INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

UMI®

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

A STUDY OF SOFTWARE AGENT DESIGN AND IMPLEMENTATION

Qin Huang

A thesis
in
The Department
of
Computer Science

Presented in Partial Fulfillment of the Requirements
For the Degree of Master of Computer Science
Concordia University
Montréal, Québec, Canada

June 1997

National Library of Canada
Acquisitions and Bibliographic Services
395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque nationale du Canada
Acquisitions et services bibliographiques
395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

Our file Notre référence

0-612-40220-7

Canada

Abstract

A Study of Software Agent Design and Implementation

Qin Huang

Software agents are the result of the development in the fields of human computer interaction, artificial intelligent and cognitive science. Recently it was the focus of both computer science research and the software industry. It has become clear that agents can take many roles in helping people complete their tasks. But, agents in real life have not been perfect things that can satisfy everybody. Some problems related to agent's usability have arisen, such as, whether an agent can provide useful help without annoying its users? Whether an agent can be trusted? Building agents that will be satisfied with and trust is important issue in the agents' design.

In this thesis, different approaches used in building agents and some current existing agents are introduced. Human-learning, machine-learning, and feedback learning are analysed. The thesis describes extensions made to an existing telephone agent. The enhanced telephone agent is presented by applying machine learning and feedback learning techniques. A demonstration is also given to show how the enhanced agent can make suggestions to its user unobtrusively with appropriate frequency and adjust its behaviour to support its user's daily work. Furthermore, a methodology for agents' development is derived from this case study. It is emphasised that through design and implementation to testing and maintenance, user's needs and user's satisfaction should always be the main concern of agent's designers. The thesis provides a frame for agents' development, which allows the construction of agents that meet these criteria.

Acknowledgments

My deepest gratitude is extended to my supervisor, Dr. Peter Grogono. His continued guidance, encouragement, enthusiasm, and support are greatly appreciated, and will be genuinely missed. I feel very lucky to have been one of his students.

I appreciate the many helpful comments and suggestions from my committee members, Dr. Rajjan Shinghal and Dr. Ferhat Khendek. I would also like to thank Pardo Mustillo and Gliff Grossner for their helpful suggestions during the implementation phase.

My sincere thanks to my parents for their unwavering love and support. My particular thanks also to my parents-in-law who have taken the utmost care of my son and me during the last six months. They made my completing this thesis possible.

Many thanks to all my friends for their encouragement and help. I would like to specially thank Xiaoyang Zhu and Yun Wang for their thoughtful discussions.

Finally, I would like to express my indebtedness to my beloved husband, Nan, for his support, patience and faith in me, and to our lovely son, Keith, for his encouragement through his innocent smile! I wish to dedicate this work to Nan and Keith. They gave me more pleasure which could not be expressed in words.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

During the course of civilization, human beings have been making ever greater efforts to emancipate their physical labour force. The industrial revolution during 18th century, marked by the general introduction of power-driven machinery, improved productive forces enormously and promoted social development. Today, it is widely recognized that the "computer revolution"in this century, marked by the widespread application of information-driven machinery, enables information to play an increasingly important role in our daily lives, and furthermore emancipates our minds. As information technology has pervaded many fields, people have to deal with enormous information in their day-to-day work. On the other hand, many people feel overwhelmed by the amount of searching and managing works for these information coming from the "information highway". Nowadays, many kinds of information is available through the internet. Obviously, it is more convenient for users to search information other than by traditional means, such as searching through the books in libraries. However to find out needed information or to categorize sets of information has become more complicated and difficult. Since the information existing is so vast, searching for the useful information just like fishing for a needle in the ocean. Although there are a number of existing search engines, they need some form of supervision by the user — i.e. they need constant refinement to filter out irrelevant results. Hence there is a demand of having an assistant who can handle all needed tasks autonomously. An agent can be this assistant.

1.1 Software Agents

What is an agent? The definition of the term "agent" in a dictionary is interpreted as "An entity authorized to act on another's behalf". When used in the computer science domain, software agents are computer programs that act to accomplish tasks on behalf of their users in order to ease the burdens that computers put on people [Mae95]. Figure 1 shows the basic concept of an intelligent agent.

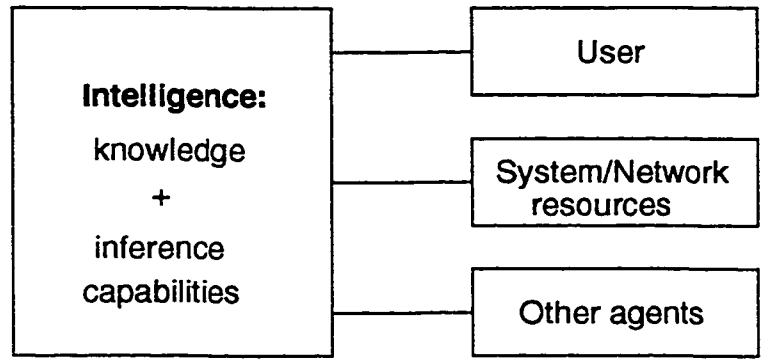

Figure 1: Basic Concept of An Intelligent Agent (Adapted from Mustillo,1996)

It was in the 1980's that the concept of "Autonomous Agent" was introduced into Artificial Intelligence. Within the last decade, the software agent, the result of the development in the fields of human computer interaction, artificial intelligence and cognitive science, is one of the fastest growing areas of research and new application development. Nowadays, the agent concept is put into practice by many universities, research groups, and companies. A lot of research work is being done in this field and the work has shown how it can contribute to human life in many aspects. It has become clear that agents can take many roles in helping people to complete their tasks. For example, agents can recommend books [Fey93], music [SM94] or other forms of entertainment to their owners; they can assist the user with electronic mail [Mae94]; they can learn the user's interests in order to assist them in browsing the World Wide Web [Lie95]; they can manage user's daily telephone communications [Wil94], etc. The functionality of agents is obvious — they behave as personal secretaries: they gather knowledge about the tasks, habits and preferences of their users, and then use this knowledge to adjust their behaviour to make their users satisfied and to increase

their productivity.

1.2 A Typology of Agents

As the multiplicity of roles that agents can play increases, and as more and more agents are being developed, some researchers have tried to categorize agents for further analysis and research. Nissen categorized agents in different groups which reflect different characteristics of intelligent agents [Nis95]. For example, agents which can look for specific information or events for the user are called watcher agents, agents which can adapt to an individual's preferences by learning from a user's past behaviour are learning agents, agents which can do comparison shopping and finding the best price for an item are shopping agents, agents which can be used for network management are helping agents, etc. King presented a role-specific classification of agents, such as report agents, analysis and design agents, etc [Kin95]. Nwana categorized the current existing agents with multiple metrics [Nwa96]. Since agents exist in a multi-dimensional space, Nwana's method is much more logical and clear.

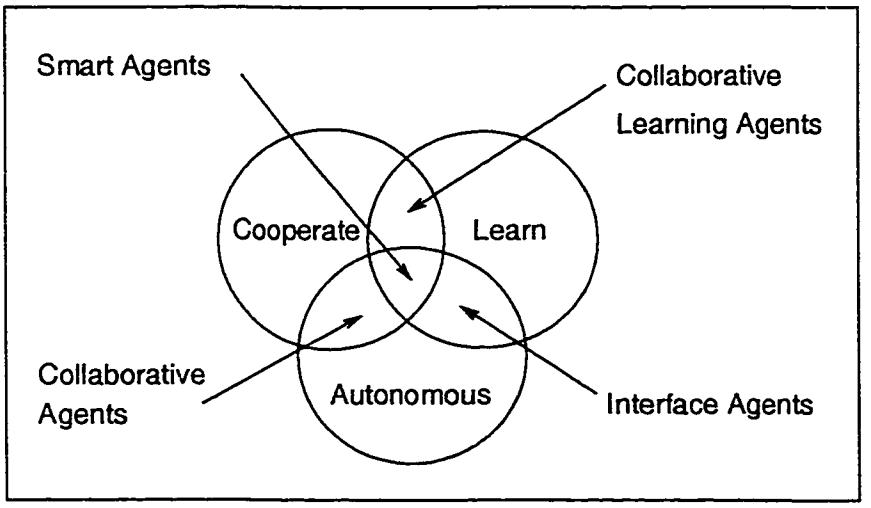

Figure 2: A Part View of an Agent Typology — Classify Agents Based on Three Primary Attributes (Adapted from Nwana,1996)

Nwana pointed out that agents may be classified by their mobility, activity, primary attributes and roles. Based on mobility, agents can be classified as static or

mobile. Basing on activity, they also can be classified as either deliberative or reactive. The three ideal and primary attributes of agents are autonomy, learning and cooperation. Figure 2 derives four types of agents according to the three basic characteristics of agents: *collaborative learning agents, interface agents, collaborative agents, and smart agents.* By combining the other two characteristics (mobility and activity) in conjunction with these four agent types, there are total twelve categories of agents. Since agents may also be classified by their roles, Nwana gives a relatively all-round view of the current existing agent field.

In the following part of this thesis, we will focus on the study of *Interface Agents*, which Maes defined as:

> "computer programs that employ Artificial Intelligence techniques to provide active assistance to a user with computer-based task. ...The metaphor used is that of a personal assistant who is collaborating with the user in the same work environment." (quote from [Mae94]).

As shown in Figure 2, interface agents have two important characteristics: *autonomy* and *learning.* This means that the agents can not only operate on their own without being guided by their users all the time, but also learn when they interact with their external environment. These agents provide an easy-to-use interface to computers and complex systems. So both novices and experts can operate them easily.

1.3 Objective and Scope of this Thesis

The objective of this thesis is to get a better understanding for agents from the different points of view of the fields of human-computer interaction, artificial intelligence, and human cognitive science, which are related to the development of intelligent agent field. The study of the intelligent agent could benefit from the study of these fields, since the intelligent agent field is in the overlapped area of these three fields. It tells us that to design a good agent the designer should not only choose a good algorithm, but also keep cognitive processing skills and limitations of the user in mind. A general survey of current learning agents is also given in this thesis. And an example of making enhancements to a telephone-based agent is presented as well. From this example, we can also get some general ideas for deriving a methodology of agents'

development.

Human-computer interaction investigates user interface design techniques, including the incorporation of intelligence into the interface. Intelligent agents are part of the future of the user interface. Since the use of computers is no longer confined to the experts, many users are now people who have little technical skill. Intelligent agents can help these users to make effective use of computers. Moreover, agents are tireless assistants which can work for their owners 24 hours a day, 7 days a week. Agents will increase the productivity of the users.

The concept of the intelligent agent focuses on the word "intelligent", which means the agent can acquire new knowledge through learning over time and communicate with the user and system/network resources. The more advanced intelligent agents may also can communicate and cooperate with other agents to carry out tasks beyond the capability of a single agent. From these points of view, agents are related to the cognitive process (problem solving, learning and reasoning). In this thesis, the related cognitive aspects of human-computer interaction will be discussed, as well as the process of man-learning and machine-learning.

Although the term of "agent" sounds attractive and has begun to be used in more and more commercial products, the agent in real life has not been a perfect thing that can satisfy everybody. Some people don't like agents because they sometimes make simple things too complicated, and sometimes agents even annoy their users. Moreover, the features provided by a software agent are limited by the user's capability of learning the agent feature set and the effort required to use a particular feature. So how to build an agent which the owner will like is an important issue in the agent's design.

By applying machine learning techniques, an agent can eventually gather "enough" knowledge by observing a user's behaviour and then make suggestions to its user. This technique has been applied to many agents' design. For example, Robin Kozierok and Pattie Maes [KM93, Koz93] implemented a meeting scheduling agent that detected patterns in users' behaviour in order to assist the user by automating the scheduling task or making suggestions in the future; Charles River Analytics implemented an

agent "Open Sesame!"[CSJ+96], which is an user interface learning agent sitting in the background and observing user actions, finding repetitive patterns, and automating them upon approval. A machine learning approach has also been used in telephone-based agents in a research phase by Roland Younes recently. He named the agent FATMA (Futuristic Automated Telecommunication Management Agent), which is a command-driven agent for call management [You97]. In this thesis, machine-learning and feedback learning approaches are employed to build an enhanced agent based on the FATMA model. Though this example some solutions for the question mentioned above are discussed. The main goals for making enhancements for FATMA agent are to present how the process of human cognitive can be integrated in the agent's design, and how the feedback learning can improve agent's performance — by avoiding annoying subscribers. The enhanced agent can specify the differences among subscribers and to adjust its behaviour according to the feedback provided by the subscriber when it makes suggestions.

1.4 Organization of this Thesis

In next chapter, a literature review of the development in fields of Human-Computer Interaction, Cognitive Science and Artificial Intelligence is given. The process of human-learning and machine-learning, and feedback learning will be analysed. Finally, it introduces some current learning agents and the different approaches in building intelligent agents.

Chapter 3 first introduces an existing electronic assistant — Wildfire and a proactive telephone-based agent — FATMA which is a Wildfire-like agent. The chapter then presents an enhanced FATMA agent which can make suggestions to its subscriber unobtrusively with appropriate frequency by applying feedback learning techniques.

Chapter 4 proposed two predefined scenarios demonstrating how the enhanced FATMA adjusts its behaviour to support the subscriber's daily work more efficiently by adapting feedback learning approach.

Chapter 5 focuses on deriving a methodology for building agents satisfied by users and presenting the lifecycle of agents' development.

Chapter 6 draws conclusions based on the survey and the experience of implementation of the enhanced call management agent — FATMA. It also contains suggestions for future work.

Chapter 2

Background

As the agent field crosses three domains of Human-Computer Interaction, Cognitive Science and Artificial Intelligence, these three domains can be seen as fundamental to the agent field. It is necessary to overview these three fields in order to get a better understanding for agents, and to build agents that will satisfy both designers and users. The first portion of this chapter is a literature review of these three areas. The second portion will introduce several different ways of building agents. The third portion is a survey of learning interface agents existing currently.

2.1 Literature Review

2.1.1 The Development of Human-Computer Interaction

Human Factors Engineering had its birth during World War II. It is defined as "the scientific study of the interaction between people, machines, and their work environment with emphasis on organizational and psychological interaction" [Mar90]. By using the knowledge gained from this study to create systems and work environments, people can be more productive and be more satisfied with their work life.

The field of Human Computer Interaction (HCI) developed in the early 1970s as a part of the field of Human Factors. HCI is interested in user-interface design techniques, including the incorporation of intelligence into the interface. During the first decade of research in HCI, the mechanical aspects of the interaction between human and computer were the main interests of this field. In the last decade, researchers

began to address the cognitive issues of HCI after they found that integrating "intelligence" into interface design is a good way to enhance human computer interaction. Since then, HCI has become a collaborative research area among computer scientists, human factors engineers, and cognitive scientists.

History of HCI

For better understanding about the current state of HCI, let's firstly look at how it has developed. Since the 1940's, when the first modern electronic computers appeared, the way in which humans interact with computers has changed greatly. The development so far has been described by Nielsen in the following stages [Nie93]:

1945–1955 Batch Systems.

There was no real interaction between users and computers. Computer systems designers were designing systems for their own use or for other experts, since all the programs were written in machine language and computers executed user's commands in batches.

1955–1965 First Generation of HCI — Line-Oriented Systems.

The users must conform to the machine since they have to remember all commands in computer required format. Assembly language and high-order languages were introduced. The user could only interact with the computer on the command line and the text was frozen once it had scrolled above the command line. It was a one-dimensional interface since the user was allowed to modify only the last line. The machine was reserved only for professional users.

1965–1980 Second Generation of HCI — Full-Screen Systems.

After the electronic mouse was invented in 1963, new interaction styles were introduced, such as menus, function keys, and form-filling dialogues in which the user could edit a number of fields in any sequence. These techniques freed the user from remembering options. The user communities expanded to include people who were specialists in their disciplines but not computer specialists (such as accountant, administrative support people). High level languages were introduced. The full-screen interface allowed the user to move about in two dimensions in a screen.

1980–1995 Third Generation of HCI — Graphical User Interface (GUI).

User Interface design was considered as separate from the application functions, i.e., the core of the system was hidden and the people interacted with the system through an attractive and simple top layer. GUIs enabled a much more abstract view of the operation of the computer. The mouse became very important, since the primary interaction style of GUI was direct manipulation (point-and-click interface). Interaction between human and computer became more intuitive. Computers could be used by both novice and expert users. Consequently, the popularity of computers, particularly that of home computers, was largely due to the introduction and further developments of the GUI. The graphical interface is often called a "two-and-a-half dimensional" interface. Although we can have overlapping windows on the screen, we can't see the contents of obscured windows without moving them to the top.

1995– Fourth Generation of HCI — Speech-Based and Gesture-Based Interface.

Speech and gesture UIs bridge the gap between the advanced computer features and the users. Computers are widely used in many fields and by more and more untrained users. Interfaces becomes three-dimensional by adding sound and gesture. Intelligent agents are introduced as part of the future of the user interface.

From the view of the history of HCI development, the software designer realized that users are becoming more and more important in computer applications. Consequently, the position of user interface design has changed from a nonessential aspect to a critical one in the development of computer systems, since, to some extent, it reflects the capabilities of the entire system.

Goals of HCI

The goals of HCI research are divided into four categories by Landauer [Lan88]:

1. comparison of existing systems or features;
2. invention or design of new systems or features;
3. discovering and testing relevant scientific principle;
4. establishing guidelines and standards.

The conduct of HCI research is to guide the development of useful, desirable, and usable computer systems. A good system should not only contain the functions that allow people to do the things which they want to do, but also be easy to learn and be well liked by people. An important property of a user interface is *usability*, which determines how easily the users can use the functions provided by the system [Nie93]. Figure 3 gives a clear description for the relationship between the system acceptability and usability.

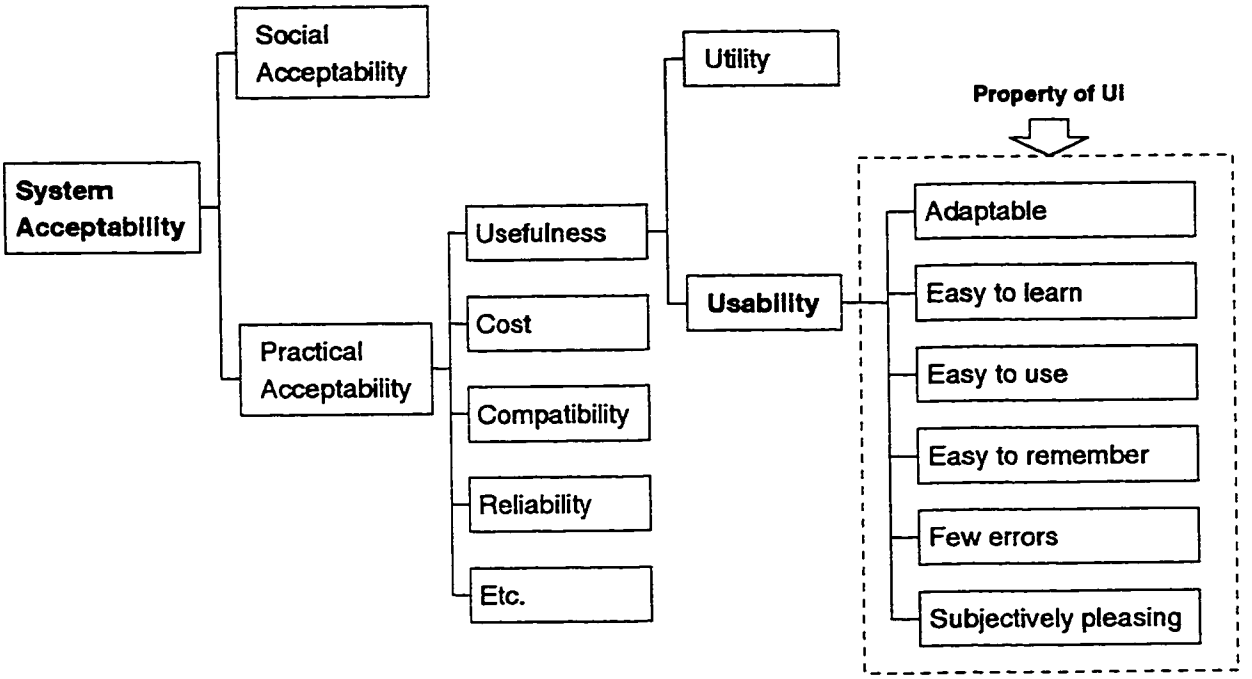

Figure 3: Usability — The Road to System Acceptability (Adapted from Nielsen,1993)

It has been demonstrated that a good interface can increase the ease of use, reduce the time and effort required to learn a new system, decrease the number of errors committed by the user, increase satisfaction of the user with the system, help the users absorb information, and improve the user's retention of the system's functionality and syntactic aspects. All of these factors prevent users from wasting time re-learning the application. All of these advantages impact the productivity of the user [BKL88, GBL+87, Nic81, Nor86, Shn87]. Bailey *et al.* demonstrated that the interface design of their application improved productivity by as much as 77% [BKL88].

An appropriate interface is also critical to the survival of the hardware and software product. A product that is aimed at the mass market cannot hope to sell well if it is not easy to use and easy to learn. Functionality alone is no longer sufficient. The development of walk-up-and-use systems such as automatic bank tellers make it necessary to develop applications that require minimum training. Nickerson [Nic81] conducted a study to determine why people did not use a system when it could benefit them. Seven of the ten detected reasons are directly related to the human-computer interface, such as difficulties of starting and stopping, training and user support, command languages, hard to follow the system, etc. Consequently, a poorly designed interface can ruin the chances of a product to gain acceptability in the marketplace.

