27	56		0.527778	0.222222	0.358025	0.195652	0.255708			1.696429
	36	45	81	138	219									
group1.txt														
user	ADD	MOD	TotEvts	MSG	Total		%ADD	%MOD	%Events	%Msgs	%active		X more active	
2	13	30	43	48	91		0.371429	0.857143	0.614286	0.527473	0.565217		1.75	
1	14	3	17	35	52		0.4	0.085714	0.242857	0.384615	0.322981		2.888889	1.75
0	8	2	10	8	18		0.228571	0.057143	0.142857	0.087912	0.111801			5.055556
	35	35	70	91	161									
group8.txt														
user	ADD	MOD	TotEvts	MSG	Total		%ADD	%MOD	%Events	%Msgs	%active		X more active	
3	9	18	27	22	49		0.18	0.418605	0.290323	0.333333	0.308176		1.113636	
1	18	8	26	18	44		0.36	0.186047	0.27957	0.272727	0.27673		1.1	1.113636
0	16	9	25	15	40		0.32	0.209302	0.268817	0.227273	0.251572		1.538462	1.225
2	7	8	15	11	26		0.14	0.186047	0.16129	0.166667	0.163522			1.884615
	50	43	93	66	159									
group7.txt														
user	ADD	MOD	TotEvts	MSG	Total		%ADD	%MOD	%Events	%Msgs	%active		X more active	
2	24	20	44	13	57		0.521739	0.740741	0.60274	0.541667	0.587629		1.425	
0	22	7	29	11	40		0.478261	0.259259	0.39726	0.458333	0.412371			1.425
1	0	0	0	0	0		0	0	0	0	0			
	46	27	73	24	97									
group9.txt														
user	ADD	MOD	TotEvts	MSG	Total		%ADD	%MOD	%Events	%Msgs	%active		X more active	
1	11	5	16	16	32		0.423077	0.277778	0.363636	0.410256	0.385542		1.142857	
0	8	11	19	9	28		0.307692	0.611111	0.431818	0.230769	0.337349		1.217391	1.142857
2	7	2	9	14	23		0.269231	0.111111	0.204545	0.358974	0.277108			1.391304
	26	18	44	39	83									

COUNTS FOR ALL RECORDED SESSIONS													User/next	First/user
group11.txt														
user	ADD	MOD	TotEvts	MSG	Total		%ADD	%MOD	%Events	%Msgs	%active		X more active	
0	14	6	20	4	24		0.666667	0.666667	0.666667	0.8	0.685714		2.181818	
1	7	3	10	1	11		0.333333	0.333333	0.333333	0.2	0.314286			2.181818
	21	9	30	5	35									
group5.txt														
user	ADD	MOD	TotEvts	MSG	Total		%ADD	%MOD	%Events	%Msgs	%active		X more active	
3	9	17	26	14	40		0.264706	0.404762	0.342105	0.311111	0.330579		1.111111	
0	10	9	19	17	36		0.294118	0.214286	0.25	0.377778	0.297521		1.44	1.111111
1	8	8	16	9	25		0.235294	0.190476	0.210526	0.2	0.206612		1.25	1.6
2	7	8	15	5	20		0.205882	0.190476	0.197368	0.111111	0.165289			2
	34	42	76	45	121									
group10.txt														
user	ADD	MOD	TotEvts	MSG	Total		%ADD	%MOD	%Events	%Msgs	%active		X more active	
0	22	6	28	22	50		0.733333	0.315789	0.571429	0.536585	0.555556		1.25	
1	8	13	21	19	40		0.266667	0.684211	0.428571	0.463415	0.444444			1.25
	30	19	49	41	90									
group2.txt														
user	ADD	MOD	TotEvts	MSG	Total		%ADD	%MOD	%Events	%Msgs	%active		X more active	
2	16	16	32	41	73		0.8	0.64	0.711111	0.532468	0.598361		2.354839	
0	2	7	9	22	31		0.1	0.28	0.2	0.285714	0.254098		1.722222	2.354839
1	2	2	4	14	18		0.1	0.08	0.088889	0.181818	0.147541			4.055556
	20	25	45	77	122									
group4.txt														
user	ADD	MOD	TotEvts	MSG	Total		%ADD	%MOD	%Events	%Msgs	%active		X more active	
0	34	13	47	25	72		0.485714	0.276596	0.401709	0.510204	0.433735		1.263158	
2	20	18	38	19	57		0.285714	0.382979	0.324786	0.387755	0.343373		1.540541	1.263158
1	16	16	32	5	37		0.228571	0.340426	0.273504	0.102041	0.222892			1.945946
	70	47	117	49	166									

