165

ADDIS ABABA UNIVERSITY

SCHOOL OF GRADUATE STUDIES

SCHOOL OF INFORMATION STUDIES FOR AFRICA

APPLYING INTERFACE AGENT TECHNOLOGY TO SELECTIVE DISSEMINATION OF INFORMATION (SDI) USER PROFILE MANAGEMENT : THE CASE OF ILRIAlerts

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENT FOR THE DEGREE OF MASTER OF SCIENCE IN INFORMATION SCIENCE

BY

DAWIT YIMAM SEID

May, 1998

ADDIS ABABA UNIVERSITY
SCHOOL OF GRADUATE STUDIES
SCHOOL OF INFORMATION STUDIES FOR AFRICA

APPLYING INTERFACE AGENT TECHNOLOGY TO SELECTIVE DISSEMINATION OF INFORMATION (SDI) USER PROFILE MANAGEMENT: THE CASE OF ILRIALERTS.

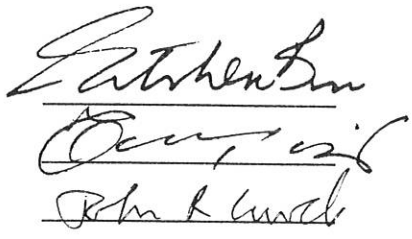
By
Dawit Yimam Seid

Name and Signature of Members of the Examining Board

Ato Getachew Birru, Chairman, Examining Board

Ato Tesfaye Biru, Advisor

Dr. J. Cowell, External Examiner



DEDICATED

To

Elizabeth Endeshaw

ACKNOWLEDGMENT

My greatest thanks go to my advisor, Ato Tesfaye Birru. Apart from bringing the research area to my attention, his keen insight, high standards, and skillful pushes to get me explore had a huge impact both on this research and on my academic development. I also would like to extend my sincerest gratitude to Ato Nega Alemayehu who supplied his invaluable comments and sent me the required articles for this research. I also thank Dr. Juergen Koenneman of GMD (Germany) for his inputs to this research in the forms of comments and references to valuable literature. I would also like to offer my gratitude to ILRI and its staff, particularly Ato Getachew Bulfeta and W/ro Azeb Abraham, for their unreserved and all rounded support with out which this research can not be done at all.

It gives me a great pleasure to extend my thanks to all who supported me during the course of this research. I find it hard not to mention Ato Sisay Fissaha who did all he can to guide me into the jungle of statistical methods and who shared ideas in programming. Thanks is also due to W/ro Ayneabeba Amenu and Frewoini who helped me type the manuscript of this thesis with amazing accuracy. I would also like to thank Ato Solomon Teferra and all other colleagues of mine who provided me with the encouragement that propelled me forward. Thanks also to all the SISA community who have influenced this work in many ways.

Last, but not least, I am grateful to my wife Elizabeth Endeshaw whose care, tolerance and attention made the whole process of this research less difficult than what it would have turned out.

ABSTRACT

SDI services and systems are most important components of information services. However, they are also widely reported to be resource intensive and time-consuming for both the service providers and users. As a result, despite the immense contribution they are proved to provide, they are rarely implemented in developing countries whose information systems and services are mainly characterized by paucity of resources.

In particular, previous studies have repeatedly stated that the creation and maintenance of the user profile is the most problematic component. This thesis research investigated in to the problems of this particular component with the view to apply an advanced Artificial Intelligence technology known as *interface agents* which are a type of software systems generally called *agents*. To this end, the SDI user profile management problems are analyzed taking a case SDI system known as ILRIAlerts of the International Livestock Research Institute (ILRI), Addis Ababa. This has enabled to discover numerous ways in which agent technology could bring about improvements on SDI user profile management. Furthermore, the review of the literature has shown that the technology is already being employed in systems which are similar to, or enhancements of, SDI systems like those commonly known as filtering systems that are developed for Internet resources.

Drawing on these findings, an agent-oriented modelling of the user profile management process has been made and some of the basic methods and procedures suggested in this model have been demonstrated by developing a prototype agent. Using this agent, some small scale testing of the performance of the suggested methods have been made taking few test data from ILRIAlerts users. The prototype development and tests have indicated the possibility of employing agent methods to SDI user profile management.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1. Background

As Ryan (1991) puts it, the biggest problem with information for most people is that there is too much of it and it is extremely difficult and time consuming to get hold of relevant ones. This has resulted in the phenomenon called "*information overload*"(Hiltz and Turoff, 1985). Information overload (earlier of printed information and recently electronic information) has today become a distinguishing feature of the information world. As a result, situations where there is relevant information in a certain framework but is not utilized due to absence of systems to trace and bring it to the attention of the right user have become common.

In particular to developing countries, it has been repeatedly stated that most often few or no systematic programs for the collection, analysis and dissemination of available information exist. As a result, the potential users remain unaware of the existence of relevant information, and the quality of their decision may be the poorer for it (Boadi, 1987). A recent document of UNECA (1996) also tells us that the situation is all the same today. One major reason cited for this is the fact that most of the information service providers/systems in the developing world are usually under-staffed, unorganized, unequipped and suffering from severe shortage of financial and other resources (Nganga 1995).

To attack these problems, various approaches and systems have been suggested (Saracevic and Wood1981). Just to mention a few of the solutions presently in wide use:

- More active involvement of information systems with user demand; studying the user more closely; evaluating services/products in terms of their use
- Tailoring the variety of information services/products to the variety of user levels and demands; filtering/selecting relevant, and only relevant, sources for their users
- Organizing information resources in ways that are more appropriate for socio-economic developments and presenting them in predigested, problem-oriented forms.

Selective Dissemination of Information (SDI) service is one of the prominent services designed along these lines. It is, in fact, the most widely implemented and utilized current awareness service(Ravi 1994; ILRI, 1995). SDI particularly refers to a computer-assisted subscription-based personalized service that notifies a user about new literature and data in accordance with his/her statement of interest (Leggate, 1975; Rowley, 1993). Leggate (1975) states the objective of SDI as: reduction in the user effort in selecting from a mass of information sources, while Torkelson(1973) views it as a method aiming at preventing information overload.

The most common types of SDI systems are those which are based on computer systems and databases, which are sometimes known as *Automatic SDI* or *Mechanized SDI services* (Leggate, 1975; Rowley, 1993). Technically speaking, these are services that enable users to receive automatically literature search updates from chosen bibliographic or full-text

databases on a continuous basis at a certain frequency. These systems work by saving in a file subject search strategies (which are called *User profiles* or *standing orders*) done on the most appropriate database(s) (which may be on-line or on CD-ROM). Every time the appropriate/applicable database is updated, the stored profile is run against its new segment. Search results are then passed onto the client, either on diskette, via e-mail or as a computer printout. The client, in return, forwards his/her feedback on the appropriateness of the results for consideration in refining his/her profile.

1.2. Statement of the Problem

According to many authors (for example, Hamilton 1995; Rowley 1993; Weaving 1991; Leggate, 1975), the user-interest profile is the core of all SDI systems. In fact, the whole functionality of an SDI system depends by and large upon the quality of the profile since it is the basis for filtering the information likely to be relevant to the user. As such, the major difficulties of SDI implementation, maintenance and management are reported to relate to this component.

While the initial construction of user profiles that may be in the hands of end-user, a database specialist or other intermediary is a difficult task in the existing conventional SDI system, maintaining and managing them have also proved to be as challenging. Normally, an SDI profile can be refined over a great number of search runs since the nature of existing SDI systems demands this and since it is common for the intermediary who constructs the profiles to fail to understand or misunderstand the user's statement of interest(Rowley, 1993). Hence, a user has to supply his feedback for each run to the database expert, consult with him/her, modify his profile, etc. In other words, the user

profile has to be refined based on feedback on the performance of earlier profiles. Besides, when the need to update the user profile arises due to changes in the interest of the user, the previous cycle is repeated once again. All these make the system demanding too much effort on the part of the users which doesn't seem to go along with the fundamental purpose of SDI systems as stated by Leggate : "the reduction of user effort in selecting from the mass of information sources". This situation may prove more troublesome in cases where the SDI service provider and the user are geographically separated, that is when the user uses the SDI service through a network (e-mail delivery of search results) as in Butler (1995).

Other problems that arise in this connection are the psychological and behavioral problems, that manifest particularly in cases of systems that serve researchers, academicians and other learned user groups. It is not uncommon for such users to be reluctant to accept support from intermediaries in the construction and optimization of their profiles.

Another feature of user profiles that makes them very difficult to maintain is the manner they represent users' interests. In the existing SDI technology, interests of the user should be reduced to keywords and Boolean-based search languages which are in turn used by the search engine to search the databases. In deed, what are called user profiles in SDI are simply saved search strategies which are executed at a certain interval. This is a very critical limitation of these systems which, even in the most ideal cases, leaves a big gap between users actual needs and those represented by the systems. Hence, it is important that these profiles are refined on continuous basis by tracking user's information needs.

While the aforementioned shortcomings persisted to cripple SDI systems, in parallel, magnificent achievements are being made in the field of Artificial Intelligence, particularly *software agent technology* in the area of user interest modeling and end-user assistance as it relates to the Internet.