Realizing the computer's interface significant impact on the user's productivity and the product's market success, many researchers have been motivated to study human-computer interaction in depth.

Although a great effort has been made in the HCI research, there is evidence that designing a good interface is still extremely difficult, time consuming, and can easily double the cost of developing a system [BD81]. The designer must be aware of the design alternatives and their impact on the user. This requires an in-depth knowledge of the psychology of the user. An adequate model of the brain, a complete human psychology, or even a comprehensive study of human factors in HCI could greatly facilitate the task of the interface designer. Unfortunately, none of these three items are available, and consequently, designing a good interface is quite challenging. This is another motivation of the researchers working in this field.

Agent as the Trend of HCI Development

With computers being indispensable in all areas of society, the interface agent as a future part of HCI development is inevitable. Direct manipulation, the current human-computer interaction metaphor, requires the user to initiate all tasks explicitly and to monitor all events. It has limitations as user population grows and diversify, since it is difficult to learn and to use. But an interface agent, the metaphor of personal

assistant, can collaborate with the user in the same work environment, and can learn user habits and preferences and adapt to individuals [Mae94]. It makes easier for both novices and experts to handle their computer-based tasks and other complex systems related work. Another advantage is it can make effective use of the computers and networks, and improve the productivity of the user, for instance, the agent of World Wide Web can search for the useful information for its user 24 hours a day, 7 days a week.

Usability is one of the most important issues in agents' design as well as in other HCI designs. The agent have to be easy to use and easy to learn. If agents cannot make their users feeling comfortable to work with them, or the user have to deal with hardware or software problems in order to use them, the agents will not have a bright market future.

2.1.2 The Development of Cognitive Science

The development of computers was a major factor in the emergence of Cognitive Science. Cognitive Science is interested in the psychological aspects of the interaction between humans and computers. It has developed through a merging of artificial intelligence research with other sciences such as cognitive psychology, neurophysiology, and linguistics. Donald Broadbent [Bro58] argued that one could begin to understand phenomena such as perception, attention, and short-term memory by constructing an information processing theory in which information flowed through a cognitive system. In other words, instead of looking at perception, attention, and short-term memory separately, he considered them all as interdependent ingredients in a single cognitive system. Norman expressed the goal of Cognitive Science as the understanding of the principles of intelligent, cognitive behaviour. "This will lead to better understanding of the human mind, of teaching and learning, of mental abilities, and of the development of intelligent devices that can augment human capabilities in important and constructive ways" [Nor81].

The approach taken by Cognitive Science is based on the information processing metaphor of cognition: the models and theories are "computational", so that they are testable by computer simulation. In the eighties, Rumelhart and Norman [RN82]

wrote a computer program to simulate a typist. Unlike earlier studies that focused on optimising the layout of the keyboard, they aimed to get knowledge of information processing skills required of the typist. Once this was known, then the knowledge could be applied in the design of computer assisted training programs for novice typists.

The followings are two main issues studied in Cognitive Science: knowledge representation and learning abilities.

Representations of Knowledge

Mental representation of knowledge is particularly important when we design computer systems to support users' work activities, because the structure and forms of knowledge condensed in computer systems must match the knowledge in the users' mind.

There are basically two kinds of knowledge representations which psychologists call *semantic* and *episodic* knowledge (refer to semantic and episodic memory systems, respectively) [Tul83]. Semantic knowledge refers our de-contextualized memory for facts about the entities and factual relations between entities and concepts in the world, for instance, that a ship has a mast, a funnel, a starboard, a port side, desks, and lifeboats. In contrast, episodic knowledge refers to knowledge about episodes and events, to entities that are marked as happening at a particular time and process, for instance one remembers about the experience of lying on a deck, savouring sea breeze, and enjoying sea air, (or maybe one only remembers about suffering seasickness). Sternberg argued that intelligence is more related to semantic than episodic knowledge [SD79], and research has provided empirical tests of this hypothesis [Tul83]. There are three descriptive models that can be used to represent semantic knowledge: semantic networks, connectionist networks, and production systems.

- Semantic Networks are common methods employed by researchers to represent the contents of human memory. They consist of concepts as nodes, links as relations, and activation strength linked each nodes which represent how strong one node is associated to another. Learning changes the activation values of the links between nodes.

- In connectionist Networks (Neural Networks) the nodes are multiply connected to each other, so each node is a multiple input-output system. They can "learn" to produce a particular output when certain input are given to them.

- Production systems are in the form "IF such-and-such is the case THEN do so-and-so". Production rules are a kind of structured knowledge, which refers to human knowledge of how to do things, rather than just facts and figures. It is widely used in automatic control and artificial intelligence. Production rules can also be applied in high level cognitive processes like problem solving, reasoning, and learning.

These three computational modeling techniques are used by cognitive scientists. They have been applied in knowledge engineering, artificial intelligence, and human-computer interface design. Some computer systems — such as ACT, designed by Jone Anderson to model human memory and learning [For87] — use both production rules and semantic networks working in close interaction.

Human-Learning and Machine-Learning

Knowledge represented by semantic or episodic are not rooted in humans' minds at the beginning. People learn and gain semantic and episodic knowledge while they grow up. With people's experience getting abundant, their knowledge is increased. These refer to the aspect of human-learning. The process of human-learning is valuable to machine-learning research. Only after understanding it, will we be able to tell machines how to learn and absorb knowledge (as humans do), and how to catch up with the changing of the user's mind so as to support the user's day-to-day tasks.

People's learning starts from examples — making general rules from particular instances. That is, we learn from semantic-network or connectionist-network pattern. When we modify our rules to adapt to new situations, we enrich our knowledge and improve our ability of adjusting to the changing circumstances. One mechanism driving the process is called feedback learning, or backward propagation. It describes the process of learning from random pattern to explicit rule pattern. Feedback learning has been proved by many theories (cybernetics, neuropsychology) and practices (in our daily lives, and in experimental psychology).

People can learn more and faster by the support of a computer system; but the computer system must also keep learning from its user to update its functionality and capability, and adjust its behaviour to satisfy user's needs. Otherwise, it cannot support its user in new situations. Obviously, the knowledge programmed by designers is not sufficient for a computer system to fulfil the user's ongoing requirements, learning from users' feedback is a complementary method to gain knowledge. Users' participation in the learning process is a cognitive necessity.

Cognitive Science in HCI Design

Knowledge gained through cognitive science research can be applied to situations of learning in HCI. Since the mental model is an important cognitive aspect of the human-computer interaction [Nor83], HCI design has to take into account relevant characteristics of human learning and user variables.

> "It takes at least three kinds of special knowledge of design an interface: first, knowledge of design, of programming, and of the technology; second, knowledge of people, of the principles of mental computations of communication, and of interaction; and third, expert knowledge of the task that is to be accomplished. Most programmers and designers of computer systems have the first kind of knowledge, but not the second or third. Most psychologists have the second, but not the first or third. And a potential user is apt to have the third, but not the first or second." (quote from Norman [Nor86]).

Norman not only addressed what knowledge is required for designing an interface, but also the different fields which users and designers are good at. To bridge the gap between user and designer, the prototyping approach has been applied in the whole process of design and to support the mutual learning between human and computer.

2.1.3 The Development of Artificial Intelligence

The field of Artificial Intelligence was founded at the Dartmouth Conference of 1956. It is interested in the understanding of principles that makes intelligence possible and thereby creating machines that are "able to do things that people would say require

intelligence" [Jac74].

In earlier days, researchers in the field of AI hoped to "construct an intelligent machine with a perception component that sensed and interpreted the world, a component that allowed the system to learn from its experiences, and a mechanism for solving general problems arising in interactions with the environment. The knowledge database would be provided initially by the human designer, and then develop autonomously as the system learned about the world" [FF87]. During its forty year development, the AI field has changed enormously. The emphasis of the research shifted from a general problem solving approach to a concentration on specialist problem domain in many fields, such as pattern recognition, game playing, semantic information processing, logical reasoning, etc. The development of AI makes computers more useful.

Scope of AI Research

The research of AI is presented in three layers by Kreutzer [KM90] (Figure 4). The inner layer contains the methodological heart of AI programming, the middle layer lists theoretical issues, and the outer layer shows some applications in this field. (Note that Kreutzer does not include learning as part of AI.)

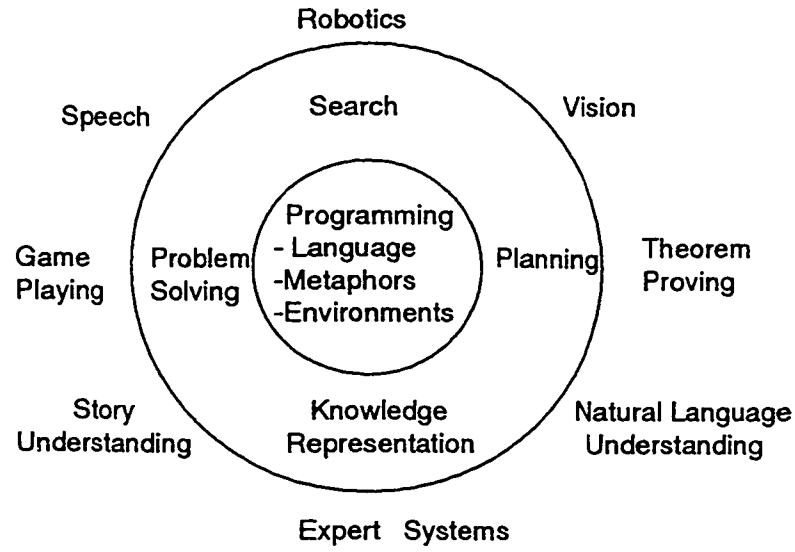

Figure 4: Three Layers of AI Research (Adapted from Kreutzer,1990)

The goal of the research in AI is to discover and describe aspects of human intelligence that can be simulated by machines. At the same time, a benefit obtained from the research in AI is the methodology used to create intelligent computers may also improve human intelligence. Just as the knowledge of information processing in cognitive science can help to make computers intelligent, theories derived with computers in mind may suggest possibilities about methods to educate people better. It is obvious that with the growing influence of the discipline of artificial intelligence, the relationship between the mind and the computer has become even closer.

The Techniques of AI Applied in HCI

As we mentioned earlier, HCI is dealing with the interaction between humans and computers to carry out a task. Integrating intelligence into interface design is a good way to enhance HCI. In this section, we will describe several AI techniques which can be applied on HCI in terms of amplifying the cognition functions of the user.

Expert Systems (Knowledge-Based System)

Expert systems play a major role in many applications of AI research and employ a number of techniques developed in AI. They generally embody a set of expert knowledge together with the rules to apply it across a range of situations. An AI application is "a program that uses large amounts of knowledge about a single domain to achieve a high level of competence in that domain" [FF87] (i.e., it can act as a human advisor acts). In the context of HCI, they can be seen as an application specialist in the service of the user: a mediator and amplifier which translates user intentions into commands, as well as data and information from the system into knowledge that can be presented to the user.

Applications for expert systems include areas as medical diagnosis, statistical data interpretation, analysis of pollution and plant diseases, and many others. Some of the important functions provided in expert systems are making predictions, checking consequences of actions, and maintaining consistency of knowledge. They could serve to amplify the user's cognition and be applied to enhance HCI.

Production Systems (Rule-Based Systems)

There are many problem-solving paradigms created in AI research, which enable the performance of the computers moving toward level of human specialists. Production systems, (also called Rule-based system), is one of the problem-solving paradigms. It is the one that is most popular part used for building expert systems. They use collections of rules to solve problems. There are three components in a production system:

1. Working Memory (WM) — the date representing the current state of the "world" .

2. Production Rules — consisting of an *if* part and a *then* part like the following one:

```
if  <condition 1,  condition 2, ... (in WM)>
 then <action 1, action 2 ...>
```

3. Rule Interpreter — the control module that carries out the matching operation to determine the next rule to be actived.

When all the conditions in a rule are satisfied by the current situation, the rule is triggered and the actions are performed. The effects of the actions changes the conditions so that further rules may be triggered, and so on. When the actions are performed, the rule is fired.

The advantages of the Rule-based systems are: (1) they provide a foundation of knowledge representation; (2) they permit representation of knowledge in a highly modular manner, and are relatively easy to modify, allowing knowledge to be added incrementally; and (3) they reflect the rule-like character of human knowledge. Although rule-based expert systems have been developed in many different domains, they lack many of the characteristics of human experts. For instance: they do not learn; they do not reason on multiple levels; they do not look at problems from different perspectives; and they have no flexible behaviour because their knowledge can be used in only one way.

Nowadays, rule-based systems have been used extensively to model human problem solving by information-processing psychologists. Grosef *et al.* have said that rule-based reasoning will give a great help in the next generation of commercial information agents [GLC+96].

2.2 Software Agents

An intelligent agent has knowledge, and capabilities of inference and communication. Although agents are diverse, they all have similar characteristics [Mus96]. They can

- sense the environment and react to it;
- operate autonomously or semi-autonomously; and
- communicate with the user and system/network resources.

More advanced intelligent agents may also:

- acquire knowledge to improve their behaviours through learning; and
- cooperate with other agents to carry out tasks beyond the capability of a single agent.

Generally speaking, intelligent agents are computer programs that act to accomplish tasks on behalf of their users in a complex, dynamic environment.

When building an intelligent agent, we must consider how the agent:

- acquires knowledge;
- interacts with the user;
- interacts with the system resources; and
- communicates with other agents.

2.2.1 Approaches for Building Intelligent Agents

Currently, there are three main approaches in building intelligent agents. They differ in the technique used for knowledge acquisition. The three approaches of knowledge acquisition are *End-User Programming, Knowledge-Based*, and *Machine-Learning.*

End-User Programming Approach (Rule-Based Reasoning)

This system requires the user to state rules for the agent to follow. The basic concept of this technique is rule-firing, (i.e., rule-based reasoning, mentioned in Section 2.1.3). The user specifies a set of rules, which consist of a trigger and an action. The trigger specifies when the action is to be taken. When the conditions of a rule are satisfied, the agent executes the actions. There are many systems that incorporate this approach, such as PAGES [HEA90], Information Lens [MGL+87], Object Lens [LM88], and Oval [MLF95].

For example, Information Lens helps users to create a rule-based agent for automatically filtering and sorting incoming electronic messages. It can also help users create messages using a set of structured templates that suggest the kinds of information to be included in the messages, so the receiver can specify rules to filter and classify their incoming messages. This technique provides users with a consistent way of specifying the behaviour of agents.

This approach is easy to implement and is especially helpful for controlling, filtering and handling actions since rules can be specified by the user directly. It also enables the user to have an agent that serves his/her individual preference – agent personalization. However, this approach has its limitations. That is, an agent cannot behave beyond the scope which its user designed for it. Whether the agent is successful or not depends on its user's ability to design and program rules. In order to have a well-programmed agent, the user must not only recognize the opportunity to state a rule, and express the rule in the language of the system, but also maintain its rules over time, as his/her habits change. Since the agent can not determine rules dynamically, it may not be able to create a decision for a specific instance of a problem, as most humans do.

Knowledge-Based Approach (Expert System)

As mentioned in Section 2.1.3, a Knowledge-Based approach relies on field experts to program intelligence into an application. That is, the agent is endowed with knowledge by experts who share their knowledge with the machine. Agents built with this approach may be implemented without user involvement. The knowledge-based approach places less burden on users than the end-user programming approach, since it emancipates users from thinking and creating rules. Office CLECK [Mac91] and COKES [KK87] are examples of using this approach.

COKES is a knowledge-based deduction system for the support of office workers at a high-level of functionality and intelligence. The various agents and servers in the system have a knowledge of office work, emulate the behaviour of human office assistants in cooperating with each other to solve problems or complete tasks by passing messages.

This approach requires a very strong domain model. It is not flexible in the ways in which systems use the knowledge, because to abstract expert knowledge so that it becomes applicable to other domains is not yet possible (and may never be). Another disadvantage is that personalized information is not a part of knowledge-based systems. Knowledge-based systems do not adapt to the user's particular habits or preferences. It arrives fully knowledgeable and fully effective. Furthermore, this approach requires time and expertise to create knowledge bases which provide domain knowledge that agents can use to support users' tasks.

Machine Learning Approach

Using Machine Learning approach to acquire agent knowledge is proposed by Maes, of MIT Media Laboratory [Mae94, LMM94]. The agent, which starts out by simply observing user behaviour, can eventually build up enough knowledge and come up with a prediction. A main technique in machine learning, Memory Based Reasoning (MBR) stores a large set of correspondences between situations and actions when the agent observes the user's behaviour. Whenever a new situation occurs, an action is selected by considering "closest" matches between the new situation and stored situation. There is no need for explicit training in MBR approach. The agent merely

watches the user until it believes it can make prediction for future actions. Machine learning enables the user to structure his/her agent implicitly, rather than explicitly.

Maes pointed out that learning agents are particularly applicable when its tasks with the following criteria [Mae94]: (1) the task contains a significant amount of repetitive behaviour; (2) the repetitive behaviour is very different for individual users.

This approach overcomes the disadvantages with End-User Programming approach and Knowledge-Based approach. But it has its own drawbacks [LMM94]: the agent learns slowly, and cannot provide suggestions until it has learned a sufficient number of examples; the agent can not make a decision in a completely new situation, since there is no example of it in agent's memory. Researchers working on this approach focus on searching for various learning schemes and algorithms so that the agent can adapt to its environment very quickly. There are four methods for an agent to improve its knowledge by extending the learning resources [Mae94] are shown in Figure 5:

- learning by observing and imitating the user;
- learning from user feedback;
- learning from examples given explicitly by the user; and
- learning from experienced agents.

The first three ways refer to *learning from the user*, which are easy to understand. The last one refers to *learning from peers*. It is usually used by the agent when it faces unfamiliar situations, or does not have enough confidence in its prediction. The agent may present the situation to other agents, and consult them for help. This method partially fixes the problems mentioned above.

2.3 Current Learning Agent

Currently, there are many applications of learning agents that have been developed or are under development both in commercial and research areas. Here, several examples are selected to give a brief overview of this field.

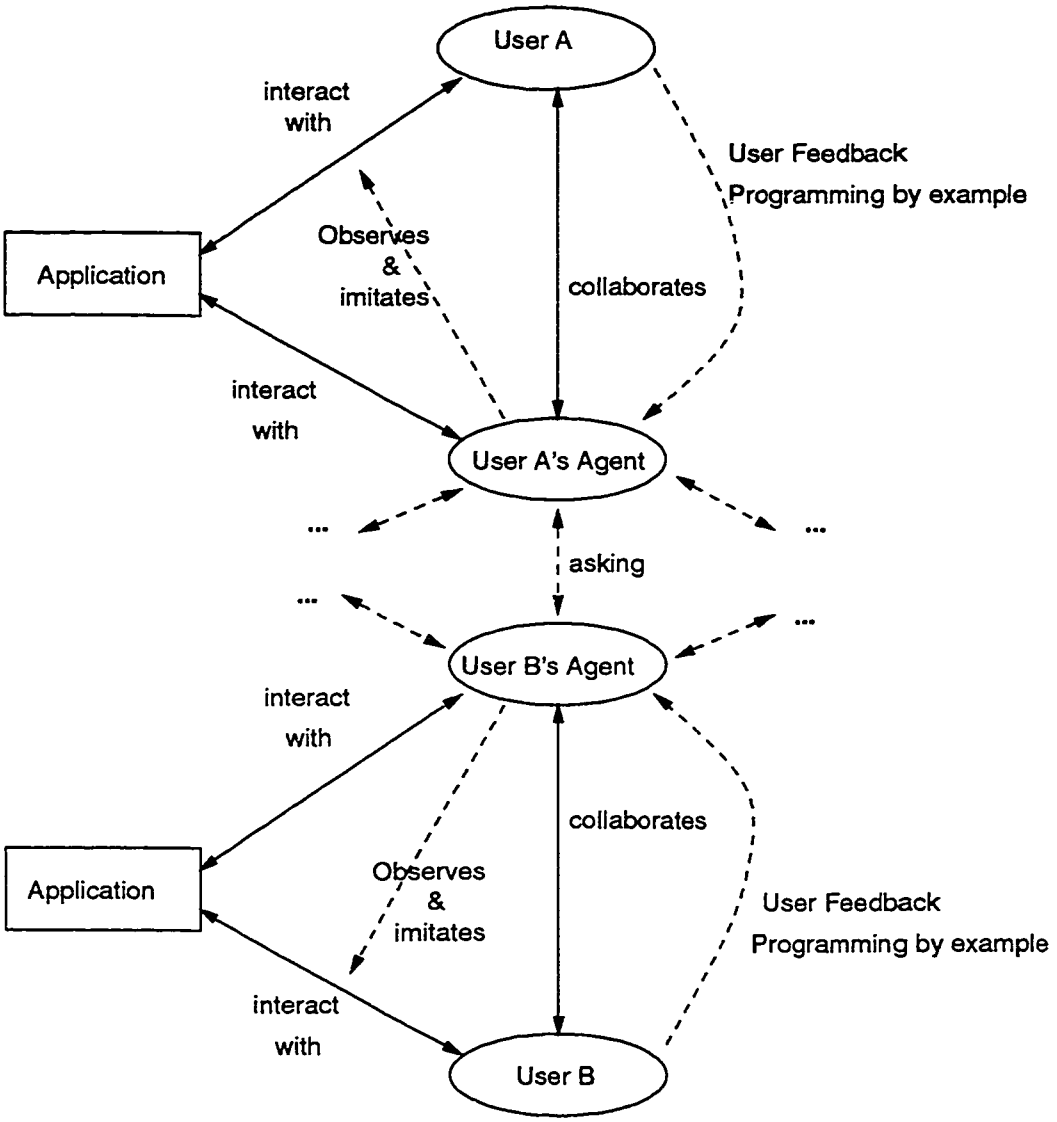

Figure 5: Agents Learn from the User and Peers (Adapted from Maes,1994)

2.3.1 Meeting Scheduler

Kozierok and Maes created a Meeting Scheduler Agent whose task is to assist a user managing and scheduling of meetings [KM93, Koz93, MK93, Mae94]. Initially, the agent starts out with minimal knowledge about its user's preference. By observing for a while what actions the user takes for meetings, it can make a prediction for future action when a meeting request arrives, such as acceptance, rejection, declination, or negotiation, etc. It remembers its user's preference (date, time, subject, participants) so as to provide assistance to the user in a very personalized way, suited to the unique habits of the user.