	adds	removes	messages
1	0.080745	0.186335	0.298137
1	0.086957	0.018634	0.217391
1	0.049689	0.012422	0.049689
2	0.131148	0.131148	0.336066
2	0.016393	0.057377	0.180328
2	0.016393	0.016393	0.114754
3	0.116883	0.227273	0.142857
3	0.11039	0.090909	0.071429
3	0.123377	0.012987	0.103896
4	0.204819	0.078313	0.150602
4	0.120482	0.108434	0.114458
4	0.096386	0.096386	0.03012
5	0.07438	0.140496	0.115702
5	0.082645	0.07438	0.140496
5	0.066116	0.066116	0.07438
5	0.057851	0.066116	0.041322
6	0.045662	0.054795	0.333333
6	0.031963	0.105023	0.173516
6	0.086758	0.045662	0.123288
7	0.247423	0.206186	0.134021
7	0.226804	0.072165	0.113402
8	0.056604	0.113208	0.138365
8	0.113208	0.050314	0.113208
8	0.100629	0.056604	0.09434
8	0.044025	0.050314	0.069182
9	0.13253	0.060241	0.192771
9	0.096386	0.13253	0.108434
9	0.084337	0.024096	0.168675
10	0.244444	0.066667	0.244444
10	0.088889	0.144444	0.211111



APPENDIX D

SAMPLE LOG AND AGENT RESPONSES

Following is an annotated log of the user actions and agent reactions during a typical session with the agent-training system. Each log entry consists of at least a timestamp and a message type. The timestamp's last three digits describe the user ID of the user whose actions created the message. In the calendar application, SEND MSG refers to an attempt by one user to explicitly communicate with another. In the agent-training tool, SEND MSG is used to introduce comments into the log. The comments listed here were used to track agent responses to different user actions. Agent's each have an ID identical to the user it serves, so Agent 2 serves User 2, and so on. On the first line describing an agent's response, the numbers in parentheses represent the activity level and preference level of an agent's user, followed by these same measurements for the acting user; if an agent's user is the cause of a message, then the second pair of values typically represents the data of the user's most extreme opponent (minimum preference if adding, maximum preference if deleting, etc.). The number after the colon is the code for the Q-learning system's advocated response (0=do nothing, 1=subtly indicate problem, 2=warn user of problem, 3=immediately reject action as insupportable, 4=automatically generate a response to a user's action). The number in parentheses indicates the reinforcement value associated with this response. Responses with a value of –1 are automatically rejected and converted into a milder response.

The user models of the agent-training system assume that a message from the agent regarding the behavior of other users will affect a user's ability and desire to respond to such behavior. This information is encoded as a probability of response and an average time to respond; a change in responsiveness is noted below as (0.90, 28) -> (1.0, 13), indicating that the user will now respond to 100% of undesirable actions in 13 seconds instead of 90% in 28 seconds. Responsiveness fades over time until it reaches the "normal" responsiveness of the

user. (An agent warning a user against its own action creates an entry that remembers the warning, for a time; a list of such entries are used to reduce the likelihood of a user model scheduling events at that time in the future. No changes are recorded in the log, although a value representing "selfishness" is reduced by repeated warnings; this value helps to determine the likelihood that a user model will pay attention to its list of warnings.)

```
193874002 SEND MSG Agent 2 (0.401024, -0.135874, 0.104266, 0.121618): 2 (-1)
193874002 SEND MSG Agent 0 (-0.0158003, 0.401024, 0.101291, 0.104266): 4 (-0.1)
193874002 SEND MSG Agent 0 WARN of other user 2 affects responsiveness
                  (0.904164,28.1852)->(1,12.9034)
193874002 SEND MSG Agent 1 (0.264214, 0.401024, 0.121618, 0.104266): 3 (-0.273621)
193874002 SEND MSG Agent 1 WARN of other user 2 affects responsiveness
                  (0.899962,256.36)->(1,105.519)
193874002 START VIEW 11/10/100
```