Basically, a software agent, or agent simply, is a component of software and/or hardware which is capable of acting exactingly in order to accomplish tasks on behalf of its user(Nwana, 1996). It is a software program that assists and performs tasks for its user within a computing environment. Hence, taking this general definition, we can include the majority of software programs written in the past forty or so years under the umbrella of agents. But as we descend deeper in to the concept of agency, we can see that there are distinct characteristics that collectively constitute a software agent. Software agents are differentiated from other applications by their added dimensions of *autonomy*, *inductive learning*, *mobility* and *the ability to interact independent of its user's presence*. These involve the capability to process information from external environments given a set of knowledge, attitudes, and beliefs of the user which are understood by the agent(Heilmann, *et al,* 1995).

Agent technology bases itself on the novel ideas and techniques of Artificial Intelligence so as to resolve various problems major among them is that of user-interest based information transfer/access in the World Wide Web environment (Maes, 1994). To date, several prototype and full-fledged agents have been developed that provide personalized assistance with meeting scheduling, electronic mail handing, electronic news filtering, selection of entertainment, personalized Net surfing, advertising, web-commerce, web-shopping, employment, etc.(Maes, 1994).

This study is conceived based on the observation that the problems dealt with agent technology seem to have fundamental similarity with some of the aforementioned problems of SDI profile management. Of particular interest are the application of agents in the areas of:

- filtering of relevant information based on stored information about user needs in the system;
- repetitive interaction with user and feedback collection and analysis which together allow the formation of models of user behavior;
- personalized services that vary for different users and that vary for a user as time goes by; and
- applicability of achievements, skills and best approaches gained with a use or at an instance to another user or instance.

As a matter of fact, as Wyle (1996) also noted, SDI systems, on their own, can be considered a kind of agent. In particular, one category of agents in common use that are known as Information Retrieval agents can be argued to have similar ultimate purpose to SDI systems, except that SDI systems deal with printed information overload while the IR agents deal with electronic information overload.

On these grounds, the current study tried to investigate into agent technology and SDI user profile management processes and uncover how the former can be applied to the latter. Specifically, it has tried to look into the following questions:

1. How do real-world SDI systems in general and user profile management processes in particular function ?
2. Is it possible to model the user profile management processes to allow the design and development of agent(s) that reduce the burden on human intermediaries and users ?
3. How, in particular, could agents be used to effectively provide SDI services using wide area networks like Internet in cases where users are geographically remote ?
4. What is the best way of refining (or optimizing) SDI user profiles ? What is the role of new artificial intelligence and information retrieval techniques and algorithms in this process ? How could these techniques and algorithms be employed to build agents for SDI user profile management ?

1.3. Justification of the Study

Particularly in many African cases, due to the chronic resource limitations, establishing the communication channel that enables the refinement and continuous updating of user profiles is obviously difficult to implement. Even in cases where such channels are created and maintained, for instance in cases of International Livestock Research Institute (ILRI) and Pan African Development Information System (PADIS), it has been learnt that the process is very time consuming for both users and service providers and, at times, very discouraging for most users. Besides, sufficient database/information experts to assist the users in such an undertaking are hardly available. The reality is, most information systems in developing countries suffer from severe shortage of information personnel (Nganga, 1995). In fact, most systems are run by para-professionals who are of little help in such expert tasks like assisting a user in optimizing and updating his profile. According to

reports of information manpower development (Birungi, 1995; Mugasha, 1995; Nganga, 1995), there is no hope of getting such information people abundantly in the near future.

Viewed in this context, it is easy to see the value of automated systems that could, to any possible extent, provide assistance to users who presently do not have access to expert intermediaries. It follows that all possible technologies need to be explored to develop such systems. Its success in many similar information retrieval problems makes agent technology a reasonable candidate for this purpose.

The following are some of the possibilities that can be envisaged to be presented by agent technology in this respect:

- Agent approach could enhance the retrieval performance of the systems by narrowing the gap between users actual needs and user interest profiles as it enables the representation of much more information about the user.
- It could enable autonomous updating of the profile based on the clues it gathers regarding user interest changes. This, in turn, saves user time and effort and allows him/her to make use of the services with little or no support from intermediaries. This can be made possible by utilizing agent technology to develop easy-to-use interface to the SDI system as well as intelligent assistance in the construction and utilization of his/her profile.
- It could enable the end user to take on an important role in developing his or her own SDI profile, run his/her profile, and make modifications with very little effort.

reports of information manpower development (Birungi, 1995; Mugasha, 1995; Nganga, 1995), there is no hope of getting such information people abundantly in the near future.

Viewed in this context, it is easy to see the value of automated systems that could, to any possible extent, provide assistance to users who presently do not have access to expert intermediaries. It follows that all possible technologies need to be explored to develop such systems. Its success in many similar information retrieval problems makes agent technology a reasonable candidate for this purpose.

The following are some of the possibilities that can be envisaged to be presented by agent technology in this respect:

- Agent approach could enhance the retrieval performance of the systems by narrowing the gap between users actual needs and user interest profiles as it enables the representation of much more information about the user.
- It could enable autonomous updating of the profile based on the clues it gathers regarding user interest changes. This, in turn, saves user time and effort and allows him/her to make use of the services with little or no support from intermediaries. This can be made possible by utilizing agent technology to develop easy-to-use interface to the SDI system as well as intelligent assistance in the construction and utilization of his/her profile.
- It could enable the end user to take on an important role in developing his or her own SDI profile, run his/her profile, and make modifications with very little effort.

1.4. Objectives of the Study

1.4.1. General Objectives

The general objective of this study is to investigate into approaches and techniques of agent technology with the aim of testing its applicability to enhance the user profile management of SDI systems by taking, as a case study, a real world SDI service namely ILRIAlerts, maintained by ILRI (International Livestock Research Institute).

1.4.2. Specific objectives

With the aim of achieving the general objective of this study, the following specific objectives were drawn:

1. Studying the domain of human intermediary based SDI users' interest profiling and investigating structures/patterns which can allow task delegation to and learning by computer.
2. Study the user-related knowledge needed, their sources and representation required to assist the user in the optimization and updating of his profile and investigate into their learning-interface agent-based implementation.
3. Outline a model of the SDI user profile management process so as to allow agent-based solutions.
4. Identify the particular artificial intelligence and information retrieval techniques and algorithms that can be employed to develop agents that address SDI user profiling problems

5. Test the model by developing a prototype agent based on the findings of the study.

1.5. METHODOLOGY

For the successful completion of this study, the following methods were employed:

1.5.1. Review of Related Literature

Related published(both hard and soft copy) literature have been reviewed for the purpose of identifying and studying both agent technology and SDI techniques. In particular, literature dealing with interface agent development, knowledge representation, user modeling machine-learning algorithms and vector-space model have been reviewed.

Particularly for literature on agent technology and recent IR research, this study has mainly relied on the Internet. Due to the limited access available, an exhaustive survey of these sources was impossible. As a matter of fact, the lack of literature and other resources for this study were perceived at the early stages and, although it did not materialize, a visit to the German National Research Center for Information Technology(GMD) was planned.

1.5.2. Data Collection Methods

A number of survey visits were made to ILRI Information Service and various data collection methods including interviews, observation and review of documents were used to gather information on user profile management process in ILRIAlerts. Interviews have

been conducted with ILRI Information Services staff and selected users. Close observation of how ILRIAlerts function has also been made for a number of days.

1.5.3. Programming techniques

Exploratory programming approach using Agent Building Environment (ABE), a toolkit provided by IBM, has been used to develop the prototype agent. The major reasons for the selection of this toolkit are its free availability and the provision of facilities to embed an intelligent agent to a variety of applications and software environments (IBM, 1997).

For coding the required agent components, Microsoft Visual C++ Version 5.0 is used. This programming language is selected because it supports object-orientation which is the *de facto* approach in agent programming. In addition, the fact that programs developed using it can be easily integrated with ABE is taken in to account in the selection of this programming language.

Some algorithms, like stemming algorithms and text-based learning algorithms, have also been considered for modeling and implementation purposes. Frakes (1992) implementation of the Porter Stemming algorithm has been used for stemming and Fox's (1991) Stop list filter finite state machine generator has been used for stop word removal. The Rocchio feedback-based learning algorithm(Rocchio, 1971) has also been studied for feedback based user profile optimization.

1.5.4. Testing techniques

The suggested agent-oriented model has been demonstrated by developing a prototype agent that implemented the basic development options in the model. Then the prototype is tested using a controlled experiment approach because the time available did not allow testing using actual users. A reasonably representative testing documents have been taken from ILRIAlerts users and user interaction with the prototype has been simulated. Besides, the terms and concepts generated by the prototype have been analyzed using statistical methods.

1.6. Scope and Limitation of the Study

This study has tried to carry out three major activities. First it tried to establish what aspects of SDI user profile management are amenable for agent application. Second, it attempted to model and describe how agents can be applied; and third it tried to demonstrate this model by constructing and testing a prototype. Due to the limited time available to the study, the focus of the second and third activities was solely on the client side, this means the SDI server side is not considered. This was done not because it is believed that the two should be addressed separately, but time did not allow the consideration of the user profile management process as a whole.