This research demonstrated, by employing machine learning approach the agent can eventually build up enough knowledge through observing the commands used by its user. It can then make scenario suggestions to him/her. Once the user has confidence in the proactive agent, the agent can be given permission to act on behalf of him/her.

2.3.2 Musag

Goldman and her colleagues built a system to help people searching information on the Internet by using semantic methods [GLR96]. This system consists of four agents — Musag, Usag, Sag, and Ag — which have different functionalities. Musag is the top agent whose task is to learn about concepts related to a given keyword. It builds a dictionary capturing the semantic message that exists when a user mentions the keyword. That is, Musag learns the implicit and common knowledge which the user has in mind when he submits a query by specifying of a keyword. The other three agents (Usag, Sag, and Ag) will use the dictionary built by Musag to search information for the user.

By applying learning approach in information searching, this system overcomes two main problems in syntactic-based search engines. When a keyword has multiple meanings, the system can retrieve relevant documents from large number of documents, which include many irrelevant documents. On the other hand, some relevant documents will be retrieved even if the given keyword is not included in the documents' topics. With the knowledge for a keyword given by a user, this system can help

people make a search more specific, efficient, and fast. Musag acquires the knowledge by itself without getting any feedback from users.

2.3.3 Open Sesame!

Open Sesame! is developed by Charles River Analytics (Cambridge, MA). It is the first learning agent which learns how users work with their Macintosh computers. The application sits on the desktop, observes a user's activities and learns repetitive usage patterns, and performs those repetitive tasks automatically with the user's permission [CSJ+96]. It even automates crucial maintenance tasks that the user can easily forget, like rebuilding the desktop. Open sesame! observes high-level user events, and analyses this information to identify time-based and event-based tasks for automation. A Time-based task is something that the user does at a particular time; a event-based task is something that the user does in relation to another task, such as opening the Alarm Clock desk accessory right after connecting to an expensive Internet service.

After Open Sesame! 1.0 was released in 1993, a lot of feedback went to Charles River Analytics from customers, researchers, and user groups [CSJ+96]. There were many suggestions that would help build an agent liked by human, such as these: the agent should not offer observations about topics the user has already rejected; the interruptions should happen less often; suggestions should be in the context of the user's current task. These lessons from Open Sesame! are also valuable for another learning agent's design. The mechanisms that it uses to learn, make suggestions, and confirm can be applied to more complex tasks as well.

Chapter 3

Case Study - Enhancements for FATMA

With the development of modern technology, telephony has played an increasingly important role in human communication and it has been used more and more widely in our social life. The function of the telephone has been expanded from its original purpose — people talking from a distance — to many new fields. For example, telephony can be used for telephone conference and information inquiring, such as weather forecast, personal banking service, stock exchange, etc. It helps people to obtain information quickly and easily.

The many uses of the telephone are supported by many complicated features. With more features being created and more information available through telephones, it will take much more time for people to handle their daily calls. However, people are very busy and have little time to learn to use new features. This is where a telephone-based agent can provide a useful service.

The telephone-based agent's primary function is to assist the subscriber in performing call processing functions, such as screening incoming calls or placing outgoing calls, which the owner now performs manually, or are performed for the subscriber by another person. By making use of such an agent to do what secretaries used to do, the subscriber can save time or can reduce costs by eliminating the need to hire another individual.

Both Wildfire and FATMA are examples of telephone-based agents. Wildfire has been in commercial market since 1994. FATMA is a Wildfire liked agent which is built by Roland Younes as a research [You97]. It is intended to overcome some drawbacks of Wildfire by adding proactive functionality to it, so as to improve the effectiveness of the interface in the conventional telephony. In this chapter, firstly I will introduce Wildfire and FATMA, and then I will discuss some enhancements that we could make to FATMA.

3.1 Wildfire

Wildfire, developed by Wildfire Communication Inc., is an Electronic Assistant that combines speech recognition, computer and telephone technology to provide call management for subscribers. It saves subscribers' time and increases their overall accessibility for important communication. It allows subscribers to interact with and navigate among different call management services using simple spoken commands. Amongst other things, Wildfire [Wil94]:

- screens, routes, and announces the subscriber's incoming calls;
- maintains the subscriber's contact list in order to let the subscriber "voicedial" his/her outgoing calls;
- schedules follow-up calls;
- reminds the subscriber to return calls, etc.;
- forwards calls to a subscriber specified location;
- gives the subscriber an advanced set of voice-mail features, such as leaving private, individual messages for callers; and
- can create groups enabling the subscriber to inquire for calls from a specific group of callers, or send certain messages only to specific groups.

Although the Wildfire agent is becoming more and more popular in serving today's high efficient communications, ("Wildfire is entering five new markets each month," said Rob Mechaley, president and CEO of Wildfire Communications.) the usage of

its features is limited by the willingness of the subscriber to learn them, and the willingness of the subscriber to use a particular feature. If subscribers couldn't get any value from the agent in the starting phase, instead, they feel too much work need to be done before taking a simple action, and they are not likely to want to learn a complicated feature set offered by an agent.

The problem, then, is to design agents so that they are simple to use, but at the same time, can make people's daily phone communications easier.

3.2 FATMA

Younes studied adding proactive functionality to telephone-based agents.

> "A Proactive agent, not only reacts or responds to the user's needs, but also takes initiatives and acts without the interference of the user when the agent needs to do so" (quote from Younes [You97]).

He also pointed out that there are three characteristics that an agent must have in order to be proactive: *learning, trustworthy*, and *helpful.*

- *Learning* — by applying machine learning approach, the agent can start out by simply observing the commands used by subscribers, and eventually build up enough knowledge and then apply it to performing similar tasks.
- *Trustworthy* — the agent should make the subscriber feel comfortable to assign tasks to it. The agent learns its subscriber's behaviour and preference not only from its observations but also from the feedback obtained from the subscriber.
- *Helpful* — after getting permission from the subscriber, the agent can handle particular tasks on behalf of the subscriber.

A proactive telephone-based agent — FATMA (Futuristic Automated Telecommunication Management Agent) was built by Younes. He aimed to insure that the agent offers suggestions event-based and time-based (focusing on *learning* and *helpful* characteristics of proactive agents). Event-based means the agent should make suggestions which are relevant to the context of the user's current task. Time-based means the suggestions should be made at appropriate time. For example, at 7:00

in the morning, the agent should not make any suggestions to its subscriber about what action should be done at 2:00 in the afternoon. Younes created a feature set for FATMA which is similar to what Wildfire provides. Since CALL SCREENING, which provides a filter for the incoming calls, is an important service that can be offered and supported by the telephone-based agent to the subscriber, his project focuses on this issue. In this phase, he created a testbed, built a preliminary rule set for the CALL SCREENING feature, and used the testbed to validate the functionality provided by FATMA.

Although the CALL SCREENING rule set implemented by Younes allows an agent to make suggestions to its subscriber at appropriate times, it does not permit the specification of differences between subscribers, nor does it adjust the agent *behaviour* according to the *feedback* provided by the subscriber when the agent makes a suggestion.

- *Feedback* — in form of "yes" or "no" indicates the subscriber's acceptance or rejection of a suggestion offered by the agent.
- *Behaviour* — refers to the frequency with which the agent makes suggestions.

My contribution for FATMA has been to construct an agent that can offer suggestions unobtrusively by having the agent offers suggestions with "appropriate frequency", which means the agent offers suggestion neither frequently nor infrequently. The aim of my project is focus on the *trustworthy* characteristic of proactive agents, that is, to improve the agent's performance and let the user feel comfortable using it.

In the following sections, the testbed of FATMA and CALL SCREENING of which I worked on will be introduced first, and the enhancements made for FATMA on CALL SCREENING feature will be discussed later on.

3.2.1 Testbed of FATMA

The testbed was built using three languages that were integrated together: CLIPS was used for creating the rule set, Tcl/Tk was used for the user interface, and C was used for event handling. CLIPS is an expert system tool that provides a knowledge base that contains all the rules, and working memory, in which all the information

that the agent uses to build up its knowledge is stored. The testbed operation is illustrated in Figure 6. Firstly, the agent makes observations for user's behaviour and records it in working memory; when the data observed meets specific criteria the rules will fire and generate facts (i.e., pending suggestions); finally, facts will trigger suggestions at appropriate time and the agent makes them to the user in time. It facilitates the construction and testing of rule sets.

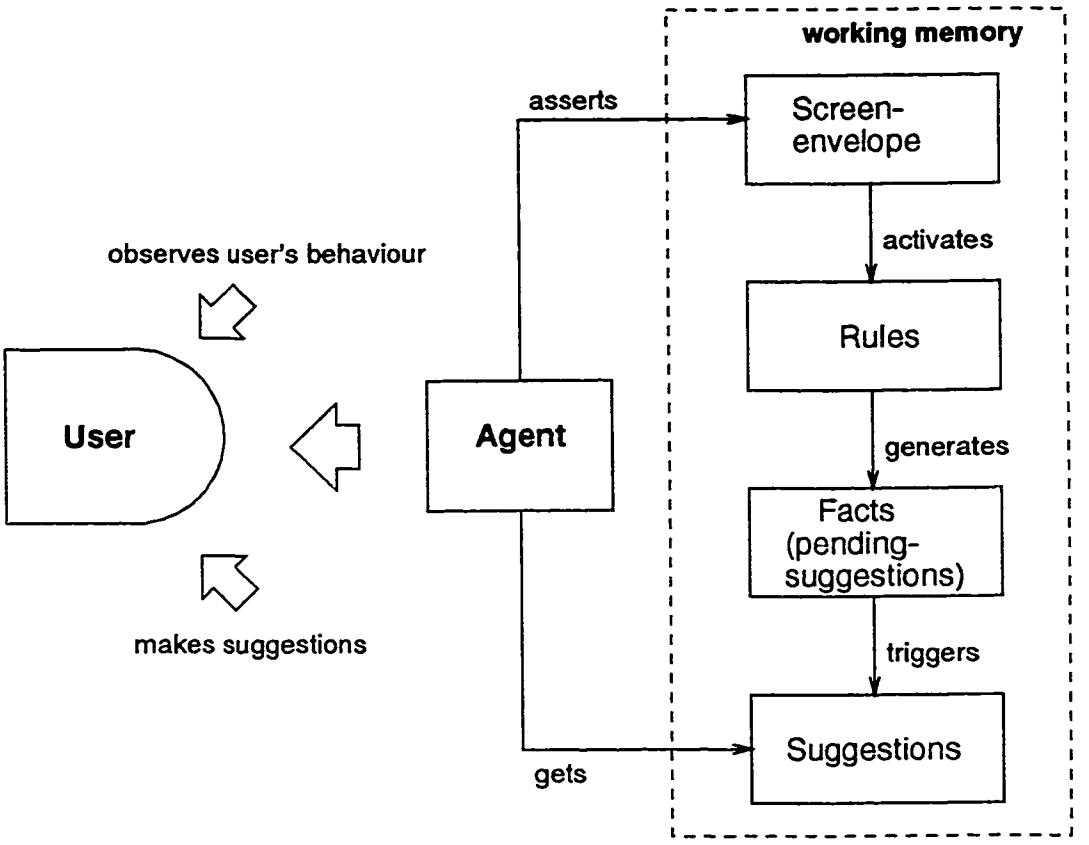

Figure 6: FATMA Testbed Operation

3.2.2 CALL SCREENING Feature for Enhanced FATMA

Younes created a CALL SCREENING feature for FATMA [You97]. I modified the categories of caller groups and call dispositions in order to present my work more clearly. The following are the CALL SCREENING features used in my project.

All callers are categorised in four groups:

- KNOWN CALLER — a caller whose phone number is stored in the subscriber's personal directory.
- UNKNOWN CALLER — a caller whose phone number is not stored in the subscriber's personal directory.
- ANONYMOUS CALLER — a caller whose phone number can not be displayed for the subscriber.
- INDIVIDUAL CALLER — a caller whose name or phone number is told to the agent by the subscriber.

When receiving an incoming call, subscribers can choose to dispose of the call in one of five ways:

- DIRECTLY TO USER — subscribers may select to pass the call through.
- VOICE MAIL — subscribers may allow the caller to leave voice message.
- SEND SPECIAL MESSAGE — subscribers may choose to play a special message to the caller.
- ASK USER FIRST — subscribers may choose to have the agent check the incoming call with them first.
- BLOCK — subscribers may select to block the specific call, i.e., the caller can't access the subscriber.

CALL SCREENING provides an ability to screen the subscriber's incoming calls, and set dispositions for each category of callers. That is, the subscriber can give any screening command based on the pair of caller and disposition.

3.3 My Contribution to FATMA Enhancement

I used the testbed created by Younes to make the following enhancements to a telephone-based agent:

1. The agent should categorize subscribers based on their potential willingness to accept new ideas in order to permit the agent to treat subscribers differently according to their "characteristics" as estimated by the agent.

2. The agent should categorize the Call Screening suggestions that might be offered by the agent based on the disposition selected and caller type the disposition is to be applied to. Thus, we can treat different categories of suggestions in different ways. For suggestions that are more likely be offered, we let the agent obtain less *evidence* before making them. By contrast, for suggestions concerning some pair of disposition and caller type that are rarely used by the subscriber, the agent will wait longer to observe and offer them to the subscriber only after it has obtained enough *evidence* to prove that the subscriber repeats this action often.

   - *Evidence* — each call screening command issued by the subscriber and recorded as a fact in working memory is referred to as a single piece of evidence.

   For example, (known callers, voice mail, 7 a.m. — 5 p.m.) is a piece of evidence.

3. I created the "lifecycle concept" for pending suggestions. The lifecycle of a pending suggestion consists of two periods: collecting evidence to identify a pattern, and confirming the pattern still be used. The lifecycle of different pending suggestions is different. All this information is stored in a table in working memory. By checking this table, the agent can decide whether to offer the pending suggestion to the subscriber or not. If a pattern is identified by the agent, and it is still repeated by the subscriber during the confirmation period, the agent will offer the pending suggestion to the subscriber. Otherwise, the pending suggestion is deleted by the agent. By using a table-driven approach instead of hand-coding information in rule sets, which Roland Younes did in the first version of the rule set, the information can be easily changed in the table without modifying rule sets.

4. The agent uses *feedback* obtained from the subscriber to adjust the its *behaviour* in order to avoid annoying the subscriber. It can automate a task on behalf of the subscriber, change the frequency with which it make suggestions to the

subscriber, or stop offering a particular suggestion that the subscriber does not want to receive.

Sections 3.3.1 through 3.3.4 discuss these enhancements in details.

3.3.1 Categorizing Subscribers

The agent classifies the subscriber belongs into one of three groups: liberal, moderate, and conservative.

- Liberal — this type of subscriber is open to suggestions and ideas, and is likely to accept suggestions from the agent.
- Moderate — this type of subscriber needs longer to accept suggestions than the liberal subscriber.
- Conservative — this type of subscriber would like to receive suggestions less often, and needs even more time before trusting the agent.

Basing on the category of the subscriber, agents requires different amount of evidence before offering suggestions. In the beginning, the subscriber's type is assumed to be moderate by default. If a subscriber accepts suggestions offered by the agent quickly, the agent will adjust the subscriber's type to liberal. In contrast, a subscriber who usually rejects suggestion will be put in the conservative category.

3.3.2 Categorizing Call Screening Suggestions

As shown in Table 1, we rated each disposition for a particular type of caller. A rating of 4 indicates that the disposition is most likely be used for that type of caller. Thus, we assumed that for this kind of suggestion, the agent needs to obtain less evidence before making the suggestion to the subscriber than required for suggestions with a lower rating. For example:

- Suggestion 1 — automate setting ASK USER FIRST disposition for UNKNOWN callers' calls.
- Suggestion 2 — automate setting ASK USER FIRST disposition for calls from KNOWN callers.

	Directly To User	Voice Mail	Send Special Message	Ask User First	Block
Individual	3	2	3	1	3
Anonymous	1	3	1	3	4
Known	4	4	3	1	1
Unknown	2	4	2	4	2

Table 1: Rating of the Disposition- Caller Pair

For these suggestions, the agent needs different amount of evidence before making each suggestion to the subscriber. Since the disposition ASK USER FIRST is most likely be used for UNKNOWN calls, we assume that the agent needs to obtain three pieces of evidence before making suggestion 1. That is, the agent has to have observed that the subscriber had repeated this action at least for three times before it makes this suggestion. However, since the ASK USER FIRST disposition is rarely used for KNOWN callers, the agent needs to have more evidence, let's say, six pieces of evidence, before making suggestion 2. The evidence proves that the subscriber used to have this setting instead of just doing it occasionally. By collecting evidence, the agent builds up its knowledge and confidence for offering a suggestion.

3.3.3 The Lifecycle of Pending Suggestions

The lifecycle of pending suggestions consists of two phases (as shown in Figure 7). In Phase 1, the agent identifies a *pattern* by examining pieces of evidence in working memory. In Phase 2, the agent confirms that the *pattern* is still being carried out by the subscriber, again by examining the evidence in working memory. The agent needs different amounts of evidence to identify different patterns, and the time during which the confirmation should be completed is also different.

- *Pattern* — an action repeated at the same time in a given set of days in a week. The magnitude of the pattern is counted by days.

For example, putting all calls to voice mail from 10 a.m. to 5 p.m. on Monday, Tuesday, Wednesday, Thursday and Friday is a pattern. The "size" of this pattern is 5 (days).

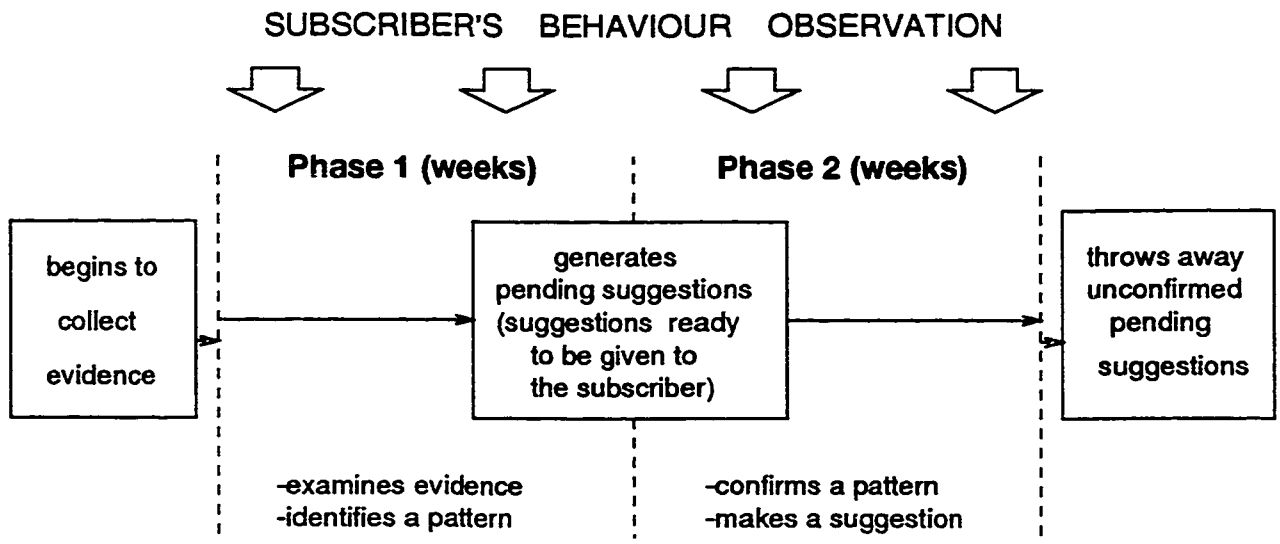

Figure 7: The Lifecycle of Pending Suggestions

Phase 1 and Phase 2 are counted as week(s), since the agent only recognizes a weekly pattern. During Phase 2, the pattern must be confirmed by the agent for a certain number of weeks. For example, there might be three weeks in Phase 2. If the agent can obtain evidence to support that the pattern is still being carried out by the subscriber in any week in this period (maybe in the first week, the second week, or the third week), the agent will generate a suggestion to the subscriber. Otherwise, if the agent could not obtain enough evidence for confirmation in this phase, the pattern is discarded. Phase 2 for different pending suggestions varies according to Phase 1. For patterns that are easily recognized by the agent, the confirmation phase is short. However, for patterns that need more evidence to identify, Phase 2 is longer. In this project, we defined that for the pending suggestion whose Phase 1 is 1 week, Phase 2 is 1 week also. If Phase 1 is 2 or 3 weeks, Phase 2 is 2 weeks. If Phase 1 is more than 3 weeks, Phase 2 is 3 weeks.