- Agent 2 advocates warning user, but user complains of too many warnings.
- Agents 0 and 1 advocate inappropriate responses that are translated into warnings.

```
193892000 SEND MSG Agent 0 (0.501072, -0.262394, 0.0967818, 0.121618): 4 (-0.1)
193892000 SEND MSG Agent 0 WARN against adding during 0 time period on 11/5 (95)
193892000 SEND MSG Agent 0 WARN against adding during 1 time period on 11/5 (93)
193892000 SEND MSG Agent 1 (-0.181191, 0.501072, 0.121618, 0.0967818): 4 (-0.1)
193892000 SEND MSG Agent 1 WARN of other user 0 affects responsiveness
                  (0.958751,162.628)->(1,88.5476)
193892000 SEND MSG Agent 2 (0.390963, 0.501072, 0.104266, 0.0967818): 0 (0.1)
193892000 START VIEW 11/5/100
```

- Agent 0 warns user against adding before afternoon.
- Agent 1 warns user that 0 may be about to add undesirable events.

```
193902002 SEND MSG Agent 2 (0.207759, -0.304294, 0.100838, 0.121618): 3 (-1)
193902002 SEND MSG Agent 0 (-0.304294, 0.207759, 0.0967818, 0.100838): 4 (1)
193902002 SEND MSG Agent 0 AUTO DELETE
193902002 SEND MSG Agent 1 (0.598275, 0.207759, 0.121618, 0.100838): 0 (0.1)
193902002 ADD ENTRY 11/10/100 11:00 - 11:30
```

- Agent 2 advocates restricting own user; program rejects restricting own actions without strong cause.
- Agent 0 automatically deletes (user rewards).

```
193902000 SEND MSG Agent 0 (-0.304294, 0.598275, 0.0969427, 0.121618): 2 (-1)
193902000 SEND MSG Agent 1 (0.598275, -0.304294, 0.121618, 0.0969427): 1 (-0.1)
193902000 SEND MSG Agent 1 NOTIFY of other user 0 affects responsiveness
                  (0.925987,197.282)->(0.928919,50.2549)
193902000 SEND MSG Agent 2 (0.576542, -0.304294, 0.100838, 0.0969427): 1 (-0.1)
193902000 SEND MSG Agent 2 NOTIFY of other user 0 affects responsiveness
                  (0.75257,26.574)->(0.755152,20.4106)
193902000 MOD ENTRY 11/10/100 0 11:00
```

- Agent 0 advocates warning user, but this is automatically rejected by software.
- Agents 1 and 2 notify their users of agent 0's act.
- Mild punishment intended to impact all recent policy since conflict has occurred.

```
193910002 END VIEW 11/10/100
193915002 SEND MSG Agent 2 (0.450122, -0.212943, 0.106944, 0.121618): 2 (-1)
193915002 SEND MSG Agent 0 (-0.0615601, 0.450122, 0.0969427, 0.106944): 4 (-0.1)
193915002 SEND MSG Agent 0 WARN of other user 2 affects responsiveness
                  (0.838259,39.2595)->(0.977991,18.2537)
193915002 SEND MSG Agent 1 (0.346413, 0.450122, 0.121618, 0.106944): 3 (-0.207418)
193915002 SEND MSG Agent 1 WARN of other user 2 affects responsiveness
                  (0.881141,173.123)->(1,80.5357)
```

```
193915002 START VIEW 9/8/100
```

- Agent 2 advocates warning against own user viewing, rejected by software
- Agents 0 and 1 advocate extreme responses that are converted to warning their users.

```
193925001 SEND MSG Agent 1 (-0.927169, 0.861065, 0.116344, 0.106944): 3 (-1)
193925001 SEND MSG Agent 0 (0.569199, -0.927169, 0.0969427, 0.116344): 3 (-0.715939)
193925001 SEND MSG Agent 0 RESTRICT 2 by 1 for 11/5, 11:0
193925001 SEND MSG Agent 2 (0, 0, 0.106944, 0.116344): 0 (0.1)
193925001 SEND MSG      11/5/100 11
```