Furthermore, while two agents have been proposed on the client-side, it was possible to prototype only one. This is so due to limited time and the extended period needed to learn and use the Agent Building environment.

In modeling the user profile management process, understanding on the user-intermediary interaction was gained only based on the information gathered from the few intermediaries in ILRIAlerts and from the literature as drawing on a primary study on this was beyond the scope of the study.

Another limitation of this study was the scarce availability of recent literature as well as prior experience around on the area. Although Internet searches have produced valuable literature, still the limited connect time did not allow exhaustive searches.

1.7. Organisation of the Study

This thesis is divided into six chapters. In the second chapter the principles and techniques of SDI systems are discussed with particular emphasis to user profile optimization and updating. This chapter also presents a brief description of the case, that is ILRIAlerts, followed by a review of current Internet based automatic SDI systems and related research to improve profile management and performance. Chapter 3 is a survey of the concepts and research on agent technology in general and Interface agent technology in particular. It focuses on areas amenable for agent applications and approaches that are employed in agent development. In chapter 4 a description of the agent-oriented model suggested by this research work is presented emphasizing on the client-side system and its constituent agents. Chapter 5 presents the prototype developed to test the agent-oriented model discussed in chapter 4 and the tests conducted using it. Finally, Chapter 6 presents the conclusions drawn and suggests directions for further research on applying agent technology to SDI systems.

CHAPTER 2

SDI AND FILTERING SYSTEMS

2.1. SDI Systems: An Overview

2.1.1. Definition

Selective Dissemination of Information (SDI), also known as "*Personal notification service*" or "*Personalized alerting service*", is one of the current awareness services[1] where materials are "selected" from the latest databases, and the "information" is "disseminated" to the user.

Pao(1989) defines SDI as follows:

> *"a service whose primary function is to alert and notify its clients of potentially useful new information on an individualized basis. It produces a continuous and dependable service which often extends to the supply of actual documents or abstracts which have been screened and filtered by the systems staff"*

Hence, if the system of notification is tailored to individual needs, rather than sending the same information to a large group, this can be referred to as SDI service. As such, basic to SDI services is the idea of "*personalization*" and "*selection*". The selection of information in SDI services is made possible through a record of personal information needs, called *user interest profile*, maintained by the systems. And it is this unique user profile, commonly maintained in computerized form and readily modified as information needs of

[1] There are lots of literature that use "Current awareness service" interchangeably with SDI. But here SDI and current awareness services are distinguished with the latter considered as a broader concept consisting of both personalized or non-personalized, selective or non-selective services.

user changes, that distinguishes SDI from other forms of current awareness services like information bulletin and newsletter.

Actually, in SDI, computer-based profile is prepared and, periodically, matched with the new segment of the database(s) by the computer. When a record in the database matches a profile, it is recorded as a 'hit' for that profile. Then details of the hits corresponding to each profile are output by the system and sent to the profile-holder.

2.1.2. Origin and Development

The idea of mechanized SDI services was introduced by Luhn at IBM in his article "*A Business Intelligence System*" published in 1958, and a subsequent article in 1961. In his 1961 article, Luhn, as quoted by Mauerhoff (1974), described SDI as follows:

> *"a service within an organization which concerns itself with the 'machine-assisted' channeling of new items of information, from whatever source, to those points within the organization where the probability of usefulness, in connection with current work or interest is high."*

He also argued that this method could improve the communication of scientific information by the selective dissemination of information assisted by machines (Luhn, 1958, as quoted by Pao(1989)). After Luhn's suggestion, SDI received increasing attention throughout the '60s and '70s as a method of keeping users informed about new advances that were recently published (Mondschein, 1990). As Leggate(1975) notes, SDI is one of the most successful services developed in the sixties.

2.1.3. Why SDI ?

SDI services have the objective of updating the receiver regularly and currently from a range of articles and documents otherwise beyond his normal search capability and willingness to scan. As such they are a quick and effortless way to stay current on important topics. Mauerhoff (1974) notes that while SDI was not intended to be the ultimate solution in the scientific communication process, it was presented as promising immediate reduction in intellectual effort and time. Pao(1989) goes to the extent of arguing that SDI enables to bypass the need to verbalize information need as it "meets" information needs of users without explicit requests from the user.

Whitehall(1997) argues that the value of SDI - where information is actively disseminated to users - is that the user saves time finding and scanning current published material or database. Further, by SDI, the information service may even be doing something for users that they may never do for themselves. This is because many professional people cannot find the time habitually to look for new information in their area. In some cases they do not want to spend time in libraries or on computer networks keeping up to date when they could be spending this time on what they regard as their "real" job.

In discussing the purposes of SDI systems, it is important to note their differences from retrospective search services. The major differences that exist between retrospective search service and SDI (IDRC, 1985) are summarized in the table below:

Table 2.1: SDI versus Retrospective Searches

SDI	Retrospective search
. regular and automatic	. on demand
. only up-to-date information	. includes older items
. output in small amount	. generally in big quantities
. broader search strategies hence wider coverage	. Search strategies narrow to avoid retrieval of large outputs
. batch processing	. interactive

2.1.4. Structure of SDI Systems

Based on their components, we can divide SDI systems into two categories, namely Networked and Non-networked SDI systems. In the non-networked systems, the service is provided from a stand-alone computer in the library/information center. Such SDI system is normally operated by an intermediary and users don't directly access the system. Rather, the intermediary gathers user's subject interest either through interview or written statements, constructs the profile and delivers the notifications in the form of computer printouts or diskette.

The other types of SDI configuration, namely the network-based ones, are of different types. Basically, in this case, the computer from which SDI services are provided and the computer of the user are somehow networked in some form - be it local or wide area network. The earlier version of this category are the SDI services provided by on-line database hosts.

2.1.5. Service Procedures

The basic operations of any SDI service include the construction of user profiles, matching of profiles with databases and notification and feedback processing. While the user profiling operation is discussed in detail in the following section (2.2) as it is the major

concern in this study, the matching and notification processes are briefly described in the following paragraphs.

Matching : Once the user interest profile is developed (as will be discussed in section 2.2), it is matched with one or more pre-defined database(s). When a record that matches the profile is found, it is said to be a 'hit' for that profile. For each matching, a list of hits would be delivered to the user.

Basically, there are two types of search logic leading to varying criteria for a hit to occur: *Boolean statements* and *weighted search*. In the Boolean statements, even a single occurrence of the terms in the profile results in a hit while in the later, a hit requires that the weighted proportion of codes in the document pattern which matched those in the profile exceed a predetermined threshold (Rowley, 1993). In other words, a weight that expresses its relative importance is somehow attached to a term and, not only should it be present in the database for a hit to occur, but its importance in relation to the document should also be assessed and should have reached a certain level before a hit is recorded. Luhn's proposed SDI system, for instance, used weighted searches.

Notification and feedback : This refers to the delivery of those records (i.e. citations or abstracts) retrieved on the basis of the user profile to the user either on diskette, via e-mail or as a computer printout. Usually, notifications are delivered along with *feedback forms* which are filled by the user and returned to the SDI system.

2.2. User Profiling

2.2.1. Information on Users

The information kept on users in SDI systems can be categorized into three, namely *the user-interest profile, the user identification data(* which are kept by the computer system) and *other related information* which is often gathered and accumulated by database expert (intermediary) interacting with the user.

The User-interest Profile

According to Kemp(1979), the user-interest profile is "an expression of the current interests and needs of a client, stated in a way which enables it to be used to identify the records of documents which will meet those interests and needs." Pao (1989) describes it as " a listing of keywords joined by Boolean operators characterizing a user's (topical) interest."

Technically, the user-interest profile refers to the terms and Boolean search statements that are saved in the system to be regularly used for searching the document database(s). In the literature, we find various ways of referring to these saved terms and statements. Apart from "user-interest profile", phrases like "user profile" "user subject-interest profile" "search profile"(e.g. Leggate, 1978), "interest Profile", "SDI profile"(Rowley, 1993) are frequently used. It is also common to find the term "profile" used in many documents without any qualifying adjective.

Contrasting with retrospective search expressions, Rowley (1993) identified the following distinguishing features of user profiles.

1. They are likely to be less specific (as only a short period of time is covered).
2. They are possibly more likely to be regularly matched against several databases, rather than merely searching one or two.
3. They can be refined over a greater number of runs (i.e. by an iterative process of running the profile, reviewing output, and looking for ways to remove false drops from future output, such as the use of subject categories suggested by Sprague and Freudenreich(1978).

Content wise, according to Leggate(1975), anything that is machine-readable can be considered as index term for search profile - word or word fragment from a title or abstract, keyword, subject index term, journal coden and author's name, parts of the bibliography, and so on, though a processing center may decree that only certain data elements are searchable (and hence included in the search profile).

In addition to the subject terms, a profile is often supplemented with other user requirements, including the following (Pao, 1989; Mauerhoff, 1974):

- The names of key authors whose works are essential reading or colleagues pursuing related research.
- Certain sponsoring agencies and journal titles in the fields from which important papers are often found for the user (i.e. key journals and organizations noted for their high research activity). Information on sources identified by the user as must inclusions is also maintained.

- Key papers or key figures whose works are often cited in the user's special research area.
- Significant words and word associations appearing in the title of tables of contents pages, footnotes and quotes from key papers.