Table 2 shows the lifecycle of pending suggestions that is used by the agent to determine how many pieces of evidence it requires before offering a suggestion. There are six attributes in the table:

- Subscriber's type — liberal, moderate, or conservative.

subscriber's type	suggestion rating	days within a week	number of consecutive complete weeks	number of days in the following week	lifetime of pending suggestions
Liberal	3 or 4	5	1	1	2
Moderate	3 or 4	5	1	3	2
Conservative	3 or 4	5	2	1	2
Liberal	1 or 2	5	2	1	4
Moderate	1 or 2	5	2	3	4
Conservative	1 or 2	5	3	1	5
Liberal	3 or 4	4	1	2	2
Moderate	3 or 4	4	1	4	2
Conservative	3 or 4	4	2	3	2
Liberal	1 or 2	4	2	2	4
Moderate	1 or 2	4	2	4	4
Conservative	1 or 2	4	3	3	5
Liberal	3 or 4	3	2	1	4
Moderate	3 or 4	3	2	3	4
Conservative	3 or 4	3	3	1	5
Liberal	1 or 2	3	3	1	5
Moderate	1 or 2	3	3	3	5
Conservative	1 or 2	3	4	1	7
Liberal	3 or 4	2	2	2	4
Moderate	3 or 4	2	3	1	5
Conservative	3 or 4	2	4	1	7
Liberal	1 or 2	2	3	2	5
Moderate	1 or 2	2	4	1	7
Conservative	1 or 2	2	5	1	8
Liberal	3 or 4	1	3	1	5
Moderate	3 or 4	1	4	1	7
Conservative	3 or 4	1	5	1	8
Liberal	1 or 2	1	4	1	7
Moderate	1 or 2	1	5	1	8
Conservative	1 or 2	1	6	1	9

Table 2: Lifecycle of Suggestions

- Suggestion rating — indicates the likelihood of the occurrence of suggestions (please refer to Table 1). Thus far we have treated suggestions with rating of 3 and 4 the same, suggestions with rating of 1 and 2 the same as well, to simplify the implementation.

- Days within a week — number of days in a week in which an action is observed at the same time. This column also indicates how many pieces of evidence the agent should collect in a week.

- Number of consecutive complete weeks — indicates how many consecutive weeks in which a pattern must be repeated by the subscriber. This column refers to the Phase 1 in the pending suggestion lifecycle.

- Number of days in the following week — indicates how many pieces of evidence the agent should have collected for confirming the pattern. Then the agent can generate an appropriate pending suggestion to the subscriber.

- Lifetime of pending suggestions — indicates the number of weeks a pending suggestion can be stored in working memory. The number in this column minus the number in column "number of consecutive complete weeks" refers to Phase 2 in the pending suggestion lifecycle, since the lifetime of pending suggestions consists of Phase 1 and Phase 2.

For example, the first three tuples indicate that, although the suggestion rating and the number of times the same command issued by the subscriber in a week are the same, the same pending suggestion is generated at different times for different types of subscribers.

Let us say that the agent recognizes a pattern where the subscriber sends calls from unknown callers to voice mail from 9 a.m. to 5 p.m. on Monday, Tuesday, Wednesday, Thursday and Friday in the first week when the pending suggestion is generated. For a liberal subscriber, if the agent observes that the subscriber takes the same action in the following Monday morning, it will make a suggestion to automate this action. For a moderate subscriber, if the agent observes this pattern in the first week, the action would also have to be repeated by the subscriber on the following Monday and Tuesday. If the same command is issued by the subscriber on Wednesday morning,

the agent will make the suggestion to the subscriber. For a conservative subscriber, the same pattern should be repeated for two consecutive weeks. When the subscriber asks the agent to do the same task on Monday morning of the third week, the agent will give the suggestion to the subscriber. Consequently, a liberal subscriber would have most calls screened, but a conservative subscriber would have an almost inactive agent. The lifetime of this pending suggestion will be 2 weeks since the first Monday for the liberal and moderate subscriber, and 4 weeks for the conservative subscriber.

3.3.4 Subscriber's Feedback

Whether the agent can offer the right suggestion unobtrusively at the right time affects the subscriber's willingness to use the agent. Feedback is an important method by which the agent can get knowledge about the subscriber's preference.

The feedback from different suggestions is treated differently by the agent. In this project, there are three types of suggestions that an agent can make to the subscriber:

- Type 1 — The agent suggests to automating the time at which calls from a particular type of caller will be treated in a particular way (i.e., with a particular disposition).
- Type 2 — The agent offers to make suggestions more or less often.
- Type 3 — The agent offers to stop making suggestions which refer to screening calls from a particular type of caller using a particular type of disposition in a specified period of time.

The details of how the agent adjusts its behaviour basing on the feedback for these different suggestions will be described in Section 3.4.

3.4 Implementation Issues for the Enhanced Agent

3.4.1 Key Facts Stored in Working Memory

Before discussing the details of how the agent adjusts its behaviour, I first introduce some facts which are stored in working memory and used by the agent to determine its behaviour.

Fact (screen-envelope) — Contains the screening commands given by the subscriber and the time when the command is issued.

Fact (check-similar-days) — Records number of days in a given week in which a *behaviour x* is observed by the agent.

Behaviour x can be any valid call screening command the subscriber may issue to the agent.

Fact (check-frequency) — Records number of weeks in which the behaviour x is repeated by the subscriber on the same day, and at the same time of day.

Fact (pending-suggestion) — Indicates the agent already obtained amount of evidence to recognize a pattern and terminated phase 1 of lifecycle of suggestions.

Fact (make-suggestion) — Indicates that a suggestion is ready to be made to the subscriber. So the agent checks this fact and makes the suggestion at *an appropriate time.*

An appropriate time refers to the agent's making suggestion related with the subscriber's current action.

Fact (feedback) — Stores the last response from the subscriber for a suggestion offered by the agent.

Fact (change-level) — Records whether the subscriber has accepted or rejected two consecutive suggestions.

Fact (set-yes-flag) — Indicates if the subscriber has accepted a suggestion since the subscriber's type was last changed.

Fact (type-modified) — The agent offers to make suggestions more or less often when the fact **(type-modified)** is exist. If the subscriber does not agree with this suggestion, the fact **(type-modified nil)** is used to record it.

Fact (confirm-not-offer-suggestion) — Indicates that the subscriber does not want to receive a suggestion any more, even though the agent might observe the same behaviour again.

Fact (not-confirm-stop-offer-suggestion) — Indicates that the subscriber does not stop the agent making the same suggestion again although he/she has rejected the suggestion many times before.

3.4.2 Feedback from Offering Type 1 Suggestions

As described in Section 3.3.4, a Type 1 Suggestion is offered by the agent when it wants to automate the time at which calls from a particular type of caller (i.e., KNOWN, UNKNOWN, or ANONYMOUS) will be treated with a particular disposition (i.e., DIRECTLY TO USER,VOICE MAIL, SEND SPECIAL MESSAGE, ASK USER FIRST, or BLOCK).

If the subscriber accepts a suggestion offered by the agent, the agent has permission to take care of this particular action for the subscriber. If the subscriber rejects the suggestion, the agent will make this suggestion again when it observes the same behaviour carried out by the subscriber. The feedback obtained when a suggestion is offered not only determines whether the agent automates some task on behalf of the subscriber, but it also determines whether the agent should adjust the frequency with which suggestions are made to the subscriber.

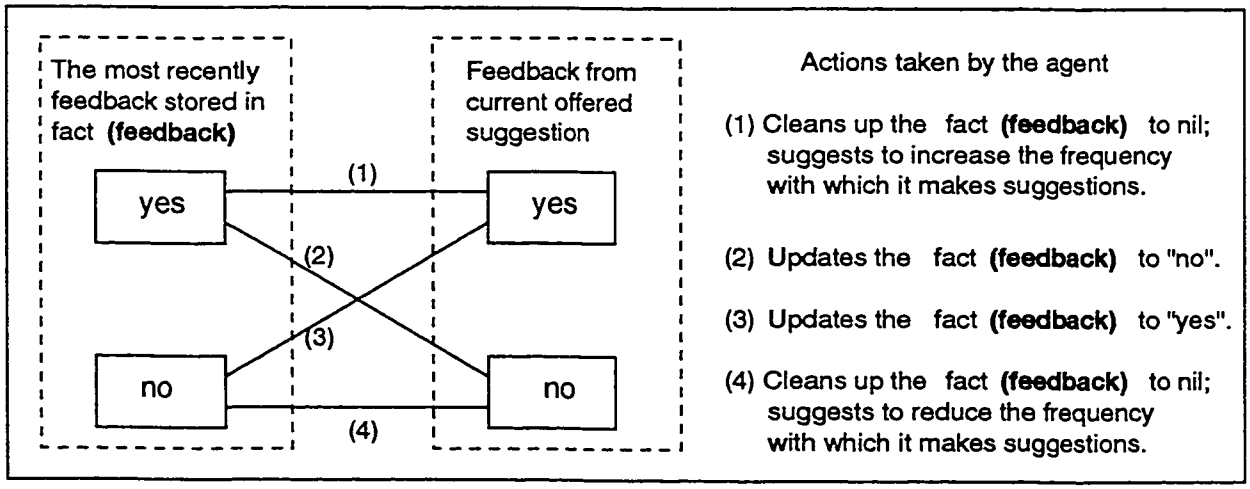

Figure 8: Use Feedback to Adjust the Frequency with Which Suggestions are Made

A fact **(feedback)** is stored in working memory for holding feedback from the

suggestion made most recently. When the subscriber provides a feedback for a new suggestion, the comparison for these two feedbacks is made by the agent. As shown in Figure 8, after obtaining two consecutive positive or negative feedbacks, the agent will suggest to change the frequency with which the suggestion will be made. Otherwise, the agent update the fact **(feedback)**.

Feedback obtained when a suggestion is made for the first time

When the feedback from offering a suggestion that is made for the first time is examined, two different actions can be taken by the agent, as indicated in Figure 9.

- Branch 1 — When the current suggestion is the first suggestion offered by the agent to the subscriber since the subscriber was of the current type, or the agent has just offered to make suggestions more or less often, (indicated as there is no fact **(feedback)** stored in working memory), or the feedback from the current and previous suggestions are not the same, the agent will store the feedback received from the subscriber for the current suggestion by asserting a fact **(feedback)** in working memory. The agent will decide if it is an appropriate time to offer other remaining suggestions by observing the fact **(make-suggestion)** in working memory.

- Branch 2 — By comparing the feedback obtained from offering the current suggestion and the previous suggestion, a decision is made as to whether to adjust the subscriber's type. If both the current and previous suggestions are accepted, the agent can propose to make suggestions more often (internally, it moves the subscriber up one level). If both the current and previous suggestions are rejected, the agent proposes to make suggestions less often (internally, it drops the subscriber one level).

Feedback obtained when the same suggestion is made for the second time

As shown in Figure 10, the feedback obtained when the same suggestion is made for the second time can be used by the agent to decide whether to offer to stop making a particular suggestion. The following are two actions the agent can take:

- Branch 1 — If the subscriber accepts the suggestion, the agent treats this case as if it were the first time that this particular suggestion is made to the subscriber.

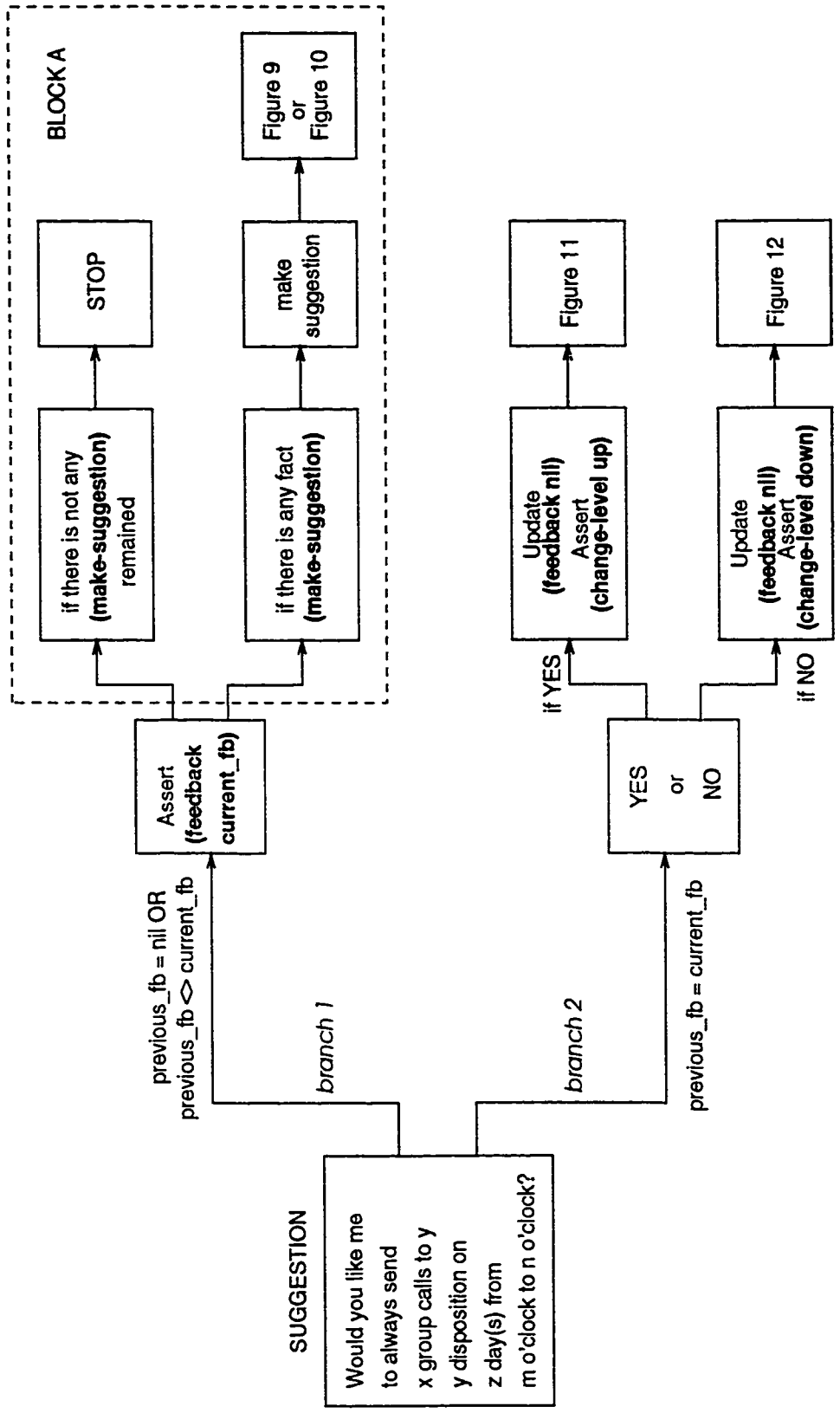

Figure 9: Feedback Obtained When a Suggestion is Made for the First Time

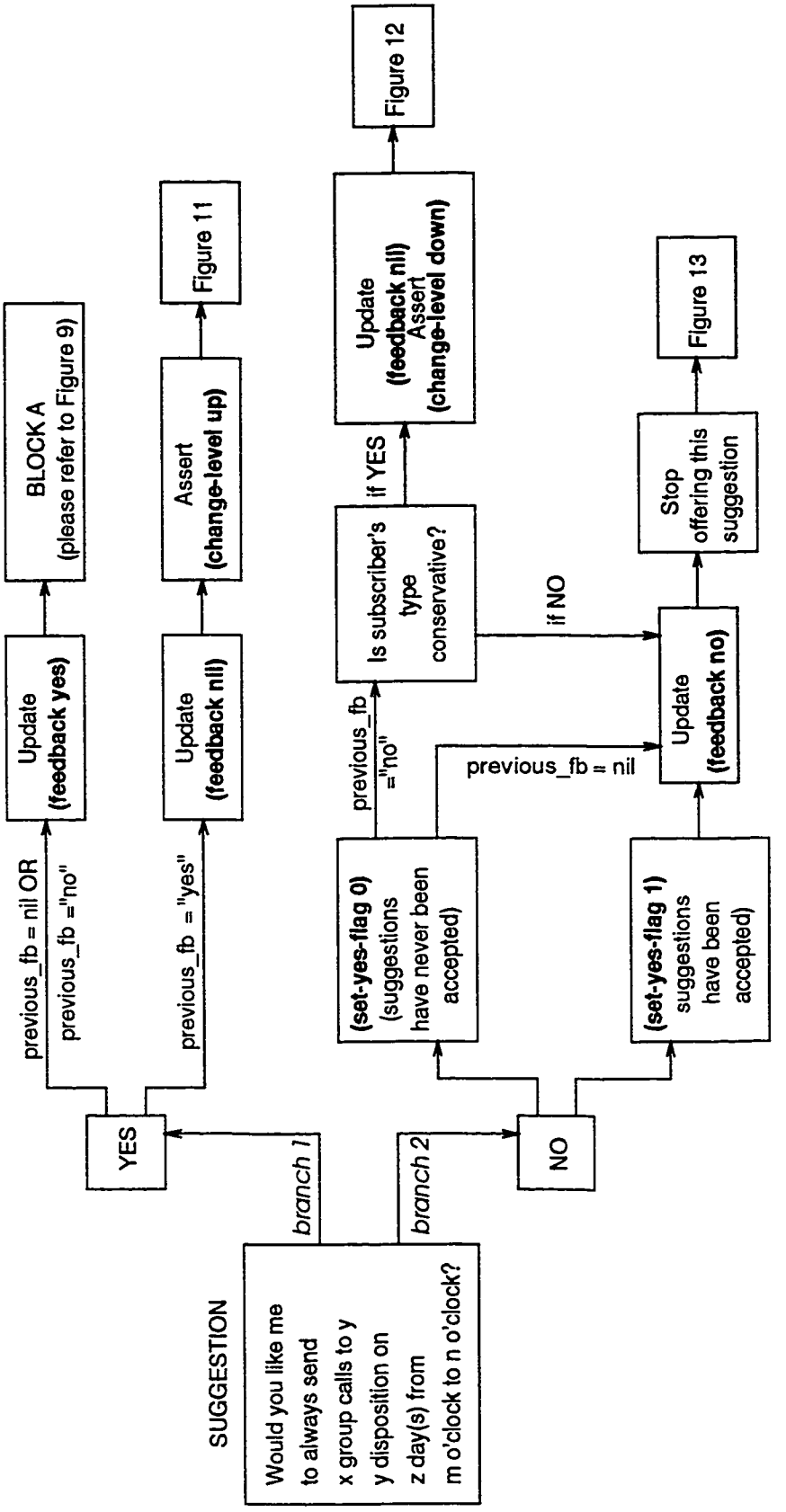

Figure 10: Feedback of the Suggestion Made the Second Time

The agent will automate the time at which calls from a particular type of caller will be handled with a particular disposition. Using the feedback obtained with the previous suggestion, the agent can take two different actions as described in Figure 9.

- Branch 2 — When the subscriber rejects the same suggestion that is made the second time, there will be two different scenarios based on whether or not the subscriber has accepted a suggestion since the subscriber's type was last changed.

  Scenario 1 — If the subscriber has accepted any suggestions offered by the agent, (i.e. the fact **(set-yes-flag 1)** exists), it means that the subscriber would like to accept suggestions except this particular one since he/she rejected this suggestion twice. So, the agent will propose to stop offering this suggestion.

  Scenario 2 — If the subscriber has not yet accepted any suggestion offered by the agent (indicated as the fact **(set-yes-flag 0)** is exist), and there is a fact **(feedback nil)** stored in working memory, which means the agent just cleaned it up for proposing to adjust the subscriber's type, the agent will suggest to stop offering this particular suggestion. If the feedback for the previous suggestions is "no", it means that the subscriber has not come to trust the agent to work on his/her behalf. If the subscriber type is conservative (i.e. the lowest level in the subscriber's type), the agent would suggest to stop offering this particular suggestion as well. If the subscriber type is not conservative, the agent then suggests to make the suggestions less often (i.e. internally, it drops the subscriber's type one level).