- Agent 1 auto restrict on own user's remove is rejected.
- Agent 0 auto restricts user 1's DELETE action.

```
193932002 SEND MSG Agent 2 (0.924159, 3.79502e-005, 0.104538, 0.116344): 0 (0.1)
193932002 SEND MSG Agent 0 (3.79502e-005, 0.924159, 0.0969427, 0.104538): 1 (0)
193932002 SEND MSG Agent 0 NOTIFY of other user 2 affects responsiveness
                  (0.898816,30.5044)->(0.925933,19.9567)
193932002 SEND MSG Agent 1 (0.395681, 0.924159, 0.116344, 0.104538): 1 (0)
193932002 SEND MSG Agent 1 NOTIFY of other user 2 affects responsiveness
                  (0.925987,192.227)->(0.927444,79.4392)
193932002 ADD ENTRY 9/8/100 17:00 - 17:30
```

- Agents 0 and 1 notify their users of agent 2's action.

```
193938001 END VIEW 11/5/100
193943001 SEND MSG Agent 1 (0.459892, -0.315945, 0.118654, 0.104538): 4 (-0.1)
193943001 SEND MSG Agent 1 WARN against adding on 10/27 (152)
193943001 SEND MSG Agent 0 (-0.315945, 0.459892, 0.0969427, 0.118654): 4 (-0.1)
193943001 SEND MSG Agent 0 WARN of other user 1 affects responsiveness
                  (0.865972,31.5788)->(1,17.4402)
193943001 SEND MSG Agent 2 (0.461899, 0.459892, 0.104538, 0.118654): 0 (0.1)
193943001 START VIEW 10/27/100
```

- Agent 1 inappropriate extreme response changed to warning against adding any events.
- Agent 0 inappropriate extreme response changed to warning of user 1's presence.

```
193943002 END VIEW 9/8/100
193948002 SEND MSG Agent 2 (0.396492, -0.256921, 0.108664, 0.118654): 4 (-0.1)
193948002 SEND MSG Agent 2 WARN against adding on 11/25 (125)
193948002 SEND MSG Agent 0 (-0.256921, 0.396492, 0.0969427, 0.108664): 4 (-0.1)
193948002 SEND MSG Agent 0 WARN of other user 2 affects responsiveness
                  (0.912702,29.9911)->(1,10.3982)
193948002 SEND MSG Agent 1 (0.514955, 0.396492, 0.118654, 0.108664): 3 (-0.1)
193948002 SEND MSG Agent 1 WARN of other user 2 affects responsiveness
                  (0.85041,262.258)->(0.957571,146.852)
193948002 START VIEW 11/25/100
```

- Agent 2 inappropriate extreme response changed to warning against adding any events.
- Agents 0 and 1 in appropriate extreme responses changed to warnings of user 2's presence.

```
193955001 SEND MSG Agent 1 (0.36407, -0.558226, 0.116888, 0.108664): 2 (-0.00728141)
193955001 SEND MSG Agent 1 WARN that ADD is in conflict (35)
193955001 SEND MSG Agent 0 (-0.558226, 0.36407, 0.0969427, 0.116888): 3 (1)
193955001 SEND MSG Agent 0 RESTRICT 0 by 1 for 10/27, 12:0
193955001 SEND MSG Agent 2 (0, 0, 0.108664, 0.116888): 0 (0.1)
193955001 SEND MSG      10/27/100 12
```

- Agent 1 warns user that ADD creates conflict.
- Agent 0 auto restricts unacceptable ADD by user 1.

```
193965001 END VIEW 10/27/100
```

BIBLIOGRAPHY

Baird, L. (1995). Residual algorithms: Reinforcement learning with function approximation. *Machine learning: Proceedings of the twelfth international conference,* A. Prieditis and S. Russell, eds., 9-12 July 1995.

Banerjee, B., S. Debnath, and S. Sen (1999). Using Bayesian network to aid negotiations among agents. *Working notes of the AAAI-99 workshop on negotiation: Settling conflicts and identifying opportunities (AAAI technical report WS-99-12).* pp. 44-49.

Barto, A., S. Sutton, and C. Anderson (1983). Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics* SMC-13 (5). pp. 834-846.

Boy, G. (1997). Active design documents. *Symposium on designing interactive systems: Proceedings of the conference on designing interactive systems: Processes, practices, methods, and techniques,* 18-20 Aug. 1997, Amsterdam, The Netherlands. pp. 31-36.

Card, S., G. Robertson, and W. York (1996). The WebBook and the Web Forager: An information workspace for the world-wide web. *ACM conference on human factors in software (CHI '96).* pp. 111-117.

Chin, D. (1991). Intelligent interfaces as agents. *Intelligent user interfaces.* J. Sullivan and S. Tyler, eds. pp. 127-206 (Ch. 9). Reading, MA: Addison-Wesley.