Besides, a list of key references may be used to retrieve all subsequent current papers that cite any one or more of the papers in the starter bibliography.

Normally user profiles are divided into two, based on whether the service aims at a single user or a group of users with the same interest. In the cases where the service aims at individual users, what is called "*personalized profile*" (or "*individual profile*") is designed for each user that documents his interest and other related information like his work, discipline, aims, etc. In the cases where the service aims at a group, a single profile is used for more than one client. According to Kemp(1979), such profiles, in turn, fall into two categories: '*standard profiles*'(pre-set profiles covering a number of standard subjects or topics) and '*group profiles*' (profiles of a group of users, for e.g. a team of researchers working together on different aspects of the same problem).

In addition to the idea of having a single profile for an individual or a group, there are also situations where more than one profile, called *multiple profiles,* are established for a single user for various reasons including the need to use different databases which need different profiles to be effective, or because such profiles result in convenience to the user to receive different notifications for the different interests, even if, as may happen, they overlap slightly.

The User Data

The user identification data needed by the SDI system is maintained either along with the user-profiles or as a separate databases. The contents of this component/database vary from system to system. But, usually it would contain such static data as name of the user, or if a group, name of project leader; user's number, if any; address like telephone number, postal address (e-mail address in case of network-based SDI); preferred format of delivery of search results; subscriber's profession and other work-related details.

Besides, additional information like the interval the system should wait before sending the subscriber a new set of messages, the default number of messages the subscriber wishes to receive per delivery interval, specific databases on which the search should be made or should not be made are commonplace. Most SDI systems also record per user a number of statistical data including the date on which the last time records were sent to the user, the total number of different bibliographic records that were found, the total number of sent messages, etc.

Other User Related Information

In most cases, an SDI system is implicitly or explicitly supported by lots of information that is either consciously or unconsciously gathered by the intermediary interacting with the user. These information are mainly gathered during the repetitive interviews and other interactions made with the user (profile-holder) although consulting other user-related sources and documents can also be used. Expectedly, the amount of such information

varies from system to system. Most successful systems accumulate as much information as possible in the following aspects:

- User's attitude toward recall (coverage) and precision (relevance)
- Objectives and motivations of the user like the research being conducted by the user, the administrative duties or committees on which he or she sits.
- Personal background of the inquirer, for example his/her status and level of training , the relationship between his/her needs to what he/she already knows, his background in the use of information sources or certain information system/center
- The amount of information and the level of information needed.
- Information extracted from reports of projects (both ongoing and completed) by the user or of future plans.
- Information gathered by attending meetings where the information worker can hear about what is going on in the organization
- Information on the activities in the organization/of the user
- Information on the materials, methods and processes used in the user's work.
- Whether the user is looking for technical or commercial information
- Whether a theoretical or practical approach to the subject is needed.

2.2.2. The Construction Process

While the success of an SDI system is contingent on the accuracy of the user profile, the user profile in turn is largely dependent on the work performed on the profile construction and on profile revision (Sprague and Freudenreich, 1978). For this reason, the process of

profile construction and maintenance have become the main focus of almost all literature dealing with SDI.

In general, we observe that basically profile construction involves the following four steps:

- Gathering initial statement of user interest
- Development of tentative/provisional profile
- Iterative testing and refining of provisional profile
- Write up of the usable profile

In the literature, we find many other alternative methods of constructing user profile. Pao (1989) cites a profile construction method for users conducting research (and publish) where, instead of establishing the profile from a user's verbalization of interest, another type of profile may be indirectly established based on the user's own publications. It is suggested that one's own writings may be used in profile construction in the following two ways:

1. Index terms assigned to the writings may be used to construct a tentative profile subject to revision by the user;
2. The publications of users themselves can be used to retrieve other publications similar in subject content. This approach is based on the premise that citing papers are relevant to the cited publications.

The other approach to profile construction is the one based on standard subjects and topics in a certain field of study. This method is usually employed in the construction of group (or standard) SDI profiles. As well as being cheaper to generate, standard SDI profile is much

easier to use, provided that an appropriate subject is available. However, such profiles are less specific and hence widen the field of which the user is kept aware (Rowley, 1993).

Another alternative is what is called *on-line profile construction,* and is done in on-line environments. This may take to form of *vendor-supplied* or *saved* or *re-keyed*, depending on the facilities offered by the service.

2.2.3 Feedback Processing /Evaluation

While the SDI service continues to be provided, it is important that feedback on each notification is received and analyzed. This component of SDI system evaluates the performance of the SDI service in terms of user-reaction on the retrieved materials. On the basis of such evaluation, the individual interest profile may be revised and improved for subsequent retrieval. Feedbacks are also the most important means of constantly tracking and keeping up-to-date with the changes in the needs of the user.

As pointed out by Pao (1989), such feedback may take many forms. The reactions to each notification may be either in written or verbal form. In most cases, each notification is accompanied by what is called "*feedback form*" or "*control-and-follow up*" on which the user can indicate whether the document really interests him/her, whether he/she wants a copy or has one already, or why it is of no interest to him (Guinchat and Menou, 1983). In other cases, the feedback component is built-in the notifications themselves. The user returns a copy of the notification filling in his feedback.

Feedback is said to be especially crucial if the first few retrieval transactions produce poor results(Pao, 1989). Generally, feedback is an important element for the success of any SDI service, but requires full cooperation by the clientele. The user participation, in turn is dependent on whether the feedback requires the least amount of time and effort on the part of the user.

To this end, it is often recommended that initiating for feedback, and consequently updating of profile, should be taken by the SDI system very seriously. Monitoring is also required to make sure the user is providing the necessary feedback. Mondschein(1990) discovered that when an SDI service begins retrieving irrelevant information, the user usually requests the service be curtailed, or he or she simply stops reviewing the printouts. To help resolve this major problem, the following steps are recommended:

- An effort by the staff of the information center to keep track of SDI profile updating.
- Regular follow-up with all users who have recently obtained new SDI profiles to ensure that the information they receive is meeting their needs.
- Keeping records on requests for reprints (or original copies) of full papers based on the SDI printout.

2.2.4. Revision/ Updating Process

As Kemp(1979) says, "profile compilation is not a capital, once-for-all effort". Although there are user profiles that tend to remain relatively stable in cases where the holders continue to pursue the same research over long period of time(Pao, 1989; Mauerhoff, 1974), still there are lots of instances where the profiles need regular, if not continuous,

revision. Mauerhoff (1974) and Kemp (1979) identify the following instances where profile revision may be required.

1. Because of changing needs of the client (for various reasons including a redirection of interest feedback, relocation, and perhaps promotion).
2. Because of changes in the subject field, and in its terminology.
3. Because of changes in the system, especially in the thesaurus.

Pao(1989) adds some more reasons like redirection of the organization's emphasis or product lines, new mergers and acquisition of the parent organization, relocation, or new areas of exploration. He also suggests that if the users become more knowledgeable about the potentials and limitations of the SDI system, their demands may also change over time.

According to Kemp (1979), changes to profiles can be of two kinds.

1. Changes intended to increase recall, so that the user receives notifications of items of interest which he/she has not previously received. This requires the insertion into the profile of additional terms with OR logic.
2. Changes to increase precision, so that the user no longer receives notifications which do not now interest him/her. This may involve the simple omission of terms, the use of and logic instead of OR logic for one or more terms, and the use of terms with NOT logic for the exclusion of terms related to developments in which the client has no interest.

It has been widely noted (for example Whitehall, 1986; Rowley, 1983; Whitehall, 1997) that although their interests can change quite frequently, users do not normally fasten to report their new interests than to ask for an interest to be dropped from the profile. All these indicate the need for the SDI service provider to take the responsibility of initiating profile update rather than waiting for the user to make a request for same.

2.3. Approaches to Automation

2.3.1. Early Suggestions and Motivations

So as to resolve the problems of conventional SDI systems, various suggestions and researches to develop automated systems that assist in profile construction and refinement were made from early on. In particular, earlier researches were basically motivated by at least three main reasons.

1) The fact that profile construction using human intermediaries has proved to be very expensive, time-consuming and demanding too much effort on the part of the user.
2) The incidents of profile ineffectiveness as a result of communication failure between user and intermediary (Leggate, 1975) and users' lack of interest to maintain continuous discourse with intermediary
3) As a consequence of 2, the idea of a user constructed and refined profile, since the user is believed to be the best judge of what materials he is seeking and what terms he should use (Barker, *et al.*, 1972). Dammers quoted by Whitehall(1986) further explains that using an intermediary between the user and the literature is a system of doubtful merit.

One solution for these that is experimented by some was training users on the intricacies of search strategies and thesaurus utilization. This was believed to enable users to help themselves or demand very limited assistance from an intermediary. However, this approach was widely criticized on the following grounds:

1. The very idea of SDI (as also advocated by Luhn) is that users should exert the minimum possible effort.
2. It is unlikely that users would (like to) acquire the knowledge and experience of the system that would enable them produce effective and efficient profiles.

As an alternative, and especially with the coming of SDI services by on-line hosts and large database suppliers, the utilization of standard profiles was also adopted despite its weaknesses already discussed in the foregoing.