3.4.3 Feedback from offering Type 2 Suggestions

For Type 2 Suggestions, the agent offers to make suggestions more or less often. Figure 11 shows the feedback from the subscriber when the agent offers to make suggestions more often. Two possible actions are described as follows:

- Branch 1 — If the subscriber accepts the suggestion that he/she would receive suggestions more often (i.e. internally, change one level up), the agent will

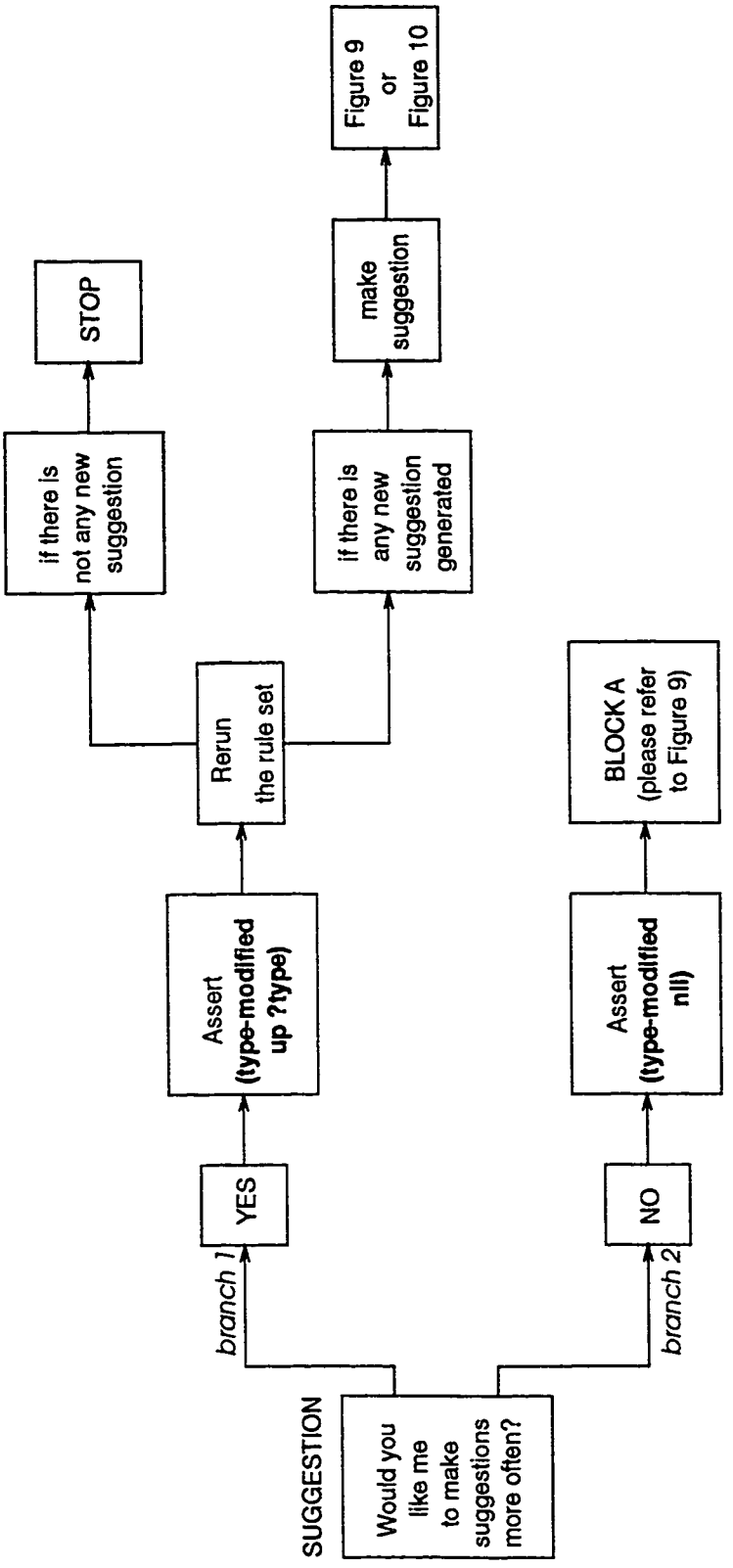

Figure 11: Feedback Obtained When the Agent Suggests to Make Suggestion More Often

update this change in working memory. When there is any new pending suggestion generated the agent will keep making suggestions. If there are no more suggestions that are appropriate for the current operation carried out by the subscriber, the agent stops offering suggestions.

- Branch 2 — If the subscriber rejects that the agent would makes suggestions more often, the agent simply observes if there are any remaining pending suggestions in current working memory, and takes the appropriate action on them, as shown in Figure 9.

Figure 12 shows the feedback from the subscriber when the agent offers to make suggestions less often.

- Branch 1 — If the subscriber accepts this suggestion (i.e. internally, change the subscriber one level down), the agent will update the subscriber's type and stop offering suggestions. If there is a suggestion which has been rejected twice by the subscriber, the agent will remove this suggestion as it has never been offered to the subscriber.

- Branch 2 — If the subscriber rejects the suggestion that the agent would makes suggestions less often, the agent will observe if there is a suggestion already rejected twice by the subscriber. If no such suggestion exists, the agent will examine if there is any remaining suggestion that is related to the current action taken by the subscriber, and then it will take the appropriate action on them, as shown in Figure 9. If there is a suggestion which has been rejected by the subscriber twice, the agent will suggest to stop offering this particular suggestion.

3.4.4 Feedback from offering Type 3 Suggestions

For this type of suggestion, the agent offers to stop making suggestions with reference to screening calls from a particular type of caller using a particular type of disposition in a specified period of time.

As shown in Figure 13, two actions can be taken by the agent after it obtains the feedback from the subscriber for the suggestion.

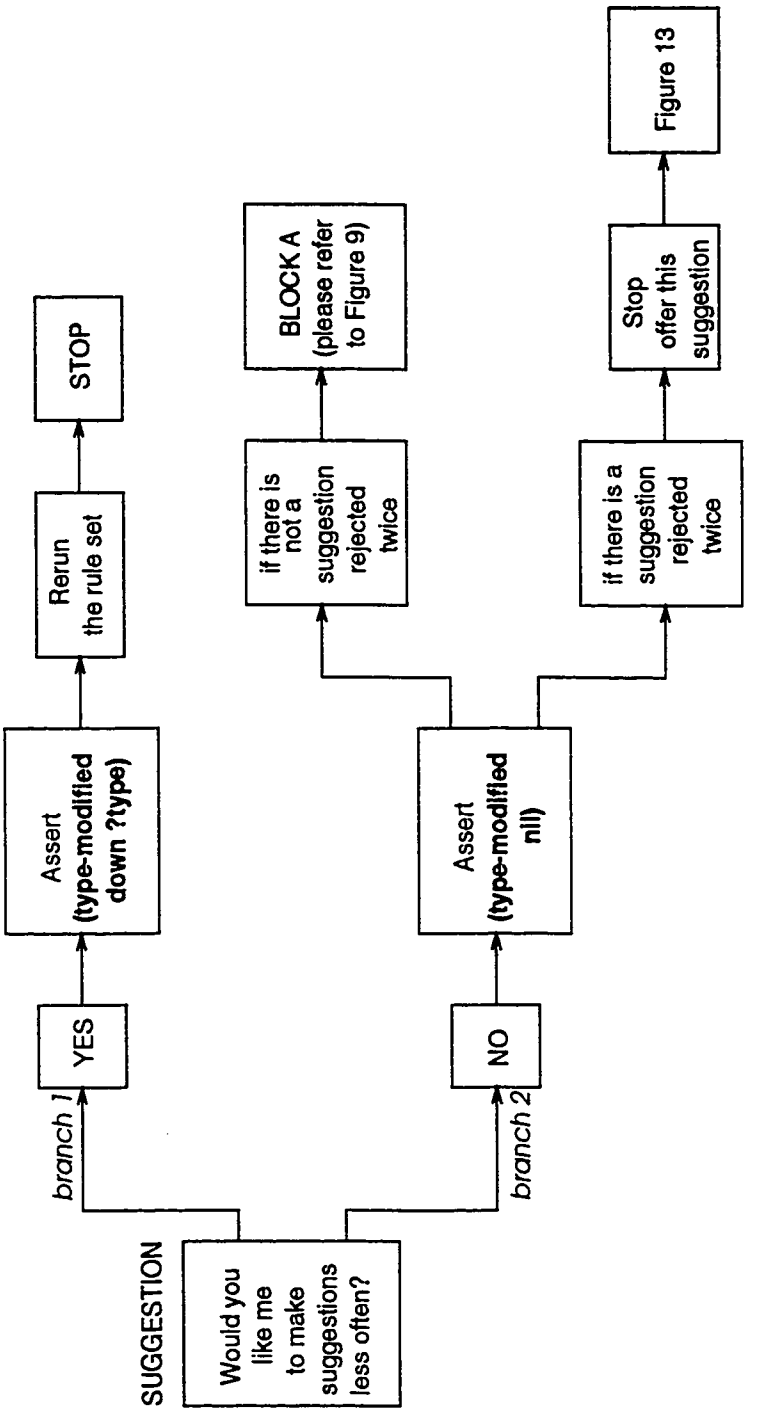

Figure 12: Feedback Obtained When the Agent Suggests to Make Suggestion Less Often

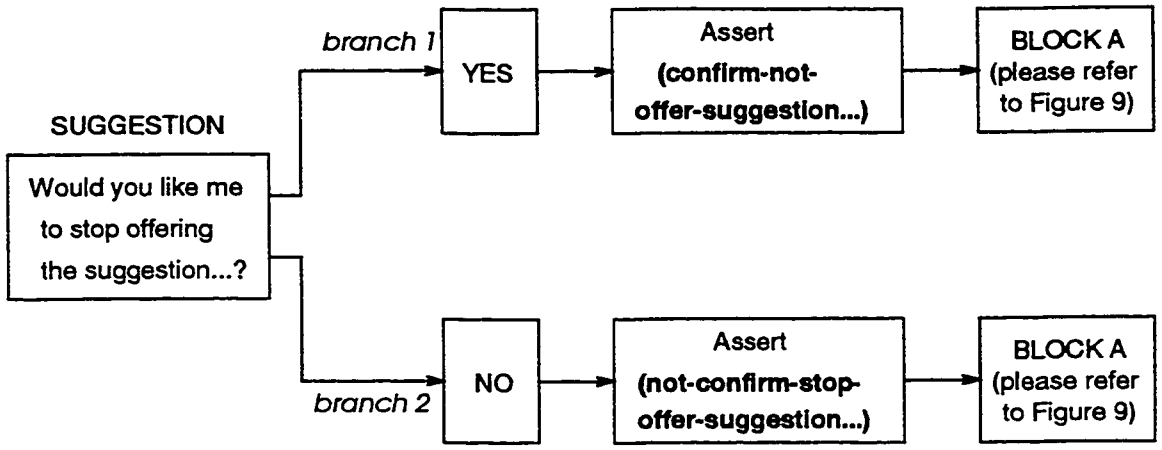

Figure 13: Feedback Obtained When the Agent Suggests to Stop Offering a Suggestion

- If the subscriber accepts the suggestion, it means that the subscriber does not want to receive the suggestion any more, even though the agent might observe the same behaviour again. So the agent will not make this particular suggestion to the subscriber later on.
- If the subscriber rejects the agent's suggestion, it indicates that the subscriber doesn't stop the agent making the same suggestion, although he/she has rejected the suggestion many times before. So the agent will make this suggestion again when it observes the same pattern carried out by the subscriber.

After making any one of the decisions described as above, the agent will decide if there is any suggestion it should make to the subscriber at this time, as shown in Figure 9.

3.5 Summary

The main goal for making enhancements for telephone-based agents is to avoid the agent annoying subscribers and improve the usability of the agent. The enhancements can be concluded as follows:

- categorizing users in order to allow agents to treat users differently;
- presenting users' satisfaction in order to avoid the agent annoying users;

- using machine-learning approach to improve agents performance.

In the next chapter, two scenarios are created for demonstrating how the enhanced telephone-based agent helps its subscriber to handle his/her daily phone calls, and how the agent adjusts its behaviour to suit to the unique habits of the user.

From this special case, we can also get some general ideas about how to build a human-like agent and the lifecycle of the agents' development. These issues will be discussed in Chapter 5.

Chapter 4

The User Interface of the Enhanced Agent

Basing on Roland's FATMA, an enhanced agent was built. In this chapter, a demonstration of how the enhance agent works will be given by using two predefined scenarios.

Figure 14 shows several features that the FATMA can provide to its subscriber [You97]:

- Call — helping the subscriber to perform outgoing calls, and maintaining the personal directory for the subscriber.
- Private Call — allowing the subscriber to perform private outgoing calls.
- Call Screen — screening subscriber's incoming calls and allowing the subscriber to set desired dispositions for different callers.
- Call Forward — forwarding calls to a specific phone number, i.e., home phone number, working place phone number, cellular phone number, or a manually entered phone number.

In this project, we focus on the CALL SCREENING feature. As mentioned in Section 3.2.2, all callers are categorized in four groups: KNOWN, UNKNOWN, ANONYMOUS, and INDIVIDUAL CALLER. Moreover, there are five possible dispositions: DIRECTLY TO USER, VOICE MAIL, SEND SPECIAL MESSAGE, ASK USER FIRST, and BLOCK for each category of callers. Figure 15 presents the structure of the CALL SCREENING feature in screen-based interface.

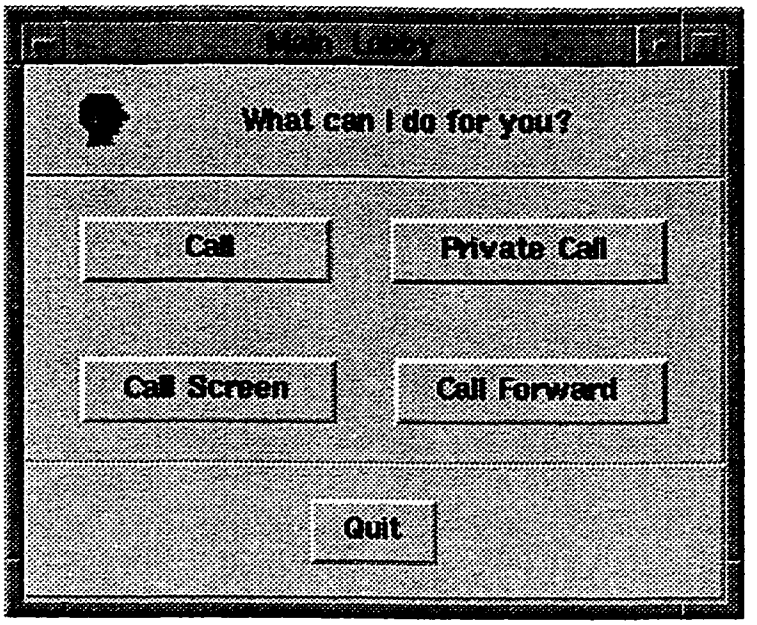

Figure 14: User Interface of an Agent

4.1 Demonstration of the Project

There are two scenarios created for the demonstration. In the first scenario, as shown in Figure 16, the subscriber uses the CALL SCREENING feature on Monday, Tuesday and Wednesday. (Assume, for example, that the subscriber leaves for another city every Thursday to Sunday.) During the three days, the subscriber sends all callers' calls to voice mail before he leaves to work at 7 a.m., puts through known callers' calls when he is back home at 5 p.m., puts through all callers' calls since 7 p.m., and sends all calls to voice mail at bed time.

In the second scenario, as shown in Figure 17, the subscriber uses the CALL SCREENING feature in five weekdays in the first week. Assume the subscriber leaves for vacation for the following two weeks. The schedule with which the subscriber sends the screening commands in the five weekdays are the same as that in first scenario.

Figure 18 shows the DEMO WINDOW.

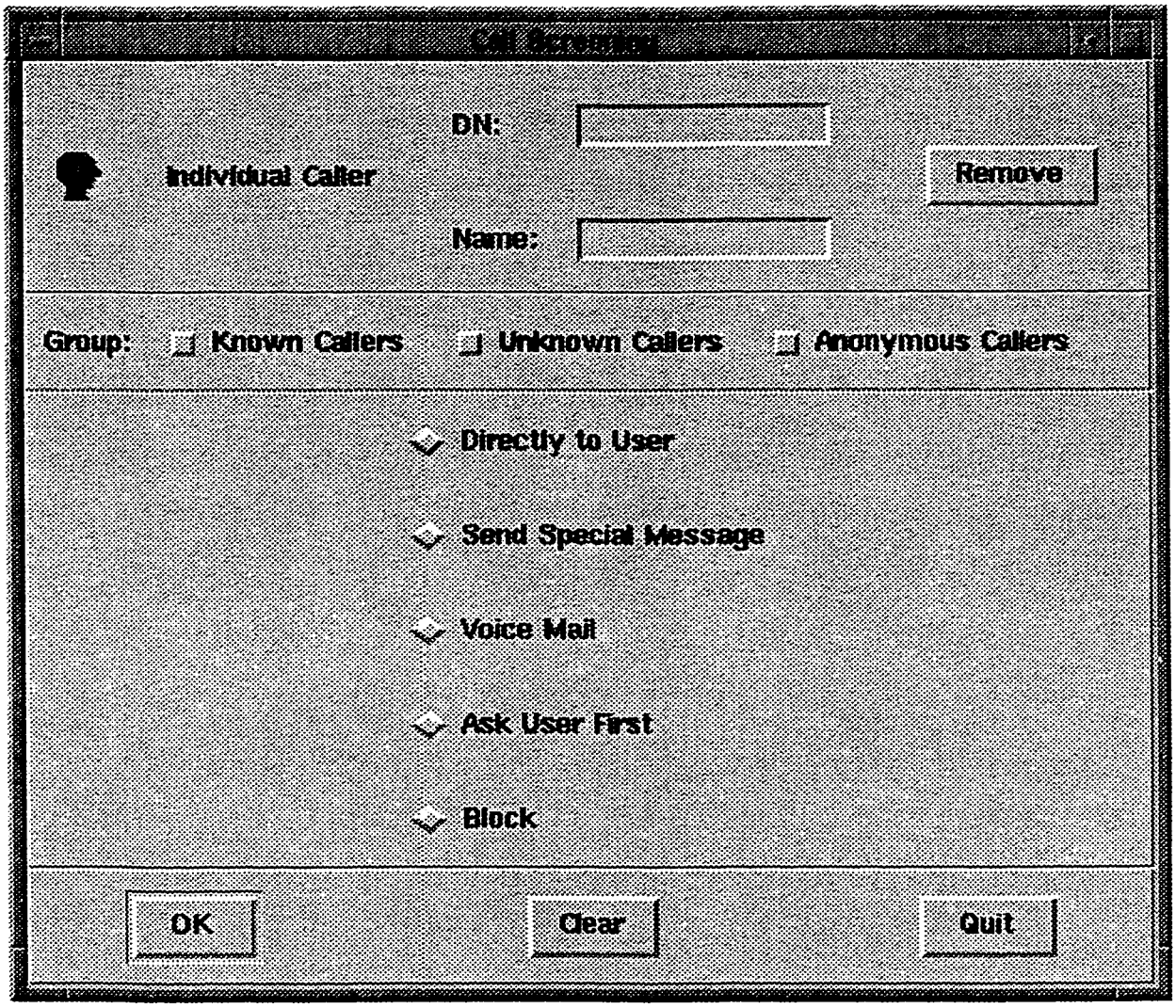

Figure 15: User Interface of CALL SCREENING Feature

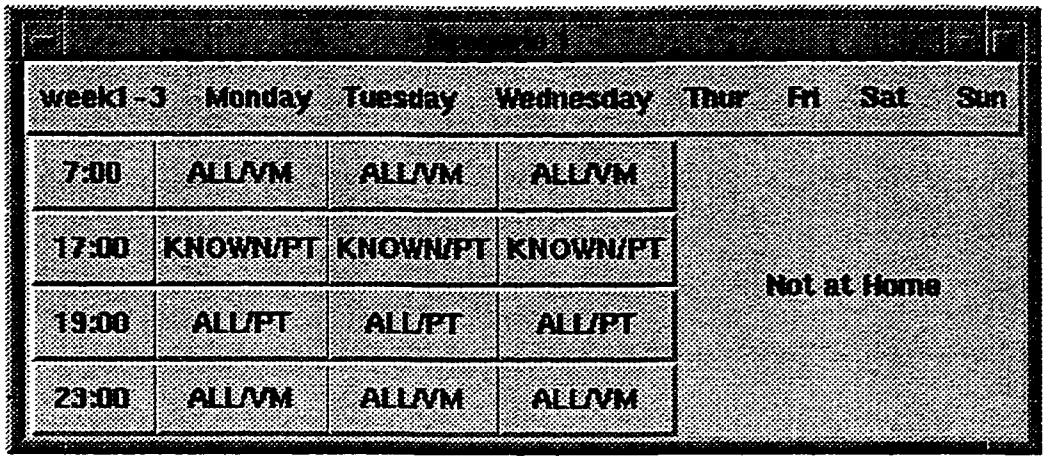

Figure 16: Scenario 1 — The Subscriber Uses the CALL SCREENING Feature on Monday, Tuesday and Wednesday

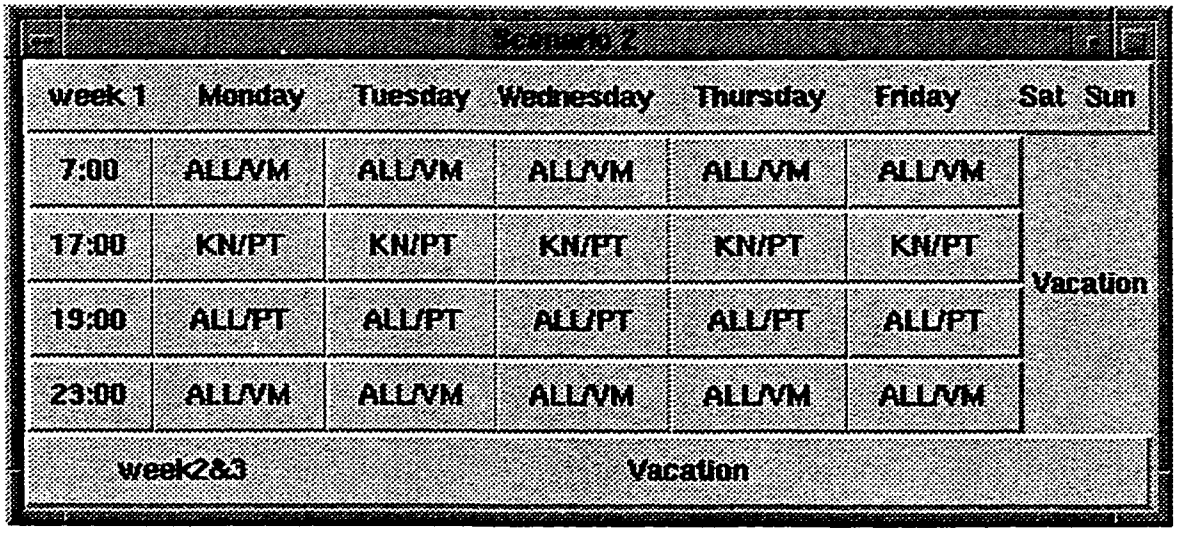

Figure 17: Scenario 2 — The Subscriber Uses the CALL SCREENING Feature in Every Weekday