Chu-Carroll, J., and S. Carberry (1994). A plan-based model for response generation in collaborative task-oriented dialogues. *Proceedings of the twelfth national conference on artificial intelligence,* 1-4 Aug. 1994, Seattle, WA. pp. 799-805.

Dean, T., K. Basye, and J. Shewchuk (1993). Reinforcement learning for planning and control. *Machine learning methods for planning.* S. Minton, ed. pp. 67-92. San Mateo, CA: Morgan Kaufmann Publishers, Inc.

Decker, K., and V. Lesser (1995). Designing a family of coordination algorithms. *Proceedings of the 13th international workshop on distributed artificial intelligence,* Lake Quinalt, WA. pp. 65-84.

DiMicco, J., and W. Bender (2004). Second Messenger: Increasing the visibility of minority viewpoints with a face-to-face collaborative tool. *ACM conference on intelligent user interfaces,* Jan. 2004, Madeira, Portugal.

Dix, A. (1994). Computer supported cooperative work: A framework. *Design issues in CSCW*. D. Rosenberg and C. Hutchison, eds. pp. 9-26. London: Springer-Verlag.

Dommel, H., and J. Aceves (1997). Floor control for multimedia conferencing and collaboration. *ACM Multimedia Systems* 5 (1). pp. 23-38.

Dourish, P., and V. Bellotti (1992). Awareness and coordination in shared workspaces. *Proceedings of ACM CSCW '92 conference on computer supported cooperative work,* November 1992, Toronto, Canada. pp. 23-38.

Durfee, E., V. Lesser, and D. Corkill (1989). Trends in cooperative distributed problem solving. *IEEE Transactions on Knowledge and Data Engineering* 1 (1). pp. 63-83.

Fenster, M., S. Kraus, and J. Rosenschein (1995). Coordination without communication: Experimental validation of focal point techniques. *Proceeding of the international conference on multiagent systems,* 12-14 June 1995, San Francisco, CA. pp. 102-108.

Foner, L. (1997). Entertaining agents: A sociological case study. *Proceedings of the first international conference on autonomous agents,* Marina del Rey, CA. pp. 122-129.

Garrido, L., R. Brena, and K. Sycara (1996). Cognitive modeling and group adaptation in intelligent multi-agent meeting scheduling. *First Iberoamerican workshop on distributed artificial intelligence and multi-agent systems.* C. Lemaître, ed. LANIA, UV. pp. 55-72.

Gaver, W., R. Smith, and T. O'Shea (1991). Effective sounds in complex systems: the ARKola simulation. *Proceedings of CHI `91 conference*. pp 85-90.

Goonatilake, S., and S. Khebbal (1995). Intelligent hybrid systems: Issues, classes and future trends. *Intelligent hybrid systems.* S. Goonatilake and S. Khebbal, eds. London, UK: Wiley.

Greenberg, S. (1991). Personalizable groupware: Accomodating individual roles and group differences. *Proceedings of the European conference of computer supported cooperative work (ECSCW '91)*, 24-27 Sept. 1991 Amsterdam. pp. 17-32.

Greenberg, S., C. Gutwin, and A. Cockburn (1996). Awareness through fisheye views in relaxed-WYSIWIS groupware. *Proceedings of graphics interface*, 21-24 May 1996, Toronto, Canada. pp. 28-38.

Gruber, T. (1991). The role of common ontology in achieving sharable, reusable knowledge bases. *Principles of knowledge representation and reasoning: Proceedings of the second international conference.* pp. 601-602.

Grudin, J. (1994). Groupware and social dynamics: eight challenges for developers. *Communications of the ACM,* 37 (1). pp. 92-105.

Gutwin, C., and S. Greenberg (1998). Effects of awareness support on groupware usability. *Proceedings of the CHI '98 conference on human factors in computing systems.* pp. 511-518.

Gutwin, C., and S. Greenberg (2004). The importance of awareness for team cognition in distributed collaboration. *Team cognition: Understanding the factors that drive process and performance.* E. Salas and S. Fiore, eds. pp. 177-201. Washington: APA Press.

Gutwin, C., S. Greenberg, and M. Roseman (1996). Supporting awareness of others in groupware. *Conference on human factors in computing systems,* Vancouver, Canada. p. 205.