It is with in this milieu that the idea of developing programs that provide assistance, particularly to the end-uses, in the construction and refining of profiles surfaced (Barker et al, 1972). However, until recently, not much success was registered in this regard apart from the adoption of some simpler-to-use front ends to on-line databases (Mondschein, 1990).

2.3.2. Current Trends and Developments

Among the relatively recent developments that showed the way for realizing the idea of automated assistants to end-user search construction was the adoption of artificial

intelligence techniques to automate some aspect of the intermediary function resulting in what are called *knowledge-based IR* (also called *expert* or *intelligent IR) intermediaries/systems*. The objective of research in this line was the development of expert systems that have a domain knowledge (usually in the forms of a thesaurus, frame like representation of domain knowledge or dictionary mapping) and knowledge of search strategy development techniques (in the form of rules). Various prototype expert systems with these capabilities were tested in the IR environment. A good account on such systems may be found in Gauch (1992) and Ellis (1990).

Although some of these systems were considerably successful in converting (mapping) users unstructured requests to search statements used by the database systems, they rarely tackled the issues peculiar to SDI, like *personalization*, *long-lasting interest representation*, *continuous alerting*, etc. This is probably due to the fact that most of them were framed in the retrospective IR environments.

The foremost shortcoming of the attempts in addressing the above SDI related issues is the fact that they incorporated very rudimentary, if any, user modeling mechanisms despite the early recognition of the relevance of such models to IR (e.g. Daniels, 1986). For instance, one of the systems, I^3R, incorporated user modeling in the form of stereotypes containing information on previous sessions and relevance feedback while PLEXUS constructed a temporary user model based on responses to six questions about users experience in gardening (which is the domain of PLEXUS) Gauch (1992).

An important study that tried to fill this gap is the MONSTRAT project, by researchers from the City University, London and the Free University, Berlin, which was based on the

idea that any attempt to devise a computer program intended to model the functions of the search intermediary should be preceded by a detailed knowledge of what takes place in the interaction of the search intermediary and the user. After analyzing the interaction between users and human intermediaries, the project recommended a distributed expert based intermediary incorporating a user model that includes user's goals, status and category, background, extent of familiarity with IR systems and state of knowledge of the field of inquiry. While the value of such models was clear, implementing them was found to be difficult which probably was due to the limitations of the expert systems techniques available at the time. In this connection, some earlier researchers have even gone to the extent of expressing serious doubts on the applicability of AI techniques to IR. The following statement by Brooks (1987) as quoted by Ellis(1990) indicates this observation:

> *"For several reasons, IR does not seem to be an ideal problem domain for an expert system application. It is a domain that is neither well bounded nor narrow nor homogeneous .... it seems that it is not feasible at present, to think of building an expert system that carries out intelligent retrieval".*

Internet, Information Filtering and the push model

The recent dramatic success of the Internet and the world wide web, and the resulting electronic information overload, have initiated a fresh look at the issues of "selection" "personalization" and "notification" giving rise to what is called *information filtering* (also called *text filtering*). Compared to prior efforts, this has turned out to be a more dynamic endeavor absorbing interests from various fields/disciplines.

Information filtering (IF) as understood today can be considered as a modern extension of SDI (Foltz and Dumais, 1992; Oard, 1997). *Automatic Information filtering systems* are defined as systems with the goal of automatically sorting through large volumes of

idea that any attempt to devise a computer program intended to model the functions of the search intermediary should be preceded by a detailed knowledge of what takes place in the interaction of the search intermediary and the user. After analyzing the interaction between users and human intermediaries, the project recommended a distributed expert based intermediary incorporating a user model that includes user's goals, status and category, background, extent of familiarity with IR systems and state of knowledge of the field of inquiry. While the value of such models was clear, implementing them was found to be difficult which probably was due to the limitations of the expert systems techniques available at the time. In this connection, some earlier researchers have even gone to the extent of expressing serious doubts on the applicability of AI techniques to IR. The following statement by Brooks (1987) as quoted by Ellis(1990) indicates this observation:

> *"For several reasons, IR does not seem to be an ideal problem domain for an expert system application. It is a domain that is neither well bounded nor narrow nor homogeneous .... it seems that it is not feasible at present, to think of building an expert system that carries out intelligent retrieval".*

Internet, Information Filtering and the push model

The recent dramatic success of the Internet and the world wide web, and the resulting electronic information overload, have initiated a fresh look at the issues of "selection" "personalization" and "notification" giving rise to what is called *information filtering* (also called *text filtering*). Compared to prior efforts, this has turned out to be a more dynamic endeavor absorbing interests from various fields/disciplines.

Information filtering (IF) as understood today can be considered as a modern extension of SDI (Foltz and Dumais, 1992; Oard, 1997). *Automatic Information filtering systems* are defined as systems with the goal of automatically sorting through large volumes of

dynamically generated information to satisfy a relatively stable and specific information need, as opposed to IR systems dealing with relatively static databases and short-term information needs (Oard, 1995). Of particular interest in this respect are what are called *Personalized information filtering systems*, that are described as providing information that match user needs in a consistent and timely manner as well as accommodating charges in user needs and adapt to those changes (Sheth, 1994). There is a lot of research going on such systems that includes investigation of the best ways of developing user models that describe the multifaceted and complicated interests of a particular user.

One aspect explored for automatic profile generation is the traditional practice in which users are required to provide few relevant materials to serve as input to profile construction. Yochun(1996) describes a prototype system developed with the Logicon Message Dissemination System (LMDS) that generates profile automatically from sets of relevant documents provided by a user, and that assigns relevance scores to the documents selected by these profiles. He tested, in the Fourth Text REtrieval Conference (TREC-4), this profile generator developed using probabilistic scoring algorithm that chooses terms and assigns a weight to each, based on its frequency of occurrence in the set of documents provided by the user, and on its frequency of occurrence in a large representative database of documents (provided for TREC-4 participants).

Foltz and Dumais(1992) compared the performance of profiles constructed using sets of keywords provided by users and using relevance feedback of users about previous abstracts in selecting Technical Memos that meet people's technical interests. They used two retrieval methods, namely keyword matching and Latent Semantic Indexing (LSI) and

found that all methods performed good with LSI, with feedback about previous relevant abstracts registering the best performance.

Yan and Garcia-Molina (1993) also suggest various index structures for indexing profiles and algorithms that efficiently match documents against large number of profiles.

A related development, which is also motivated by information overload on Internet, is the eminence of the "*push model*" to which SDI was described as a precedent. The idea is that, so as to relieve the user from the hard-work of finding information from the web, an application working somewhere in the Net "pushes" selected information in the user's direction based on some criterion, such as news categories, time of day, etc., as opposed to users having to navigate one or more web sites and "pull" it down from there (Reva, 1997). According to Amy (1997), push technology can be grouped into five categories, viz. *Notification* (i.e. automatic delivery), *Profile*, *Automated pull*, *Automated push* (like subscription) and *Channel-changer* (content channels included in user's desktop). Fundamental to this technology is that a system knows something about user's interest and automatically delivers ("pushes") information meeting that interest.

The prominence of this model has also contributed for reinvigorated research in information filtering and attracted big-players like Microsoft into the scene. As a result, lots of services have been launched particularly in the area of Internet-based News broadcasting.

2.4. Examples of Automated Operational SDI Systems

With regard to the types of SDI service providers, Rowley (1993) identifies three distinct categories: (1) Local libraries or information centers (2) Large database producers and suppliers (3) Online hosts.

Nowadays, there are also lots of SDI-services provided through Internet. In most cases, Internet is used to deliver notifications in the form of e-mail messages. Others allow statements of interest to be sent through e-mail which are then converted into profile by experts of the service provider. Still, some others use WWW (especially form processing facilities) to receive user profiles and even allow users to modify their profiles. Advances in telecommunications communications, electronic information technologies and workstation functionality are expected to continue giving momentum to this trend (Mountifield, 1995).

Particularly, there are lots of developments taking place in the area of automating the SDI servers. In what follows, we will review some of the services/systems that are reported to employ automatic search and delivery mechanisms with reduced human intermediary involvement.

Wyle (1996) describes the *Pasadena* (Periodic, asynchronous, selective extraction and dissemination of information through network access system) SDI system that was developed as test-bed but is providing a useful service to its subscribers. This is a complicated system that manages user-dialog, starts and runs the indexing and retrieval algorithms, and administers the similarity measures for comparing query texts with information items. It also provides facilities for profile management and process

scheduling. The system has attempted to implement some important approaches which are also used in this thesis. Further description of these approaches would be given in chapter 4.

Uncover Reveal is an electronic current awareness service that is based on the Uncover database which is a joint undertaking by the Colorado Alliance of Research Libraries (CARL) and Black wells. This system allows users to create profiles consisting of up to 50 journals titles which are of interest to them, as well as search strategies created on topic or author name. The system will then send the results of the searches automatically to the profiler's e-mail address (Mountifield, 1995).