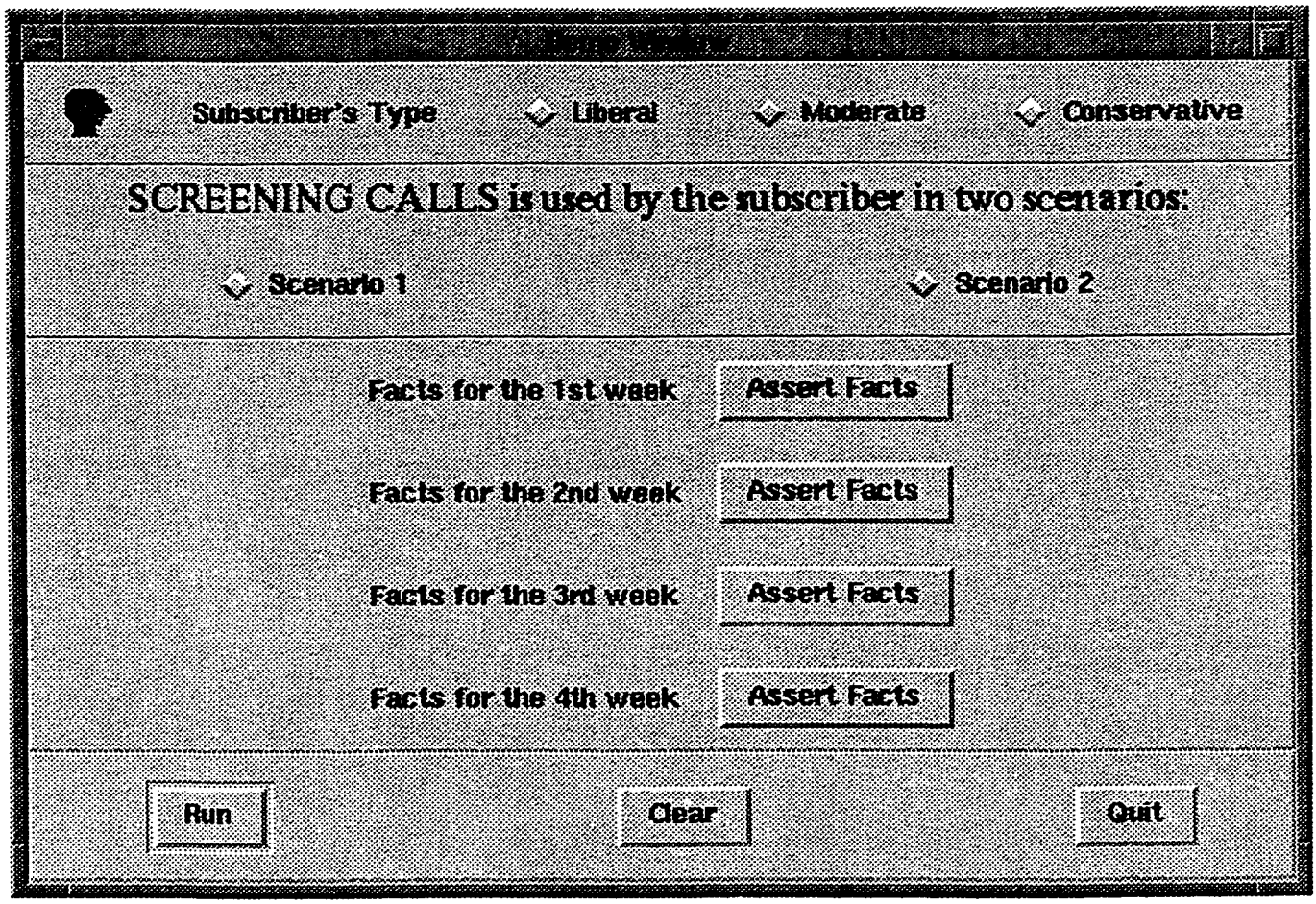

Figure 18: Demo Window

The demo has three parts.

- Part 1 — Assume the subscriber in the first scenario is Liberal type.
- Part 2 — Assume the subscriber in the first scenario is Conservative type.
- Part 3 — Assume the subscriber in the second scenario is Liberal type.

Table 3 shows the criteria used in each part.

	Subscriber Type	Screening Command	Suggestion Rating	Magnitude of Pattern	Phase 1 Identification	Phase 2 Confirmation
Part 1	Liberal	(KN,DTU) (KN,VM) (UKN,VM) (UKN,DTU)	4 2	3 3	2 3	1 2
Part 2	Conservative	(KN,DTU) (KN,VM) (UKN,VM) (UKN,DTU)	4 2	3 3	3 4	2 3
Part 3	Liberal	(KN,DTU) (KN,VM) (UKN,VM) (UKN,DTU)	4 2	5 5	1 2	2 3

Table 3: Criteria Used in Each Demo Part

- Subscriber type — Liberal or Conservative.
- Screening command — shown in the pair of (callers' type, disposition). In the two scenarios of the demo, the subscriber issues four screen commands, as shown in the third column.

  (KN, DTU) refers to known caller and directly to user disposition;

  (KN, VM) refers to known caller and voice mail disposition;

  (UKN, VM) refers to unknown caller and voice mail disposition;

  (UKN, DTU) refers to unknown caller and directly to user disposition.

- Suggestion rating — Screening command (KN, DTU), (KN, VM), and (UKN, VM) are more likely be used by the subscriber, the rating for them is 4. Command (UKN, DTU) with rating 2, indicates it is rarely used by the subscriber.
- Magnitude of Pattern — in the first scenario, the subscriber repeats the same action within three days, we say the pattern size is 3 (days); in the second scenario, the pattern size is 5 (days).
- Phase 1 (Identification Phase) — the number of weeks that an agent should take to identify the pattern.
- Phase 2 (Confirmation Phase) — the number of weeks within which the subscriber have to confirm the pattern to prevent it from being discarded.

4.1.1 Part 1 of the Demo

In the first part of the demo, we select LIBERAL and SCENARIO 1 in the DEMO WINDOW. After we asserted facts for the first two weeks, which indicates the subscriber has used the CALL SCREENING feature for two weeks, the agent identifies three patterns (shown in Figure 19). The pattern for (unknown caller, directly to user) has not been identified, since the suggestion rating for this pattern is 2, the identification phase is 3 three weeks, i.e., the agent needs to get more evidence before recognizing it. When the subscriber takes the same action in the Monday morning of the third week by selecting CALL SCREEN feature from the MAIN WINDOW (Figure 14), the patterns have been confirmed (please refer to Figure 20), then the agent will make suggestions, as shown in Figure 21 and Figure 22. (There is no suggestion generated for the third pattern shown in Figure 19, since this pattern hasn't been confirmed by the subscriber.)

After the subscriber refused these two suggestions, the agent proposes to reduce the frequency with which it makes suggestion (Figure 23). The subscriber rejects this suggestion, too. So the agent will keep the subscriber's type unchanged.

Then we assert facts for the third week. Since the subscriber repeats the same pattern in the third week, the old patterns have been identified again. Besides them, the fourth pattern (unknown caller, directly to user) has been identified as well. The

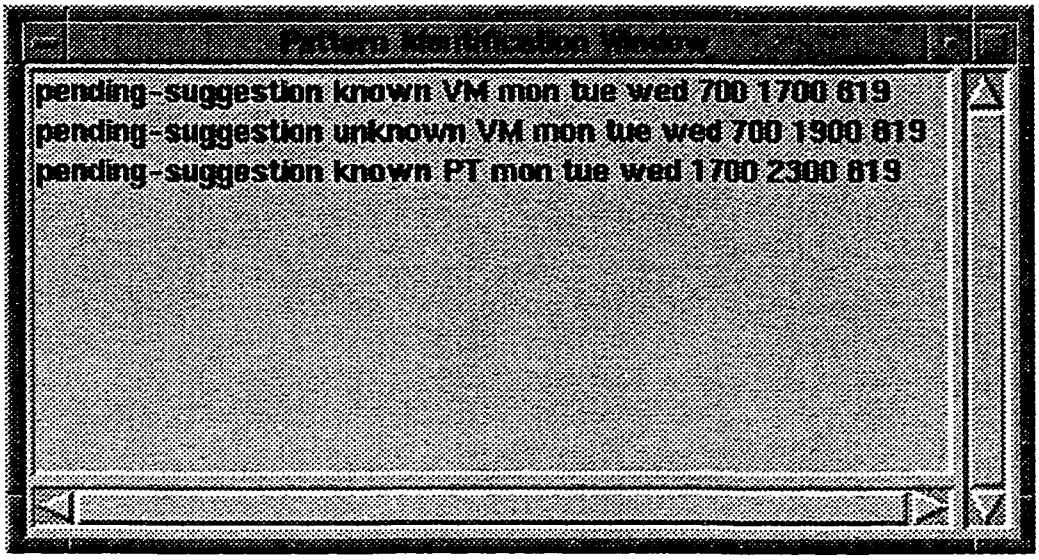

Figure 19: Pattern Identification

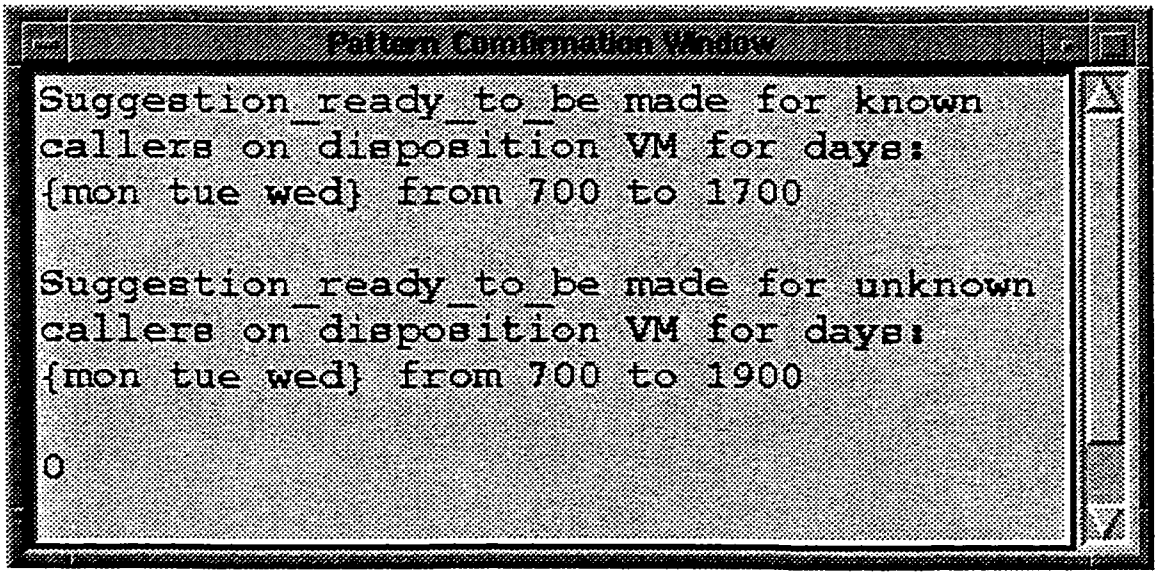

Figure 20: Pattern Confirmation

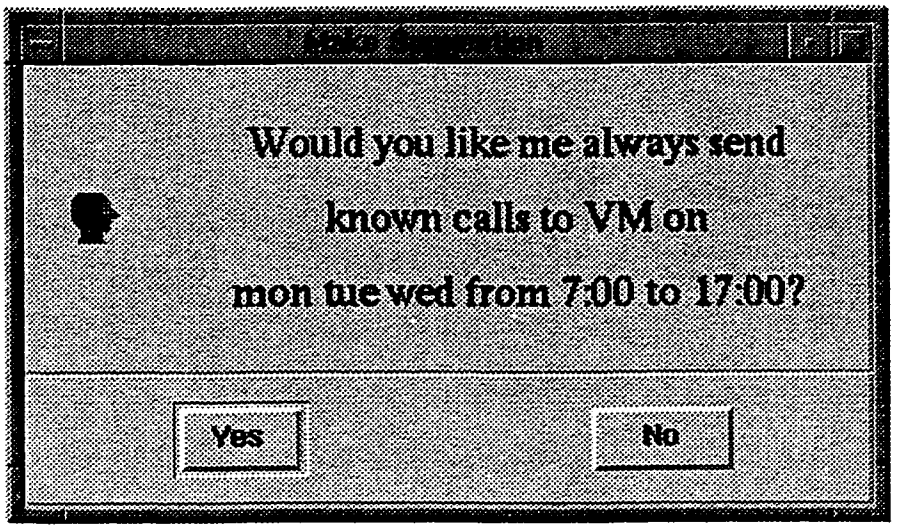

Figure 21: Suggestion 1 — for KNOWN Callers

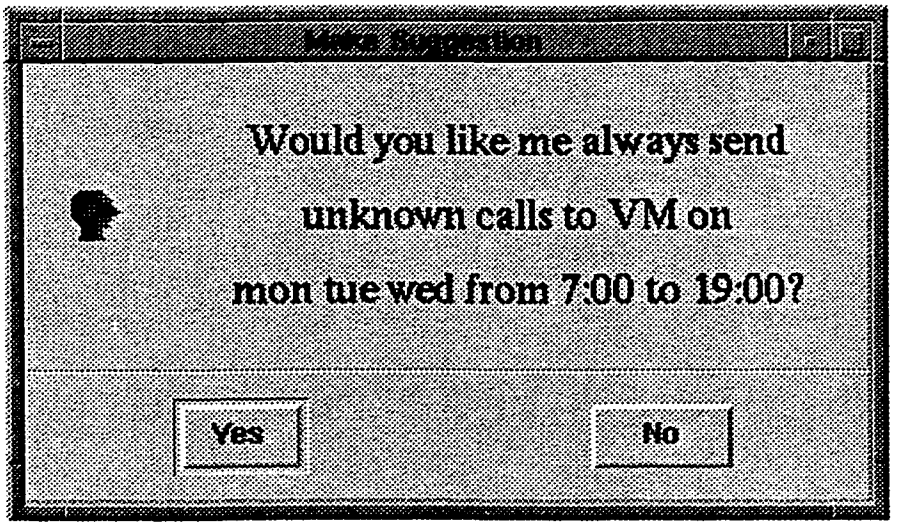

Figure 22: Suggestion 2 — for UNKNOWN Callers

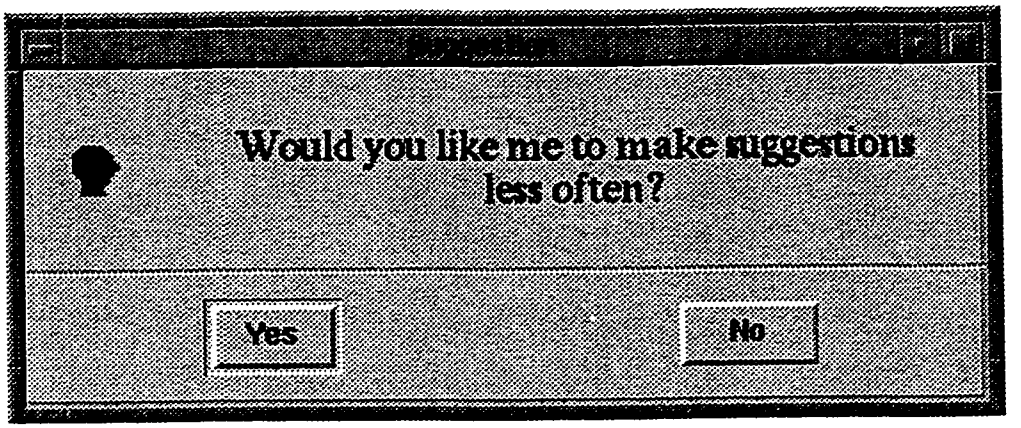

Figure 23: Drop Subscriber's Type One Level Down

agent makes the same suggestions after the subscriber took the same actions in the Monday of the fourth week. At this time, the subscriber approves that the agent can automate sending unknown callers' calls to voice mail from 7:00 to 17:00 in Monday, Tuesday, and Wednesday from now on, but refuses it to do the same thing for known callers' calls. Since it is the second time that the subscriber rejects this suggestion, the agent proposes to stop offering this particular suggestion (please refer to Figure 24).

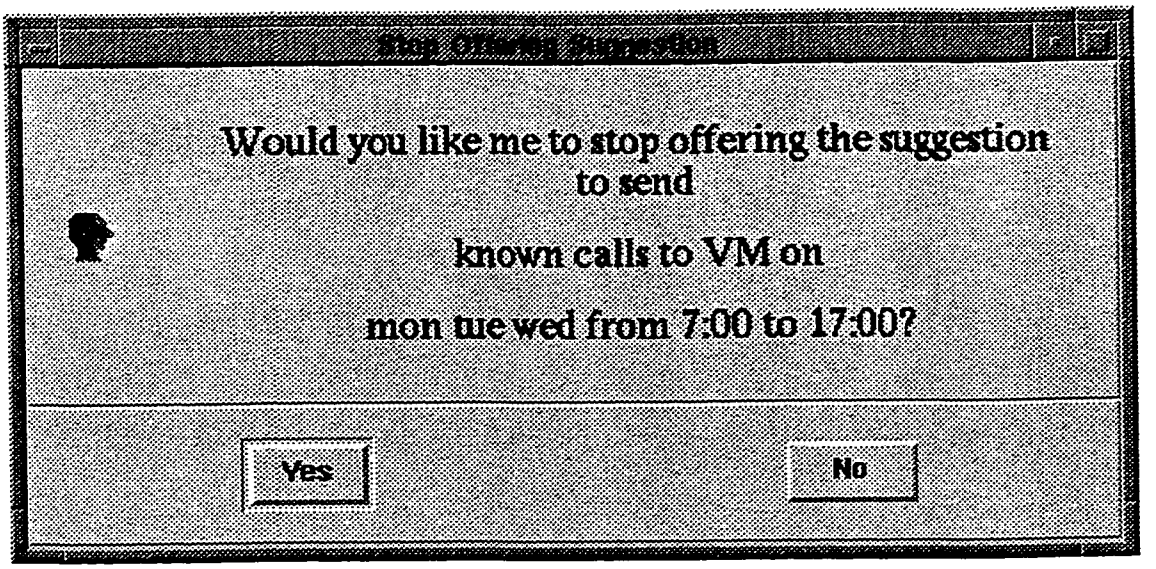

Figure 24: Stop Offering a Particular Suggestion Which Has Been Rejected Twice by the Subscriber

4.1.2 Part 2 of the Demo

In the second part of the demo, CONSERVATIVE and SCENARIO 1 are selected in the DEMO WINDOW. That is, we assume that the subscriber in the first scenario is conservative type. Since the subscriber type is conservative, the agent needs to obtain more evidence than that required for the liberal subscriber to identify a pattern. As shown in Table 3, it takes three weeks for Phase 1 — Identification for the pattern size is three (days) and suggestion rating is 4. After asserting facts for the first and second week, the agent still has no confidence to identify the patterns. Only after asserting facts for the third week, the agent can generate identified patterns. Then agent makes suggestions in the Monday morning of the fourth week after having confirmed that the subscriber still carried out the same patterns. If the subscriber accepted the two suggestions, as shown in Figure 21 and Figure 22), the agent proposes to change the subscriber one level up by making suggestion more frequently (please refer to Figure 25).

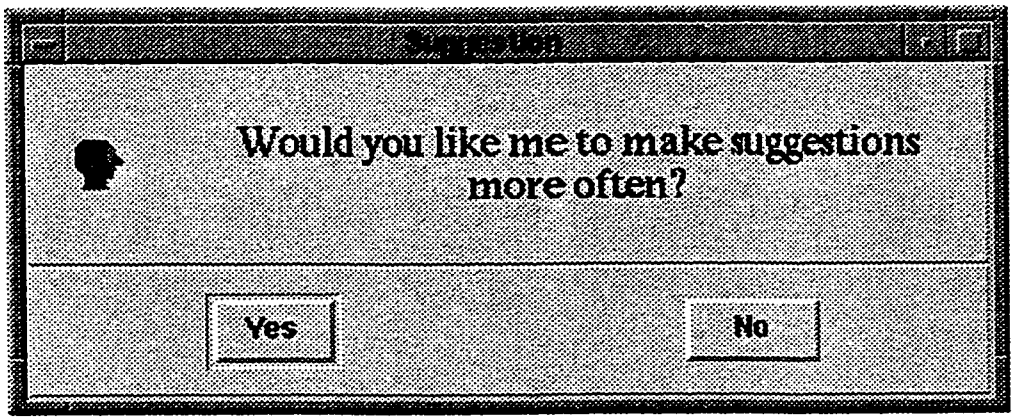

Figure 25: Move Subscriber's Type One Level Up

The pattern of (unknown caller, directly to user) has not been identified during the fourth week. So it is not shown in the Identification Window.