Haykin, S. (1994). *Neural networks: A comprehensive foundation*. Englewood Cliffs, NJ: Macmillan College Publishing Company, Inc.

Hill, J., and C. Gutwin (2003). Awareness support in a groupware widget toolkit. *Conference on supporting group work: Proceedings of the 2003 international ACM SIGGROUP conference on supporting group work,* Sanibel Island, Florida. pp. 258-267.

Huhns, M., and M. Singh (1998). Agents and multiagent systems: Themes, approaches, and challenges. *Readings in agents*. pp. 1-23. San Francisco: Morgan Kaufmann Publishers.

Huhns, M., M. Singh, and T. Ksiezyk (1994). Global information management via local autonomous agents. *Proceedings of the ICOT international symposium on fifth generation computer systems: Workshop on heterogeneous cooperative knowledge bases.* pp. 1-15.

Humphrys, M. (1996). Action selection methods using reinforcement learning. *Animals to animats 4: 4th international conference on simulation of adaptive behavior,* 9-13 Sept. 1996, Cape Cod, MA.

Isbister, K., H. Nakanishi, T. Ishida, and C. Nass (2000). Helper agent: Designing an assistant for human-human interaction in a virtual meeting space. *CHI 2000 conference proceedings: Conference on human factors in computing systems*, 1-6 April 2000, The Hague, Netherlands. pp. 57-64.

Jackson, P. (1999). *Introduction to expert systems*, 3rd ed. London: Addison Wesley Longman Limited.

Johnson-Lenz, P., and T. Johnson-Lenz (1991). Post-mechanistic groupware primitives: Rhythms, boundaries and containers. *Computer-supported cooperative work and groupware.* S. Greenberg, ed. pp. 277-285. San Diego: Academic Press, Ltd.

Jourdan, J., L. Dent, J. McDermott, T. Mitchell, and D. Zabowski (1993). Interfaces that learn: A learning apprentice for calendar management. *Machine learning methods for planning.* S. Minton, ed. pp. 31-66. San Mateo, CA: Morgan Kaufmann Publishers.

Klein, M. (1994). Computer supported conflict management in design teams. *Design issues in CSCW*. D. Rosenberg and C. Hutchison, eds. London: Springer-Verlag.

Kolodner, J. (1993). *Case-based reasoning.* San Mateo, CA: Morgan Kaufmann.

Kraus, S., K. Sycara, and A. Evenchik (1998). Reaching agreements through argumentation: A logical model and implementation. *Artificial Intelligence* 104 (1-2). pp. 1-69.

Krogsæter, M., and C. Thomas (1994). Adaptivity: System-initiated individualization. *Adaptive user support: Ergonomic design of manually and automatically adaptable software*, R. Oppermann, ed. pp. 69-88. Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.

Kuvayev, L., and R. Sutton (1997). Approximation in model-based learning. *ICML '97 workshop on modeling in reinforcement learning*, July 1997, Vanderbilt University, Nashville, Tennessee.

Lâassri, B., H. Lâassri, and V. Lesser (1990). Negotiation and its role in cooperative distributed problem solving. *Proceedings of the tenth international workshop on distributed artificial intelligence.*

Lamport, L. (1978). Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM* 21 (7). pp. 558-565.

Lee, C.-H., and C.-C. Teng (2000). Identification and control of dynamic systems using recurrent fuzzy neural networks. *IEEE Transactions on Fuzzy Systems* 8 (4). pp. 49-366.

Lenzmann, B., and I. Wachsmuth (1997). Contract-net-based learning in a user-adaptive interface agency. *Distributed artificial intelligence meets machine learning: Learning in multi-agent environments.* pp. 202-222. Berlin: Springer.

Littman, M., and D. Ackley (1991). Adaptation in constant utility non-stationary environments. *Proceedings of the fourth international conference on genetic algorithms,* July 1991, San Diego, CA. pp. 136-142.

Long, T. (1998). The optimization assistant – helping engineers explore designs through collaboration. *Proceedings of the fourth international conference on intelligent user interfaces,* Dec. 1998.

Luger, G. (2002). *Artificial intelligence: Structures and strategies for complex problem solving.* Edinburgh Gate, Harlow, Essex: Pearson Education Limited.

Lutz, E., H. Kleist-Retzow, and K. Hoernig (1990). MAFIA – An active mail-filter agent for an intelligent document processing support. *Multi-user interfaces and applications.* S. Gibbs and A. Verrijn-Stuart, eds. pp. 235-251. Amsterdam: Elsevier North Holland.