The *VUBIS-Antwerpen SDI service* is also based on a system that automatically matches profiles with the acquisition lists of the VBUIS Antwerpen Library network and automatically delivers the results to the users linked to profiles. Unlike many SDI systems, this system keeps the user data and profiles separate and when a user is added to the file of SDI clients, he can be linked to one or more profiles. The system matches all acquisitions against existing profiles and conforming descriptions are sent to the user via e-mail (either in SGML or simple text format) according to user's preference. This SDI system also records, per user, a number of statistical data like date on which the last time records were sent to the user, the total number of profiles to which the user is associated, the total number of different records that were found, the total number of sent messages, etc. (Corthouts and Philips 1996)

Hardin Library for the Health Science, University of Iowa, provides a user managed automatic SDI system using Healthnet. This system allows users (through its search

interface) to store, on a permanent basis, search strategies in which the user has ongoing interest. The user may execute the stored SDI search anytime he/she wishes. The system then automatically downloads the search results from the latest update to the user's computer from where he/she can browse, edit, print using the system's interface (Harden Library 1996).

The Jacques Delors Information Centre that is founded by the Portuguese Republic and the European Community (EU) in 1994 also provides an SDI service by which it disseminates information on subjects relating to the European union held on its databases. This system is an automatic service that deliveries information dynamically and promptly based on a pre-established subject interest preface. The information that can be requested are bibliographic information on EU subjects and/or EU legislation and users can receive the deliveries according to the frequency he/she prefers (Jacques Delors Information Centre, 1997).

The Pan American Information Network on Environmental Health (REPDISCA), a decentralized system operating in Latin American and the Caribbean, offers an SDI service via Internet to keep its users abreast of the latest developments in their fields of interest and update on recent documents recorded in its database. Users can subscribe to this service by sending an e-mail stating their field of interest and automatically receive, every two weeks, a set of document entries related to the subjects they selected (REPIDISCA, 1997).

Keenan and Montgomery (1994), describe an SDI service which electronically mails funding information by matching a researcher's keywords. The software matching

keywords in user profiles and funding databases enables researchers automatically learn about appropriate funding opportunities.

There are also a range of similar Automatic SDI services implemented by on-line hosts like STN, Dialog, OCLC, ISI (for example, Research Alerts, Corporate Alerts, Journal Tracker, Professional Alert, Personal Alert[2]). Commercial science publishers like Elsevier (for example CONTENTS-Alert), Springier Verlag science publisher also provide services that allow in one way or another the storage of personal profile and automatically deliver search results like bibliographic citations, journal table of contents, etc. (Mountifield, 1995).

2.5. The Experimental Case - ILRIAlerts

In this section we will turn to the case SDI system, namely ILRIAlerts , and try to see its functions and procedures with particular emphasis to user profile management processes. What is presented here is a summary of the findings of interviews, observations and analyses of relevant documents. Following the description of the system, attempt is made to identify its major problems

2.5.1. System Overview

ILRIAlerts is a computerized SDI/alerting service provided by the ILRI (International Livestock Research Institute) Information Services. Established in early 1983, this service is designed to keep scientists engaged in research on livestock production and related

[2] Corporate Alerts, Journal Tracker, Professional Alert and Personal Alert are Internet-delivered alerting services launched in 1996 (Library Technology, 1996).

topics abreast of new developments in their particular fields of interest by regularly scanning the world-wide animal agriculture literature.

ILRI is a research institute with the objective of conducting research on livestock production related fields as well as strengthening the ability of the National Agriculture Research Systems(NARS) to conduct technical and policy research in these fields. The ILRI Information Services is an information service unique in tropical Africa that is backed up by a documentation center that includes not only an extensive library on livestock and related topics, but also a microfiche collection of non-conventional literature and a variety of computer-based information products and service. The Information Service Unit aims to serve not only the Institute's own research staff but also those in NARS and other extra-mural users who are involved in livestock research.

ILRI assumed the role of providing the SDI service (ILRIAlerts) due to the comparative advantage it has as compared to NARS as far as access to comprehensive sources of information and possession of required computer resources and skill are concerned. The specific objectives of ILRIAlerts as stated in (ILCA, 1991) includes the following:

a) To alert users to those items of recent literature that have a high probability of usefulness to their expressed needs and interests.
b) To enable regular feedback on usefulness of items disseminated so that the service may be made dynamically responsive to changing needs and services.
c) To avoid information overload by attempting to minimize the extent of irrelevant information that is disseminated.

ILRIAlerts aims to meet these objectives by regularly sending to each scientist (group of scientists) all records recently added to the database in a manner that is tailored quite specifically to the needs and interests of each.

At ILRI, the profile construction process begins by the user filling in the ILRIAlerts Service Request Form. In this form the user provides the following information:

- *Personal information* - Personal particulars, highest degree acquired, field of specialization, university, country, year of highest degree; Category of affiliated organization; full address
- *Subject interest statement* - field of interest and on-going work, in narrative form; title of current project(s); citation of at least two documents relevant to the requested topic (if known by user); aspects of the topic(s) particularly *important* (e.g. geographic areas, sub-topics, breeds, species); aspects of the topics that are *not important* (e.g. . geographic areas, breeds, species); statement of expected level of coverage, i.e. comprehensive or specific (which dictates whether the search terms are restricted or not to a specific portion(s) of the records).

Upon receiving the completed request form, the information is converted to user profiles, i.e. a series of actual search expressions, using CAB Thesaurus, AGROVOC and ILRI list of key words. In this process, according to the policy manual (ILRI, 1991), consideration is given to the specific research projects in which the user is involved as opposed to broad description of his interests.

Once the user profile is constructed in this manner, it will be run on a sample database so that the database expert will see how much effective it is in retrieving relevant documents and vice versa. Based on the results of this, necessary refinements are made on the profile and passed onto testing on actual database.

Since the bibliographic database based on which the SDI service is provided is maintained using Micro-CDS/ISIS, a UNESCO supplied text retrieval software, the user profiles are composed of CDS/ISIS search statements describing the users' interest. As per the requirement of the *NEWSDI*[3] program which is used for the production of SDI outputs from Micro-CDS/ISIS database, the profile is maintained as a separate text file containing all the information pertaining to all users.

The ILRIAlerts is provided based on an in-house database, known as ILRIBIB, that integrates input from external databases with data from locally generated input. The external databases are CAB Abstracts and AGRIS, two big agricultural databases of the world, which are received every month and searched for records relevant to ILRI. The result of the search is then downloaded to ILRIBIB after modifications to meet the needs and formats of ILRI. The local input to the ILRIBIB database includes records on books and other conventional and non-conventional literature that ILRI adds to its collection. ILRI publications and other acquisitions of the ILRI Library and Documentation Unit are major constituents.

With regard to matching, the user profiles stored as a text file are matched by NEWSDI program against *monthly* updates of records from ILRIBIB. The program matches the user profiles one by one accumulating the output for each in a given disk file. The output

[3]This is an ISIS Pascal program developed at ICRISAT (India)

contains full address of ILRI Information Services, user's ID and address and the full record of the hits of his/her profile from the records of the current month. The output is commonly produced in two columns - one to be retained by the user while the other is to be sent back to the Information Service with a feedback. In particular, as a built in feedback mechanism, each output record contains at the end, questions and choices where the user indicates whether or not each record is relevant and whether he/she would like to get a copy of the original document (i.e. request for document delivery). The questions read as follows:

Did you find this item relevant ? [Yes/No/Can't say]

Do you want a copy of this item ? [Yes/ No]

Accordingly, the user is expected to fill in his/her responses, tear the printout into two(separating the two columns) and mail back the feedback copy (i.e. the column containing the feedback questions). According to the Information Policy and Procedure Manual (ILRI, 1991), the *Precision Ratio* for each feedback should be determined using the formula $(a/a+b) * 100$ where a is the number of items marked 'relevant' by the user in the Feedback Copy, and b is the number of items marked "Not relevant". The acceptable Precision Ratio, according to the policy, is in the range of 60 to 70 percent. A ratio less or greater than this range is an indication of profile inefficiency telling either it is too broad or too narrow, respectively. It is also stated that for each user, Precision Ratios are calculated for the first 4 to 6 months. To conclude that a user's profile is effective, a user should get output which provides an Average Precision Ration of 60 to 70 percent, calculated over the first six months.

If the above condition is not fulfilled, the profile will be modified based on analysis of those items which are marked "Not Relevant" in the different outputs. The manual (ILRI, 1991) also states that the user may be contacted in person or in writing if required. The profile is also changed if the user reports a change in his subject interest. In particular, a follow up of the feedbacks provided by the users should also be made to ascertain whether they are providing feedback on a continuous basis. And, if a user fails to provide feedback for 6 consecutive times, he/she would be asked for the reasons and if he/she responds by reporting a reason requiring the change of his/her profile, this is done according to the new interest he/she provides. Otherwise, he/she will be eliminated from the list of SDI users and a letter to this effect would be sent to the user.

With the aim of exploiting the advent and wide spread use of Internet in general and e-mail facilities in particular, the Information Services at ILRI has started sending SDI outputs via e-mail instead of postal services for those that have e-mail addresses. This has entailed some modifications to the feedback provision, where users print, fill and send back responses using e-mail. It is planned that in a year's time the postal delivery will be abandoned and all SDI outputs will be delivered using e-mail only.