4.1.3 Part 3 of the Demo

In part 3, we assume the subscriber in the second scenario is liberal type by selecting LIBERAL and SCENARIO 2 in the DEMO WINDOW. Since the subscriber repeats the

same action in five days, the agent needs just one week to identify the pattern before creating pending suggestions for (known caller, directly to user), (known caller, voice mail), and (unknown caller, voice mail). (Please refer to Table 3). That is, after asserting facts for the first week, the patterns are generated in working memory as shown in Figure 19. For the reason that the subscriber takes vacation for the following two week, and does not confirm the pattern within its lifetime (one week), the pending suggestions are thrown away. Although the subscriber makes the same action in Monday morning of the fourth week, the agent does not make any suggestion.

4.2 Summary

The three parts of the demonstration present:

- the lifecycle concept of suggestions (Identification Phase and Confirmation Phase);
- the difference of the suggestion rating affects Identification Phase of the suggestions' lifecycle;
- the difference in the subscriber's type affects Identification Phase of the suggestions' lifecycle;
- the difference in magnitude of the pattern affects Identification Phase of the suggestions' lifecycle;
- the proposal for adjusting the subscriber's type;
- the discard of the unconfirmed pattern.

They demonstrate that adding the machine-learning approach to agents' design can make the agent gradually familiars with its subscriber and to support the subscriber's daily-work more efficiently.

Chapter 5

Towards a Theory of Agents

In the previous chapters we have reviewed some examples of current existing agents which are built by different approaches. Meanwhile, we have studied a concrete example of making enhancements for a telephone-based agent. From the knowledge of agents' emergence and functionalities, we can derive a methodology for agents' development. Although we have focused on the interface agents in the previous discussion, the methodology derived from them can be used for the development of interface agents' as well as the development of other types of agents.

The lifecycle of agents' development, as shown in Figure 26), can be categorized in several steps:

1. identifying the application;
2. design and implementation;
3. testing and maintenance.

We will discuss each step in the following sections.

5.1 Identify the Application

The world changes at a rapid pace. We are experiencing the information explosion in our work and home everyday. It makes our life convenient in some aspects. For example, World Wide Web and Internet enable us to search any kind of information in any domains: politics, academics, arts, entertainments, even advertisements, etc.

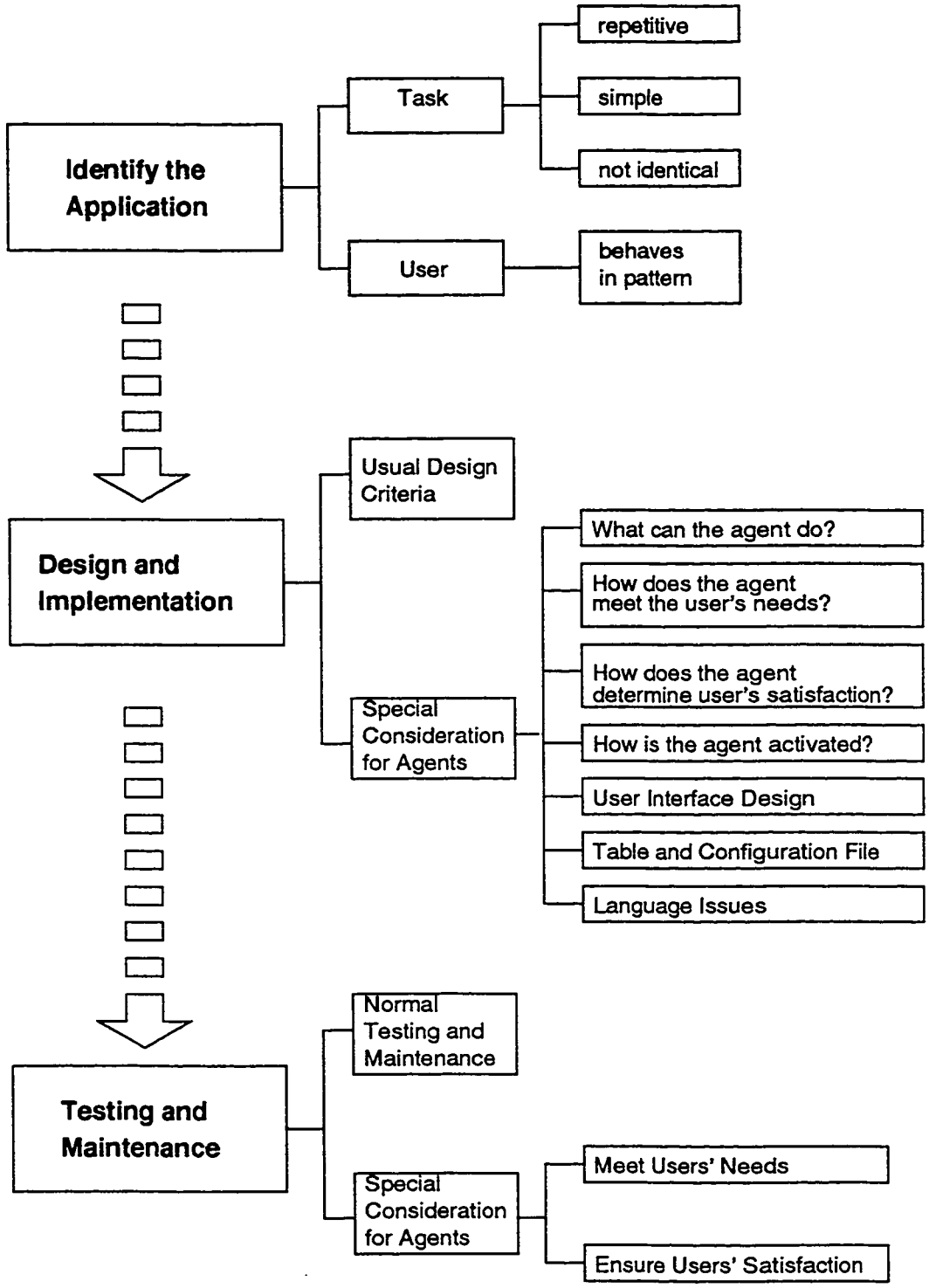

Figure 26: Lifecycle of Agent's Development

But, the amount of information available in the WWW and Internet make many users suffer from information overload. At the same time, many users with no (or with minimal) computer and information retrieval knowledge have to play librarians' roles, who complete their tasks with professional skills. Obviously, users need assistants for their information seeking tasks. Like Internet, which brings convenience and mass together to us, E-mail makes people putting up with E-mail overloading as it provides service to us. It was designed as asynchronous communication application. But now it is used for multiple purposes, such as, document delivery and archiving, work task delegation. It is also used for sending reminders, asking for assistance, scheduling appointments, and posting advertisements. Whittaker and Sinder [WS96] present a quantitative analysis after the study of twenty users' mailboxes, along with thirty-four hours of interview. Many users' inboxes are often in a jumble, piled up by important tasks, partially read messages, jokes, and junk mail. It's the same that users need assistants for managing their personal information. The problems coming with the telephony development have been mentioned in Chapter 3. With more and more information available through telephones, it will take much more time for people to handle their daily calls. In brief, there are many repetitive, tedious tasks which are undertaken by people in their daily life that can be automated by intelligent assistants. The emergence of software agents gives a new possible way for solving these problems.

After being endowed with learning ability, software agents can provide services that are more suited for users. For example, Letizia [Lie95] is an agent that assists a user browsing the World Wide Web. When the user operates a conventional Web browser, Letizia uses the past behaviour of the user to anticipate a rough approximation of the user's interests. It can initiate autonomous exploration of links from user's current position, finding information interested in by the user, releasing users from information overload. Leitzia can be used by different users in different interesting domains.

Another example is Maxims [Mae94], an agent assisting the user with electronic mail. It continuously observes the user when the user deals with E-mail. It learns to prioritize, delete, forward, sort and archive mail messages on behalf of the user. And

it can also collaborate with other agents, asking for help from them which are assisting other users with e-mail, when it doesn't have enough confidence in its prediction.

We have explained how the telephone-based agent, Wildfire, helps people to manage phone calls. It can screen incoming calls, place outgoing calls, and manage incoming voice messages and play special message to an individual or a group specified by the user. There are many other agent examples for alleviating people from repetitive tasks. For example, meeting scheduler agents schedule the meetings for the user, and shopping agents try to find the lowest price from many retailers for people.

In the examples presented above, there are several common features in the tasks which the agents can perform on behalf of users:

- the task must be repetitive; otherwise it is not efficient to design an agent just for rarely required tasks;
- they must be simple; otherwise it is beyond the agent's intelligence;
- they must not be identical; otherwise a simple knowledge-based system can solve the problem;
- the user must behave in patterns; otherwise the agent cannot learn (at least the learning speed is slow).

Besides the main advantage — alleviating people from repetitive tasks, from economic and psychology points of view, software agents also make other benefits to users. For examples:

- they save users' time in order to allow user to do more creative tasks or have a rest;
- they are less expensive than human assistants. Once the agent was bought, it can be used by the user as long as he/she wants;
- they are more productive since they do not show the problems which always occur in human performance such as fatigue, overwork, and stress;
- their ability is quickly augmented by the introduction of new software;

- they are more adaptive than human assistants for different "bosses", since they coordinate user's performance without preconditions.

Obviously, after we identify that there are repetitive tasks which might be automated, the idea of building an agent to complete these tasks is undoubtedly logical.

5.2 Design and Implementation

Since an agent is just a piece of software, implementing it is not all that different from implementing any other kinds of softwares. All (or, at least, most) of the usual criteria for "good software engineering practice" apply to it. We will not discuss well-known software engineer practices in the thesis (details may be seen in [GJM91]). We will give a brief discussion for software process models from which software agent development can make reference. There are three main methodologies in conventional software engineering. They are: (1)waterfall model, in with which the software is delivered once through; (2)spiral model, which has several iterations in design process, and is risk driven; (3)evolutionary model, which is increment driven by using the prototyping technique. The first two models are not suitable for an agent's development. Evolutionary model can be referred when building an agent. By using prototyping technique, a simple agent can be created for initial trial, and a series of agents which are better than the previous one will be built later on. So designers can start thinking on additional and following products in an early stage. There is no critical changes with which conflict the original principles are taken in a later phase. This approach is beneficial to build an agent which can "grow up" with users efficiently.

Section 5.2.1 through section 5.2.7 will discuss some special design issues in agent's development.

5.2.1 What can be Automated?

Although agents come out as assistants for humans, and can assist people to complete some tasks, they cannot substitute humans to do all tasks. There are limitations to agents' abilities. And the abilities of agents are different from type to type. For instance, a search agent is different from an E-mail agent and a telephone-agent from the aspect of its ability. Because search agents can do all the needed search for users

without users' interfering, but E-mail agents and telephone agents may need instructions from their user in some unexperienced scenarios. They have to ask users' help to complete tasks or simply leave the tasks to users. For instance, when an E-mail agent gets a message from a person who has never sent message to the user before, and the subject of the message has never been recorded by the agent either, the agent will ask its user to decide how to deal with this new message. Because the agent has no experience for this kind of message, no example can be used in this scenario. If the message is come from a long lost friend, the user may reply it immediately; if it is junk mail, the user may delete it. So, the agent will obtain an example which can be referenced in a similar case later on.

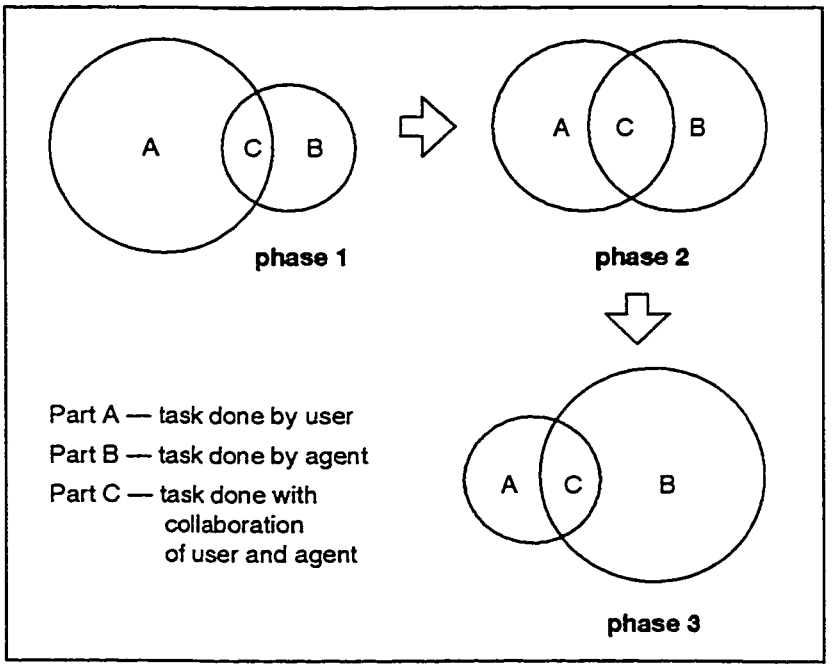

Figure 27: Tasks Automated by an Agent

When designing an agent, we have to figure out if the entire task can be automated by agents, or just part of it. If it is not the first case, the entire task can be categorized in three parts:

- tasks done by agent;
- tasks done by user;

- task done with collaboration of user and agent.

In this case, the important issue is whether the non-automated part can be automated by the agent after it learns from the user over a period of time. That is how to improve agents' abilities so as to enable them to take charge of more tasks. As shown in Figure 27, in phase 1, the agent can automated only two-fifths of entire task. By learning from the user in collaboration, the agent may gradually execute more and more tasks. In phase 3, it might automate four-fifths of the entire task. Releasing people from labour to a maximum degree, that is, changing the proportion of Part A and Part B in the entire task (Figure 27), is the basic concern of the agent technology.

Just as humans can improve their problem-solving ability from practice and learning, agents abilities can be improved as well. When we encounter any problem, the method for solving it stems from our understanding of the problem and the experience that we have for this kind of problem. There are two different kinds of experience we can have: direct experience, which we experience by ourselves; and indirect experience, which are others' experience that we usually learned, for example, from books. Since it is impossible to experience everything in one's life, learning from others' experience can save time and avoid mistakes ever having been made by others. Based on this knowledge, we can increase agents' abilities by having them observe users' behaviour and learn from it, and also give them direct instruction at appropriate time when there is no other examples from which they can learn. We will discuss this issue in more detail in Section 5.2.2.

Generally speaking, agents can take care of repetitive, simple tasks on behalf of users, as they obtain more knowledge about the task and its users, they can achieve more complex goals and release people from tedious work.

5.2.2 How does the Agent Determine the User's Needs?

When an agent is taking care a task on behalf of its user, it must be sure that its behaviour meets its user's requirements. As we mentioned in last section, the agent has to have knowledge of the tasks, and even of the preference of its users, in order to determine user's needs, provide proper assistance to users, and increase their problem-solving abilities. There are three different ways by which agents can acquire

this knowledge:

1. agents have the required knowledge when they are built. The knowledge may come from user (User-Programming approach) or expert (Knowledge-Based approach);

2. agents observe users' behaviour and record the behaviour in order to learn;

3. agents learn from other agents.

The first method can be said as "direct experience" in machine-learning, since agents are endowed with knowledge when they are built. The second and third methods are similar to direct experience and indirect experience in human-learning, respectively. Direct experience and indirect experience are all important in building our knowledge. It is the same in building agents' intelligence. Sometimes there is no base of examples for a learning approach to work. Terveen and Murray [TM96] give an example for this case. Suppose that you are on vacation, you may want a vacation message sent in response to all incoming messages. There is no time for the agent to collect examples and learn from them, since you want the rule to take effect immediately. Further, the dates of the vacation changes from one vacation to another, there is no stable rule for agent's learning. In this case, the user should give explicit instruction to the agent, so that it works immediately. As we mentioned in Chapter 2, the approach of learning from other agent remedies the shortcomings of learning from observing a user's behaviour. Since the observation takes time, and the agent cannot offer assistance with no base examples. By using the three methods together, we can improve agents' assistant abilities.

There are two factors which can affect the agent's understanding of its task and user. They are base knowledge and learning speed. Base knowledge refers to agents' experience, and learning speed is mainly determined by user's behaviours since agents learns from observing and recording users' behaviour pattern. As shown in Figure 28(a), obviously, if the learning speed is the same, the agent with built-in knowledge will get a better understanding of its task and user than the agent who only relies on its observation but no other knowledge before. If a user behaves consistently, his/her agent learns fast. In the contrast, if the user behaves inconsistently,

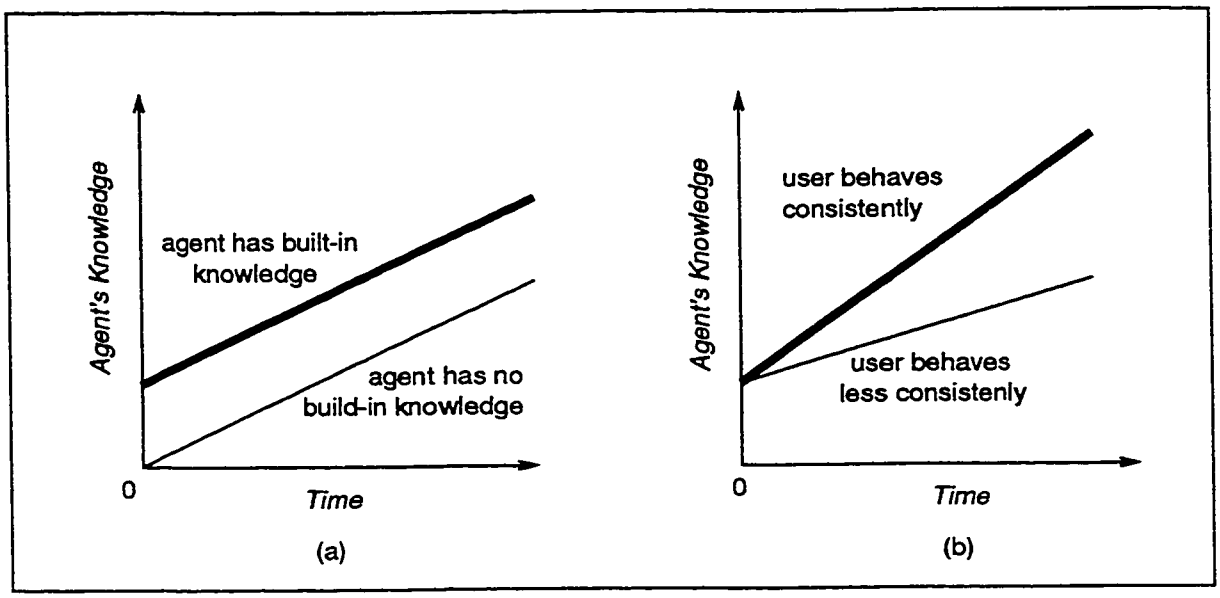

Figure 28: Two Factors Affect Agent's Knowledge Acquisition

the learning speed of the agent will be slow (Figure 28(b)).

Since users' personalities are different from one other, their needs cannot be the same. As we mentioned in Chapter 3, the agent should classifies users into different groups, in order to meet their different requirements.

The question followed are how to find patterns of user behaviour and what kind of pattern the agent should look for? For the first question, by observing and recording the user's behaviour, the agent can identify the patterns. One important issue, which is worth being pointed out here, is that learning from the observation should be time-based and event-based. For one kind of task, such as managing phone calls, time-based is more important. For tasks such as filtering e-mails, event-based learning should be emphasised. But for scheduling meetings, both time-based and event-based observations are absolutely necessary. It depends on the tasks assigned to the agent. Caglayan *et al.* [CSJ+96] give a lesson that time-based learning suggestions without the context of others events are frequently not useful in *Open Sesame!*. They have an example as follows:

> An observation such as: *"I notice you often open your AppleCD Audio Player at 11:15 AM."* is meaningless without knowing whether a CD is mounted in the CD-ROM drive. A more appropriate observation would be: *"I notice you open your AppleCD Audio Player when an audio CD is mounted"* (Quoted from Caglayan et al. [CSJ+96]).

The answer to the second question, what kind of pattern the agent should be looking for, also depends on the characteristics of the particular tasks.

5.2.3 How does the Agent Determine the User's Satisfaction?

We have already talked about how important the usability is for a system's acceptability in Chapter 2. In "Usability Engineering" [Nie93], Nielsen pointed out that usability has five main attributes: *learnability, efficiency, memorability, errors,* and *satisfaction. Satisfaction* refers to how pleasant it is to use the system. That is, users are subjectively satisfied when using it. So in order to build an agent that satisfies users, the agent should be able to provide good service to users just as human assistants do.

When an agent provides assistance to its user, its greatest concern is to avoid annoying the user. If the user is annoyed by an agent before he/she get any help from it, the agent might have no chance to show how valuable it is to the user, since the user may give up getting any assistance from it. "Talking automobiles" is an example for this case. Beginning in the early 1980s, some automobile manufactures decided to put speech synthesis chips in automobiles to signal problem conditions. The driver wouldn't have to deduce what the problem was. The talking indicators lasted for about two years. Many consumers perceived them as annoying and not particularly informative.