Maes, P. (1994). Agents that reduce work and information overload. *Communications of the ACM* 37(7). pp. 31-40.

Malcolm, C., and T. Smithers (1990). Symbol grounding via a hybrid architecture in an autonomous assembly system. *Designing autonomous agents.* P. Maes, ed. pp. 123-144. Cambridge: MIT Press.

Malone, T., K. Grant, and K. Lai (1997). Agents for information sharing and coordination: A history and some reflections. *Software agents.* J. Bradshaw, ed. pp. 109-144. Cambridge: AAAI Press / MIT Press.

Maren, A., C. Harston, and R. Pap (1990). *Handbook of neural computing applications*. San Diego: Academic Press, Inc.

Maresh, J., and D. Wastell (1990). Process modeling and CSCW: An application of IPSE technology to medical office work. *Human-computer interaction – INTERACT '90.* pp. 849-852. Amsterdam: Elsevier Science Publishers.

Masters, T. (1995). *Advanced algorithms for neural networks: A C++ sourcebook.* New York: John Wiley & Sons, Inc.

Minton, S., and M. Zweben (1993). Learning, planning, and scheduling: An overview. *Machine learning methods for planning.* S. Minton, ed. pp. 1-30. San Mateo, CA: Morgan Kaufmann Publishers, Inc.

Mitchell, T., R. Caruana, J. McDermott, and D. Zabowski (1994). Experience with a learning personal assistant. *Communications of the ACM* 37 (7), July 1994. pp. 81-91.

Nardi, B., J. Miller, and D. Wright (1998). Collaborative, programmable, intelligent agents. *Communications of the ACM* 41 (3), March 1998. pp. 96-104.

Olson, J., G. Olson, and D. Meader (1995). What mix of video and audio is useful for small groups doing remote real-time design work? *Proceedings of ACM CHI'95 conference on human factors in computing systems.* pp. 362-368.

Paetau, M. (1994). Configurative technology: Adaptation to social systems dynamism. *Adaptive user support: Ergonomic design of manually and automatically adaptable software.* R. Oppermann, ed. pp. 197-202. Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.

Peng, Y., Z. Zhou, and S. Cho (1999). Constructing belief networks from realistic data. *International Journal of Intelligent Systems* 14 (7). pp. 671-695.

Pham, L., and H. Pham (2000). Software reliability models with time-dependent hazard function based on Bayesian approach. *IEEE Transactions on Systems, Man, and Cybernetics, Part A: Systems and Humans* 30 (1). pp. 25-35.

Pierguido, V., and M. Dorigo (1994). Training and delayed reinforcements in Q-learning agents. *Technical report IRIDIA/1994-14,* Université Libre de Bruxelles, Belgium.

Pierreval, H., and J.-L. Paris (2000). Distributed evolutionary algorithms for simulation optimization. *IEEE Transactions on Systems, Man, and Cybernetics, Part A: Systems and Humans* 30 (1). pp. 15-24.

Procter, R., A. McKinlay, R. Woodburn, and O. Masting (1994). Coordination issues in tools for CSCW. *Design issues in CSCW,* D. Rosenberg and C. Hutchison, eds. pp. 119-138. London: Springer-Verlag.

Rabin, M. (1982). N-process mutual exclusion with bounded waiting by 4 log2 n-valued shared variable. *JCSS* 25 (1). pp. 66-75.

Rao, R., and S. Card (1995). Exploring large tables with the table lens. *Proceedings of ACM CHI'95 conference on human factors in computing systems,* Videos, Vol. 2, pp. 403-404.

Robertson, G., S. Card, and J. Mackinlay (1993). Information visualization using 3D interactive animation. *Communications of the ACM* 36 (4). pp. 56-71.

Rodden, T. (1996). Populating the application: A model of awareness for cooperative applications. *CSCW '96: Proceedings of the ACM 1996 conference on computer supported cooperative work,* M. Ackerman, ed., 16-20 Nov. 1996, Boston, MA. pp. 87-96.

Rosenschein, J., and G. Zlotkin (1994). Designing conventions for automated negotiation. *AI Magazine* 15 (3). pp. 29-46.

Ruvini, J., C. Fagot (1998). IBHYS: a new approach to learn users' habits. *Proceedings of ICTAI '98.* pp. 200-207.