2.5.2. Limitations in User Profile Management

According to my observation following the survey, at ILRI, keeping the profiles of users dynamically updated is becoming a difficult and costly function. This is so for various reasons including the following:

1. There is minimum or no interaction between most users and intermediaries. For example, the fact that the users are distributed throughout Sub-Saharan Africa is one of the reasons for this situation while the limited capacity of the system in terms of number of intermediaries is the other. This situation renders the system short of a very important component that is highly recommended in services of this kind namely close interaction with and follow up of users and their information interests.

2. Mostly users don't tend to actively provide feedback in the current systems. This is for various reasons major among which is the fact that the process takes up much of their time. The situation is aggravated by the fact that the existing system expects item-by-item relevance feedback which is highly tasking;

3. Mostly, user profiles don't fully reflect the dynamic interest of users. There are also cases where the system continues to deliver information based on profiles which no longer represent user's interest. Most of the user profiles based on which the service is provided have not been updated for more than two years now and their performance has not been appraised for long. One reason for this, according to ILRI staff, is that users don't tend to request the change of their profile - especially if the change of their interest is dynamic - due to factors like lack of time or unwillingness to exert the effort required;

4. Under the circumstances, the human intermediary should initiate profile updating. This, however, is not efficiently practiced due to the fact that it is tasking on the part of the limited number of intermediary in place.

5. Feedback analysis and refining of search profiles based on it is a labor-intensive process, hence more overhead to the system (this is also being addressed by researches in feedback analysis systems). In ILRIAlerts, although there is a relevance feedback system built in the service, the system usually does not analyze them due to shortage of man power. Although the feedback component was meant to optimize and update the user's profile based on his/her reaction to the previous search results, due to the capacity problem, the system has resorted to monitoring whether the user provides feedback or not so as to determine user's need of the service. For instance, recently the system tried to count how many times a user has provided feedback in the last six months so as to determine how many actual users they have. The same capacity problems have also forced ILRIAlerts to constrict the coverage of its services. Currently ILRIAlerts service is limited to the Sub-Saharan Africa region only while ILRI's mandate includes India, South-east Asia, China and Latin America.

6. As indicated earlier, SDI systems should optimize/update the user's profiles (user model) by gathering information on environmental and organizational factors and documents like projects a user is involved in, user's information utilization habit, goals and motivation, past publications, etc. Again, ILRIAlerts is not successful in this regard.

2.6. Summary

As can be learnt from the review of the literature, the whole SDI/filtering sphere can be characterized by two diametrically disparate trends. While there is a very dynamic and highly innovative work that aims to exploit the latest computing techniques that mainly

focuses on Web-based information filtering, most of the widely established traditional SDI systems based on Luhn's model still employ techniques that date decades back. However, few developments have been made with regard to enhancing the user friendliness of these systems. As opposed to the filtering literature, even recent literature on SDI describes end-user friendly systems as belonging to the future. Although it is felt that the review made in this study may have suffered from the inaccessibility of recent literature on both areas, only few works like the dissertation by Wyle (1996) and works by Yan and Garcia-Molina (1993a, 1992, 1994) tried to treat these areas together.

As this review demonstrates, there remains a lot to be done in applying the techniques of the advancing Web documents filtering area to the retarding SDI domain. Given the wide availability and use of SDI systems, such an undertaking can benefit a lot of operational systems. And, it is based on these premises that the current work reported in the succeeding pages has been conceived and carried out.

CHAPTER 3

INTERFACE AGENT TECHNOLOGY

3.1. Overview of Agent Technology

As has been stated in the introductory chapter, this thesis aims to explore whether the new technology known as agent technology can be applied to solve the problems related to SDI discussed in the preceding chapter. To this end, an attempt has been made to review the literature on this technology in general and Interface agent technology in particular to serve as a background to the proposed model. Despite the relatively recent emergence of agent research and development, a large body of literature has already been produced on the area. In this chapter, however, a very summarized review of the technology along with a brief discussion on why it has been thought to be appropriate in the SDI domain is presented.

3.1.1. What is an Agent ?

Being a recent development in the computing world, there is no agreed definition for the concept of agent. Various authors present different, and sometimes mutually exclusive, features that they believe should characterize an agent. As this thesis is not meant to deeply analyze the characterization and taxonomy of agents, the definitions that appear to be widely accepted are taken as a reference. Very broadly, there appear to exist at least two different ways in which agents are understood. One that views an agent as a software "*assistant*", while the other that views it as an extension of programming *objects*.

Viewing agents as "assistants", Sycara, *et al.* (1996) define them as "programs that act on behalf of their human users to perform laborious information gathering tasks". In this

paradigm, a primary aspect in the use of intelligent agents is the concept of *delegation* of authority (Maes 1994). This means, the user delegates the responsibility (and drudgery) of performing certain time consuming computer operation to "smart" software, i.e agents. Based on this, the concept of *agency* is defined as the degree of autonomy and authority that the user permits the agent to have (Gilbert, et al, 1995).

Another definition of agents is based on Shoham's (1997) *Agent oriented programming*. Bradshaw, *et al* (1997) define agents from this perspective as follows.

> *" ..... an agent can be thought of as an extension of the object oriented programming approach, where the objects are typically somewhat autonomous and flexibly goal-directed, respond appropriately to some basic set of speech acts (eg. requests, offer, promise), and ideally act in a way that is consistent with certain desirable conventions of human interaction such as honesty [trustworthiness] and non-capriciousness. From this perspective, an agent is essentially an object with an attitude."*

Although not exactly the way put in the above, various authors have used the dichotomous way of defining agents. For example, agent as *ascription* and agent as a *description* (Bradshaw 1997), *weak* and *strong* definition of agent (Wooldridge and Jennings 1995), *adaptive functionality* and *agent metaphor* (Erickson, 1997).

More concretely, agents can be defined as software entities at varying stages of capabilities. Gilbert *et al* (1995) of IBM describe the various skills of agents in terms of a space defined by the three dimensions of *agency*, *intelligence*, and *mobility* as shown in the figure below.

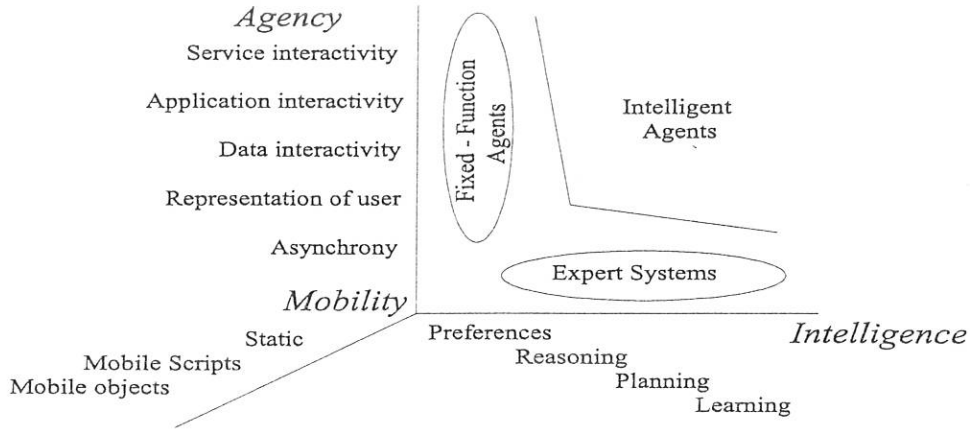


Figure 3.1. IBM intelligent agent graph

In general, the various features that are attributed to agents by various authors as summarized by Bradshaw (1997), include:

- *Reactivity* (selective sensing and acting);
- *Autonomy* (goal directedness, proactive and self-starting behavior);
- *Collaborative behavior;*
- *"Knowledge level" communication ability;*
- *Inferential capability* (ability to use prior knowledge);
- *Temporal continuity* (persistence of identity and state over period of time);
- *Personality* ("believable" character);
- *Adaptivity* (learning and improving with experience); and
- *Mobility* (ability to navigate through a network and perform tasks on remote machines).

Structurally, an intelligent agent consists of a *sensing* component that can receive events, a *recognizer* or *classifier* that determines which event occurred, a set of *logic* ranging from hard-coded programs to rule based inferencing, and a mechanism for *taking action* in the world (Bigus, 1996).

With the above features, agents came as extensions of existing concepts like objects and expert systems. However, they add a number of unique and advanced capabilities and philosophies. For example, while they, to a great extent, share the feature of being *autonomous* with expert systems, they also have abilities to *cooperate* and *learn* which are absent in expert systems (Nwana, 1996).

Originally, the emergence of agents is attributable to two major phenomena:

(1) The challenge of locating information sources on Internet and accessing, filtering, and integrating information from this network in support of decision making as well as coordinating information retrieval and problem solving efforts (Sycara, *et al*, 1996); and

(2) The failure of stand alone AI systems which lead to Distributed Artificial Intelligence (DAI), which in turn resulted in agents. Sarma (1996) explains this situation as follows:

> *"stand-alone AI systems failed after a stage because of their brittleness. A more powerful strategy which has been explored is to put the system in a society of systems where it can draw on the expertise and capabilities lying elsewhere. The overall system then becomes a collection of individual modules which interact productively to solve a large ill-structured problem. Each individual module then can be called an intelligent agent."*

3.1.2 Types of Agents

As with the definitions for agents, a myriad of typologies of agents are documented in the literature. Bradshaw (1997) provides a summary of some of these typologies. Here, we review the one provided by Nwana (1996) which is relatively comprehensive and fitting to

the purpose of this thesis. According to this typology, there are seven types of agents and combinations of any two or more of these.