For interface agents, keeping this concern in mind is more important than for other types of agents. Besides providing general services to users as any other applications do, interface agents may also has their social nature — interacting to the user. When an interface agent interrupts the user at the wrong time or makes a less

than helpful suggestion, it will be criticized at "human" level: "it looks less intelligent". For proactive agents, when they have suggestions after their observation, these suggestions must be offered on time-based and event-based principles.

The most important criteria which justify users' satisfaction is feedback from the users. There are two kinds of feedback that users may have:

1. explicit feedback — refers to the user's saying *"good/bad"* or *"yes/no"* for the agent's suggestions;

2. implicit feedback — user's frequently using the agent means that he/she is satisfied with its assistance; user's giving up to use the agent means that he/she is not satisfied with the agent, and has no more interest in using it.

Since implicit feedback cannot be asked from the user, the agent is not able to make any adjustment to itself using implicit feedback. Once the user gives up using it, the process is irreversible. Explicit feedback is the only way from which the agent can acquire how much the user is satisfied with its performance. The agent should not only ask for explicit feedback from the user and assess user's response, the more important issue is that it should be able to modify its behaviour according to its user's explicit feedback. In the following discussion, we use "feedback" to refer to explicit feedback.

Although the expression of feedback given to agents is simple, just *yes* or *no*, or *good* or *bad*, the meaning of it is abundant. For example, an agent makes a suggestion to its user, say, *"Would you like me to always remove e-mails which comes from unknown person?"* When the user say *"no"*, it may mean:

- the user is not sure if he/she would like the agent to do it automatically, but for the sake of safety, she/he would like to have the control on this kind of task;

- the user is too busy to think about this suggestion, he/she would like to put this suggestion aside for the time being;

- the user may really mean "no".

In the first two cases, the user may not mind if the agent make the same suggestion again. But in the third case, the user may feel annoyed when the agent re-offers this rejected suggestion. So how to treat the feedback obtained from the user is important.

5.2.4 How is the Agent Activated?

There must be some event that activates the agent. Different kinds of agents require different kinds of events to trigger them. For examples:

- telephone-based agent — incoming telephone call;
- web search assistant — user starts browsing web;
- OS assistant — user issues any OS command;
- shopping assistant — user starts shopping program.

Alternatively, the user may explicitly start the agent, for instance, a user commands for a outgoing call to a telephone-based agent.

5.2.5 User Interface

The UI is an important component of an agent. It is important to design and implement the UI with careful consideration for the needs of potential users. At the same time, as mentioned in Chapter 2, an appropriate interface is also critical to the survival of software product. A poorly designed UI can ruin the chances of an agent to gain success in the market place. The following issues can be taken as design criteria:

- easy to learn — users should have confidence to get started;
- easy to understand — users should not be confused;
- present only when necessary — screen should not be cluttered by dormant agents;
- customizable — users may choose which parts of UI they want to see and use;
- simple — users should not be confused by a plethora of options;
- adequate — users should be able to do what they want.

In a short, an good agent should provide not only useful functionalities, but also a "pleasure to use" user interface.

5.2.6 Tables and Configuration Files

If the agent's behaviour is embedded in code, it will be hard to customize and modify the agent. Consequently, it is better to describe the agent's behaviour using tabular data or configuration files that can be easily changed without modifying and recompiling code. Tables can be changed by the vendor of the agent or perhaps even by the users themselves in some special cases. For example, for the E-mail and telephone-based agent, the user can give the explicit instructions to its agent for changing their behaviour temporarily (or immediately when there is no time for the agent to collect examples for making decision). If users can change tables, the program must be very robust (so that it does not do stupid things if the tables contain invalid entries) and the vendor must provide documentation (so that the user can make sensible changes). Alternatively, the vendor can provide a UI that allows the user to change settings.

Furthermore, users should be allowed to describe and change their profiles — their preferences and habits. For example, the user may change the baseline and ask the agent to frequently check a stock quote over the internet. If the price drops below the baseline, the agent will alert its user by paging him/her. Or, an agent can check on the inventory of a product and if stock is running lower than the baseline, automatically E-mail a refill order to a supplier. Another example, in FATMA application, there should be a way that allows the subscriber to give a judgement for his/her personality, such as *"I'm a liberal person"*, or *"I'm a conservative person"*. So that the agent can have a reference for its later decision. (As in a bank, when you plan to buy Mutual Funds your agent always asks you whether you are an adventurous person or not, then they can help you to do some investments.) Sometimes people may not have right judgements for themselves. The agent should be able to provide a set of questions for the user, and make a judgement for its user according to user's answers. By allowing the user to change his/her profile, the agent may adapt itself to suit individual user more quickly.

5.2.7 Language Issues

An agent is a complex piece of software. There is no programming language that allows all aspects of an agent to be implemented simply and efficiently. Consequently, agents are typically implemented using several languages or code generators. For

examples: C/C++ are used for low level programming; Clips for rule-based programming; Prolog for logical deduction; Tcl/Tk for GUI. (Some research has been done for building agent oriented programming languages for *multi-agent systems.* Since it is out of the scope of this thesis, we will not discuss it here. Readers who are interested in this field may refer to the following papers: [Bur94, MC94, KG97, PM96]).

Some researchers are trying to provide users an easy way to build an application based on intelligent agents or to add agents to an existing application. IBM's new Agent Building Environment (ABE) [IBM97], a toolkit in Java, is a good example. It has a very clean architecture and embeds "Artificial Intelligence" components very well, such as knowledge, inference engine, etc. It consists of four parts: (1) a rule-based inference engine called RAISE; (2) adapters, which provide the interface to the "real world" by reacting to trigger events and communicating with the engine; (3) a library to hold facts and rules that the engine uses to intelligently guide the agent; and (4) a generic rule editor for writing and editing rules. Users can write their own adapters as well and guidelines. So agents can behaviour based on users' preferences.

5.3 Testing and Maintenance

As mentioned in Section 5.2, all of the usual practices for testing and maintenance apply to agents, because they are software. In this section, we will discuss ways in which agent testing and maintenance is special.

5.3.1 Testing

There are two kinds of testing for software agents: (1) testing the agent behaves correctly; and (2) testing the agent satisfies its users. They refer to *Functional Testing* and *Usability Testing*, respectively.

Functional Testing

The agent must perform according to its specification without performing incorrect or inappropriate actions, crashing, etc. This is standard practice.

Usability Testing

This testing should demonstrate that the agent does meet users' needs and does satisfy users. So the agent must exhibit the special behaviour required of agents. For example, the agent must:

- ask for information at appropriate times and intervals;
- learn from the user's responses and behaviour;
- not request the same information more than once (this is equivalent to correct learning);
- apply information that it has learned correctly;
- interact with the user properly.

It is unlikely that usability testing can be performed in the conventional way by a team of test engineers. It will probably be necessary to select a number of trial users, with different work habits and personalities (since an agent might please one user and annoy others). The trial users will have to use the agent for an extended period of time — weeks or months. During the trials:

- the agent should keep a record of all transactions for subsequent analysis;
- the trial user should record both positive and negative experiences with the agent.

There are two basic methods used to measure users' satisfaction to a software: *likert scale* and semantic differential scale [Nie93]. For a Likert scale, designers provide a set of statements, and users would indicate their degree of agreement on a 1-5 scale for each statement. The statement could be, for example, *"This system can do all the things I think I would need"*. For a semantic differential scale, designers list two opposite terms along some dimension, (for example, pleasing vs. irritating) and asks the user to place the system on the most appropriate rating along the dimension. These two methods can be refered to help user to record their experience with the agent.

5.3.2 Maintenance

Maintenance is another important phase in software agent design. It should not only keep agents running in good condition, but also adjust agent's "growing" direction. There are two aspects in maintenance phase:

- standard aspects: correcting errors, improving performance, adding features, etc.
- non-standard aspects: it will be important to obtain feedback from users to decide what features should be added, removed, or altered.

5.4 Summary

Acting as human assistants, agents are different from other kinds of softwares. They can not only complete the task on behalf of users, but also adapt themselves to users' personalities. We have developed a methodology which can be followed as a general approach on how to meet user's needs and satisfy user's behaviour in agent's design.

Chapter 6

Conclusion

Agents recently became the focus of both computer science research and the software industry. With agent usage becoming more and more widespread, problems related to agent's usability have arisen: how can an agent provide useful help without annoying its users? Will users trust agents? What are the risks involved? In this section, the contributions we have made on increasing the usefulness of software agents discussed, and some suggestions for further research are given.

6.1 Contributions to Software Agent Field

6.1.1 In General

In this thesis, we describe some enhancements to a telephone-based agent, which can make suggestions to its subscriber unobtrusively with appropriate frequency by applying machine learning and feedback learning techniques. Furthermore, based on the study of current existing agents and the implementation of the enhanced agent, we derived a methodology for agent development.

Agents are different from other kinds of software. Consequently, some special considerations should not be ignored in an agent's development. Since building an agent which the user will like is an important issue in the agent's design, which concerns increasing the usefulness of the agent, making them meet users' needs and satisfy users, we emphasised these criteria through out the lifecycle of agent's development. Following this approach, the agent can not only complete the task on behalf of users,

but also suit for users with different personalities.

How to improve the trustfulness of agents is also a critical issue in agent's development. Most users will exercise caution when they use an agent for the first time: they will check its actions carefully to ensure that the agent is doing what they want. After a while, however, users will come to trust their agents more and more, allowing them to make decisions without consultation or checking. Meanwhile, with more and more agents getting into people's daily lives, criminals may exploit an advantage of agent's trustfulness. Users who trust agents completely are subject to risks. The following two examples demonstrate the potential risks:

> A telephone answering agent receives an emergency call, decides that the the hospital is trying to sell something, and rejects the call.
>
> A financial assistant, programmed to make simple sales and purchases, responds to a scam by buying 1,000,000 shares of a goldmine in Antarctica.

The problems shown in these examples are that people can usually distinguish "routine" situations from "emergency" or "critical" situations. They use many criteria to do this (source of message, tone of voice, amount of money, etc). Ideally, agents are expected to make such distinctions reliably. By ensuring that agents meet users' needs and that agents satisfy users, and by proposing a methodology that allows the construction of agents that meet these criteria, our study helps to solve the problems mentioned above.

6.1.2 Enhancements for FATMA

This thesis describes enhancements made to FATMA. Also, a demonstration of how the enhanced agent adjusts its behaviour to support its subscriber's daily work is presented.

The main goal for making enhancements for FATMA is to avoid the agent annoying subscribers and to improve the performance of the agent. The enhanced agent can specify the differences among subscribers and can adjust its behaviour according to the feedback provided by the subscriber when it makes suggestions.

The enhancements that we have made include the following aspects:

1. categorizing the subscribers in order to allow the agent to gather different amounts of evidence before offering suggestions to each type of subscriber;

2. categorizing CALL SCREENING suggestions, so that suggestions concerning caller types and dispositions that are more likely to be used by subscriber require less evidence to be offered;

3. creating a lifecycle concept to update agents' "memory"; and

4. using feedback from the subscriber to adjust subcriber's category or to stop offering a particular type of suggestion.

6.2 Future Work

This study only touches upon one aspect of improving agent's usability in terms of meeting users' needs and being satisfied by users. The next step could be focusing on other aspects to improve agents' usefulness further, such as ease of using the agent for the first time, ease of learning, adaptation to both novice and expert users, and differentiating users more precisely (according to their personalities). Meanwhile, agents' learnability can be further improved with the progress made in AI and machine learning fields.

Also, some further enhancements for the telephone-based agent can be made by studying rule sets for adaptive dialog, for novice and expert users; and determining if part of the CALL SCREENING rule sets are reusable for other telephone-based agents' features, such as, CALL FORWARDING.

Bibliography

[BD81] Benbasat, J. and Dexter, A. An Experimental Study of the Human Computer Interface. *Communications of the ACM, Volume 24*, pages 752–762, 1981.

[BKL88] Bailey, W. A., Knox, S. T., and Lynch, E. F. Effects of Interface Design upon User Productivity. In *CHI'88 Conference Proceedings on Human Factors in Computer Systems*, pages 207–212, 1988.

[Bro58] Broadbent, Donald Eric. *Perception and communication.* New York : Pergamon Press, 1958.

[Bur94] Burkhard, Hans-Dieter. Agent-Oriented Programming for Open Systems. In Müller, J. P., Wooldridge M. J., and Jennings N. R., editors, *ECAI-94 Workshop on Agent Theories, Architectures, and Languages*, pages 291–306. Springer-Verlag, 1994.

[CSJ+96] Caglayan, A., Snorrason, M., Jacoby, J., Mazzu, J., and Jones, R. Lesson from Open Sesame!, A User Interface Learning Agent. In *Proceedings the First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM96)*, pages 61–74, April 1996.

[Fey93] Feynman, C. Nearest Neighbor and Maximum Likelihood Methods for Social Information Filtering. Internal Report, Media Laboratory, MIT, December 1993.

[FF87] Fischler, Martin A. and Firschein, Oscar. *Intelligence, the Eye, the Brain, and the Computer.* Addison-Wesley Publishing Company, Inc., 1987.

[For87] Ford, Nigel. *How Machines Think.* John Wiley and Sons Ltd., 1987.

[GBL+87] Gould, J.D., Boies, S. J., Levy, S., Richard, J., and Schoonard, J. The 1984 Olympic Message System: A Test of Behavioral Principles of System Design. *Communications of the ACM, Volume 30*, pages 758–769, 1987.

[GJM91] Ghezzi, Carlo, Jazayeri, Mehdi, and Mandrioli, Dino. *Fundamentals of Software Engineering.* Prentice-Hall Inc., 1991.

[GLC+96] Grosof, Benjamin N., Levine, David W., Chan, Hoi Y., Parris, Colin J., and Auerbach, Joshua S. Reusable Architecture for Embedding Rule-Based Intelligence in Information Agents. *CyberJournal, http://www.research.ibm.com*, 1996.

[GLR96] Goldman, C. V., Langer, A., and Rosenschein, J. S. Musag: An Agent that Learns what You Mean. In *Proceedings the First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM96)*, pages 311–329, April 1996.

[HEA90] Hammainen, H., Eloranta, E., and Alasuvonto, J. Distributed Form Management. *ACM Transactions on Information Systems, Volume 8, No. 1*, pages 50–76, January 1990.

[IBM97] IBM Corporation. IBM Agent Building Environment (ABE) Homepage. *http://www.networking.ibm.com/iag/iagsoft.htm*, 1997.

[Jac74] Jackson, Philip C. *Introduction to Artificial Intelligence.* New York: Petrocelli Books, 1974.

[KG97] Kinny D. and Georgeff, M. Modelling and Design of Multi-Agent Systems. In Müller, J. P., Wooldridge M. J., and Jennings N. R., editors, *Intelligent Agents III — Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages (ATAL-96)*, pages 1–20. Springer-Verlag, 1997.

[Kin95] King, J. A. Intelligent Agents: Bringing Good Things to Life. *AI Expert*, pages 17–19, February 1995.

[KK87] Kaye, A. R. and Karam, G. M. Cooperating Knowledge-based Assistants for the Office. *ACM Transactions on Information Systems, Volume 5, No. 4* , pages 297–326, October 1987.

[KM90] Kreutzer, Wolfgang and Mckenzie, Bruce. *Programming for Artificial Intelligence: Methods, Tools and Applications.* Sydney ; Reading, Mass. : Addison-Wesley Publishing Company, Inc., 1990.

[KM93] Kozierok, R. and Maes, P. A Learning Interface Agent for Scheduling Meetings. In *ACM SIGCHI International Workshop on Intellegent User Interfaces, ACM, Orlando, Florida*, January 1993.

[Koz93] Kozierok, R. A Learning Approach to Knowledge Acquisition for Intelligent Interface Agents. Master's thesis, Department of Electrical Engineering and Computer Science, MIT, May 1993.

[Lan88] Landauer, Thomas K. Research Methods in Human-Computer Interaction. In Helander, Martin, editor, *Handbook of Human-Computer Interaction*, chapter 42, page 906. Elsevier Science Publishers B.V., 1988.

[Lie95] Lieberman, H. Letizia: An Agent that Assists Web Browsing. In *Proceedings of IJCAI95.* AAAI Press, 1995.

[LM88] Lai, K. Y. and Malone, T. W. Object Lens: A "Spreadsheet" for Cooperative Work. *ACM Transactions on Information Systems, Volume 6. No.4*, pages 332–353, October 1988.

[LMM94] Lashkari, Y., Metral, M., and Maes, P. Collaborative Interface Agents. In *Proceedings of the National Conference on Artificial Intelligence*, pages 444–449. MIT Press, Cambridge, Mass., July 1994.

[Mac91] Macdonald, B. A. Instructable Systems. *Knowledge Acquisition 3, 4*, pages 381–420, December 1991.

[Mae94] Maes, P. Agents that Reduce Work and Information Overload. *Communications of the ACM, Vol. 37, No. 7*, July 1994.

[Mae95] Maes, P. Intelligent Software. *Scientific American, Vol. 273, No. 3*, pages 84–86, September 1995.

[Mar90] Martin, M. P. Human Factors in Information Systems. In *Proceedings of the Third Symposium of Human Factors in Information Systems*, pages 193–200, 1990.

[MC94] M[c]cabe, F. G. and Clark, K. L. April- Agent PRocess Interaction Language. In Wooldridge M. J. and Jennings N. R., editors, *ECAI-94 Workshop on Agent Theories, Architectures, and Languages*, pages 324–340. Springer-Verlag, 1994.

[MGL+87] Malone, T. W., Grant, K. R., Lai, K. Y., Rao, R., and Rosenblitt, D. Semi-Structured Messages are Surprisingly Useful for Computer-Supported Coordination. *ACM Transactions on Information Systems, Volume 5, No.2*, pages 115–131, April 1987.

[MK93] Maes, P. and Kozierok, R. Learning Interface Agent. In *Proceedings of the AAAI'93 Conference.* MIT-Press, 1993.

[MLF95] Malone, T. W., Lai, K. Y., and Fry, C. Experiments with Oval: A Radically Tailorable Tool for Cooperative Work. *ACM Transactions on Information Systems, Volume 13, No.2* , pages 177–205, April 1995.

[Mus96] Mustillo, Pardo. Human-Computer Interaction, Lecture 12. Computer Science Department, Concordia University, 1996.

[Nic81] Nickerson, R.S. Why Interactive Computer Systems Are Sometimes not Used by People Who Might Benefit from Them. *International Journal of Man-Machine Studies, 15*, pages 469–481, 1981.

[Nie93] Nielsen, Jakob. *Usability Engineering*, chapter 3. Academic Press, Inc., 1993.

[Nis95] Nissen, Mark. *Intelligent Agents: A Technology and Business Application Analysis.* November 1995.

[Nor81] Norman, Donald A. What is Cognitive Science? In Norman, Donald A., editor, *Perspectives on Cognitive Science.* ABLEX Publishing Corporation, 1981.

[Nor83] Norman, Donald A. Some Observations on Mental Models. In Gentner, Dedre and Stevens, Albert L., editor, *Mental models.* Lawrence Erlbaum Associates, Hillsdale, NJ, 1983.

[Nor86] Norman, Donald A. Design Principles for Human computer Interfaces. In Norman, Donald A. and Draper, Stephen W., editor, *User Centered System Design.* Lawrence Erlbaum Associates, Hillsdale, NJ, 1986.

[Nwa96] Nwana, Hyacinth S. Software Agents: An Overview. *Knowledge Engineering Review, Vol.11, No.3*, pages 1–40, September 1996.

[PM96] Pont, M. J. and Moreale, E. Towards a Practical Methodology for Agent-Oriented Software Engineering with C++ and Java. Technical Report 96–133, Department of Engineering, Leicester University, December 1996.

[RN82] Rumelhart, D.E. and Norman D.A. Simulating A Skilled Typist: A Study of Skilled Cognitive-Motor Performance. pages 1–36, 1982.

[SD79] Sternberg, Robert J. and Detterman, Douglas K. *Human Intelligence : Perspectives on its Theory and Measurement.* Norwood, N.J. : Ablex Pub. Corp., 1979.

[Shn87] Shneiderman, B. *Designing the User Interface: Strategies for Effective Human-Computer Interaction.* Addison-Wesley Publishing Company, 1987.

[SM94] Shardanand, U. and Maes, P. Ringo: A Social Information Filtering System For Recommending Music. Internal Report, Media Laboratory, MIT, May 1994.

[TM96] Terveen, Loren G. and Murray, La Tondra. Helping Users Program Their Personal Agents. In *CHI'96 Conference Proceedings on Human Factors in Computer Systems*, page 355, April 1996.

[Tul83] Tulving, Endel. *Elements of Episodic Memory.* Oxford University Press, 1983. Oxford Psychology Series; No. 2.

[Wil94] Wildfire Communications Inc. Wildfire Homepage. *http://www.wildfire.com*, 1994.

[WS96] Whittaker, Steve and Sidner, Candace. Email Overload: Exploring Personal Information Management of Email. In *CHI'96 Conference Proceedings on Human Factors in Computer Systems*, pages 276–283, April 1996.

[You97] Younes, Roland. Proactive Agent for Call Management in Telecommunication: Criteria Evaluation. Master's thesis, Concordia University, 1997.