Sandholm, T., and V. Lesser (1995). Issues in automated negotiations and electronic commerce: Extending the contract net framework. *Proceedings of the international conference on multiagent systems.* 12-14 June 1995, San Francisco, CA. pp. 328-335.

Schlichter, J., M. Koch, and M. Buerger (1998). Workspace awareness for distributed teams. *Coordination technology for collaborative applications: Organizations, processes, and agents (Lecture notes in computer science* 1364). W. Conen and G. Neumann, eds. pp. 199-218. New York: Springer-Verlag.

Scrivener, S., S. Clark, and N. Keen (1994). The role of replication in the development of remote CSCW systems. *Design issues in CSCW.* D. Rosenberg and C. Hutchison, eds. London: Springer-Verlag London Limited.

Seel, N., G. Gilbert, and M. Morris (1990). A project-orientated view of CSCW. *Human-computer interaction – INTERACT '90.* pp. 903-908.

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

Sen, S., T. Haynes, and N. Arora (1997). Satisfying user preferences while negotiating meetings. *International Journal of Human-Computer Studies* 47. pp. 407-427.

Sen, S., and M. Sekaran (1998). Individual learning of coordination knowledge. *Journal of Experimental & Theoretical Artificial Intelligence* 10. pp. 333-356.

Sen, S., M. Sekaran, and J. Hale (1994). Learning to coordinate without sharing information. *Proceedings of the twelfth national conference on artificial intelligence,* Seattle, WA. pp. 426-431.

Shiozawa, H., J. Noda, K. Okada, and Y. Matsushita (1999). Perspective layered workspace for collaborative work. *1999 international workshops on parallel processing,* 21-24 Sept. 1999, Keio University, Wakamatsu, Japan. p. 80.

Silverman, B. (1992). Survey of expert critiquing systems: Practical and theoretical frontiers. *Communications of the ACM* 35 (4). pp. 106-127.

Singh, S. (1992). The efficient learning of multiple task sequences. *Advances in neural information processing systems 4*. B. Spatz, ed. pp. 251-258. San Mateo, CA: Morgan Kaufmann.

Stasser, G., and L. Taylor (1991). Speaking turns in face-to-face discussions. *Journal of Personality and Social Psychology* 60 (5). pp. 675-684.

Sutton, R. (1995). TD models: Modeling the world at a mixture of time scales. *Proceedings of the 12th international conference on machine learning*.

Sutton, R. (1996). Generalization in reinforcement learning: Successful examples using sparse coarse coding. *Advances in neural information systems 8.* Cambridge: MIT Press.

Totterdell, P., P. Rautenbach, A. Wilkinson, and S. Anderson (1990). Adaptive interface techniques. *Adaptive user interfaces.* D. Browne, P. Totterdell, and M. Norman, eds. pp. 139-157. London: Academic Press Ltd.

Valacich, J., A. Dennis, and J. Nunamaker (1991). Electronic meeting support: The GroupSystems concept. *International Journal of Man-Machine Studies* 34. pp. 261-282.

Voorhees, E. (1994). Software agents for information retrieval. *Software agents – Papers from the 1994 spring symposium (Technical report SS-94-03),* O. Etzioni, ed., March 1994. pp. 126-129.

Wasserman, P. (1993). *Advanced methods in neural computing.* New York: Van Nostrand Reinhold.

Watkins, C., and P. Dayan (1992). Q-learning. *Machine Learning* 8. pp. 279-292.

Wellman, M. (1995). A computational market model for distributed configuration design. *AI EDAM* 9. pp. 125-133.

White, G. (1990). A formal method for specifying temporal properties of the multi-user interface. *Multi-user interfaces and applications.* S. Gibbs and A. Verrijn-Stuart, eds. pp. 49-59. Amsterdam: Elsevier Science Publishers.

Winnett, M., R. Malyan, and P. Barnwell (1994). ShareLib: A toolkit for CSCW applications programming using X Windows. *Design Issues in CSCW.* D. Rosenberg and C. Hutchison, eds. London: Springer-Verlag London Limited.

Woitass, M. (1990). Coordination of intelligent office agents – Applied to meeting scheduling. *Multi-user interfaces and applications.* S. Gibbs and A. Verrijn-Stuart, eds. pp. 371-386. Amsterdam: Elsevier Science Publishers.