1) *Collaborative agents* - these are agents that are mainly characterized by their autonomy and cooperation (with other agents) in order to perform tasks for their owners. These agents may learn, but this aspect is not typically a major emphasis of their operation.

2) *Interface Agents* - these are agents that are primarily characterized by their autonomy and learning in order to perform tasks for their owners. These act as personal assistants to their user and collaborate with the user in discharging responsibility.

3) *Mobile agents* - are computational software capable of navigating wide are networks (WANs) such as the WWW, interacting with foreign hosts and gathering information on behalf of its owner.

4) *Information/Internet Agents* - These are agents that perform the role of managing, manipulating or collating information from various distributed sources on the Net. Agents in this category are also called by names like *Filtering agents* (that select relevant information to a user). *Information retrieval agents*, etc.

5) *Reactive software Agents* - These are a special category of agents which do not possess internal, symbolic models of their environment; instead they act/respond in a stimulus-response manner to the present state of the environment in which they are embedded.

6) *Hybrid agent-* these are agents that are based on a combination of two or more agent philosophies with in a singular agent. These philosophies include a mobile philosophy , an interface agent philosophy, collaborative agent philosophy. etc.

7) *Smart agent-* These are ideal agent that do not yet exist.

3.1.3. Multi-agent systems

While a single intelligent agent is interesting as an intermediary between a single user and a system, the fully functional applications of agents usually involve the interaction of multiple agents. Nwana (1996) refers to such system as *heterogeneous agent systems* and describes them as "integrated set up of at least two or more agents which belong to two or more different agent classes".

The major challenge of implementing such systems is establishing a communication mechanism among the participating agents as well as a common ground of understanding. To address this, efforts are already underway to standardize an a*gent communication language* (ACL). Gensereth (1997) defines ACL as consisting of three parts: a vocabulary, an inner language called *Knowledge Interchange format (*KIF), and an outer language called *knowledge Query and Manipulation Language* (KQML).

KIF is a prefix version of the language of first order predicate calculus, with various extensions to enhance its expressiveness, using which the internal knowledge of the agent is represented and agent's internal components communicate with KQML, on the other hand, is a linguistic layer using which one agent exchanges messages with another.

3.2. Interface Agents

Among the aforementioned types of agents, the present study aims to explore one particular type of agents called Interface agents. As briefly introduced before, these agents aim at assisting the user in computer related and information seeking operations. Most of the SDI user profile management problems described in the preceding chapter also relate to user system interaction. Hence it is thought that the exploration of agents as possible solutions to these problems should probably start from interface agents. Accordingly, this section provides further details on these types of agents. As will be shown in the next chapters, some of the other types of agents are potentiality applicable for other aspects of SDI systems.

According to Maes (1994), who is said to be a key proponent of this class of agents, the key metaphor underlying interface agents is that of a *personal assistant* who is *collaborating* with the user in the same work environment. The assistant becomes gradually more effective as it learns the user's interests, habits, and preferences (as well as those of his or her community). Maes (1994) also identify the following ways in which such agents might be helpful.

- They hide the complexity of difficult tasks.
- They perform tasks on the user's behalf (through adapting over time, to the user's preferences and habits).
- They can train or teach the user.
- They help different users collaborate (allowing the sharing of know how among different users).

3.2. Interface Agents

Among the aforementioned types of agents, the present study aims to explore one particular type of agents called Interface agents. As briefly introduced before, these agents aim at assisting the user in computer related and information seeking operations. Most of the SDI user profile management problems described in the preceding chapter also relate to user system interaction. Hence it is thought that the exploration of agents as possible solutions to these problems should probably start from interface agents. Accordingly, this section provides further details on these types of agents. As will be shown in the next chapters, some of the other types of agents are potentiality applicable for other aspects of SDI systems.

According to Maes (1994), who is said to be a key proponent of this class of agents, the key metaphor underlying interface agents is that of a *personal assistant* who is *collaborating* with the user in the same work environment. The assistant becomes gradually more effective as it learns the user's interests, habits, and preferences (as well as those of his or her community). Maes (1994) also identify the following ways in which such agents might be helpful.

- They hide the complexity of difficult tasks.
- They perform tasks on the user's behalf (through adapting over time, to the user's preferences and habits).
- They can train or teach the user.
- They help different users collaborate (allowing the sharing of know how among different users).

- They monitor events and procedures.

According to Laurel (1997), interface agent have four key characteristics:

1) *Agency* - ability to take action on behalf of user;
2) *Responsiveness* - ability to be responsive to the user;
3) *Competence* - being knowledgeable about the user as well as the domain of the application or environment in which it operates; and
4) *Accessibility* - accessibility of agent's traits and predispositions to the user.

So as to be able to carry out these operations, interface agents should have the capability to learn. Maes (1994) defines four ways in which a user interface agent can learn.

1) By observing (through window-system events and imitating the user's behavior).
2) Through receiving positive and negative feedback from the user, whereby the user "grades" the agent on how well it performed an action.
3) By receiving explicitly instruction from the user.
4) Through communication with other agents.

Maes and Kozierok (1993) emphasize that for an application domain to benefit from interface agents, it should fulfill two condition (1) the use of the application should involve a lot of repetitive behavior (otherwise, the learning agent will not be able to learn anything) and (2) this repetitive behavior is very different for different users (otherwise, use of a knowledge based approach is to be considered).

Interface agents can assist the user in a range of areas including information filtering, information retrieval, mail management, meeting scheduling, selection of books, movies, music etc. (Maes, 1994). The following are some of the existing interface agents and their domains as described in the literature:

Table 3.1. Interface Agents

Agent Name	Authors	Domain
Calendar Agent	Kozierok & Maes (1993)	for scheduling meeting
Letizia	Liebermann (1995)	guides user in searching Internet
Remembrance Agent	Rhodes & Starner (1996)	retrieve relevant e-mails in user's directory relating to the e-mail currently being composed
NewT	Sheth and Maes (1993)	help user filter and select articles from a continuous stream of Usenet Netnews
Yenta/Yenta-lite match making agent prototype	Foner (1996)	Matching buyers and sellers of some item and introducing them to one another, etc.
Ring/HOME system	Shardanard, Maes 1995; Maes 1995	Personalized recommendation for music albums and artists
Open Sesame		a personalized assistant to get new releases of music, movies, books, TV and local events
IBM's Web Browser	IBM	remembers wherever a user have been on the web, what he/she found there, and can help recall any word on any page that the user has visited.
URL - minder		keeps track of web pages and other resources on the world wide web, and sends e-mail whenever users registered resources change
InfoAgent	D'Aloisi and Giannini, (1995)	Supports users in retrieving data in distributed and heterogeneous archives and repositories.

3.3. Information Retrieval and Filtering Agents

The currently widespread approach being employed in developing Personalized Information filtering (IF) systems is the application of artificial Intelligence techniques in general and agent technology in particular. The adoption of the later to IF has resulted in

what are grouped as "*Information filtering agents*" (Sheath, 1994; Maes, 1994). These are essentially systems that filter information (so far mainly from the Internet) by learning their user's interest. The most important contribution of this approach is the implementation of systems that *learn* and *adapt* to user interests using machine learning algorithms. There are lots of such systems developed for such areas as USENET new filtering (e.g. NewT by Sheth(1994)), e-mail filtering, web pages filtering, etc. What follows is a review of some related works with particular emphasis to those dealing with document databases.

Pannu and Sycara (1996) describe a reusable personal agent that learns a model of the user's research interests for filtering conference announcements and request for proposals (RFPs) from the web." Their agent starts learning user's interest from papers and proposals the user had written. For the sake of comparison, they employed two learning mechanisms, namely tf-idf weighting (IR based approach) and Neural Net techniques and found both approaches good in creating profiles with the tf-idf the most promising.

Krulwich (1995) describes an agent that learns user's interests from sets of documents that the user indicates are and are not of interest and use it to filter relevant document from "heterogeneous document databases" distributed and dynamic databases consisting of mail messages, news articles, technical articles, on-line discussion, client information, proposals, design documentation, and so on)

Balabanovic and Shoham (1995) developed "a system which learns to browse the Internet on behalf of a user". It searches the WWW taking bounded amount of time, selects the best pages and receives an evaluation from the user. The evaluation is used to update the search and heuristics. Pazzani, et al (1996) collect ratings of the explored Web papers from the

user and learn a user profile from them. Pages are separated according to their topic and a separate profile is learned for each topic.

Lang (1995) developed a system for electronic news filtering that uses text-learning to generate models of user interests. Holte and Drummond (1994), Drummond, *et al* (19950 designed an agent-based system that assists browsing of software libraries, taking keywords from the user and using a rule-based system with forward chaining inference, assuming that the library consists of one type of items and the user goal is single item. Etzioni and Weld (1004) offer an integrated interface to the Internet combining a UNIX shell and the World Wide Web to interact with Internet resources. Their agent accepts high-level user goals and dynamically synthesizes the appropriate sequence of Internet commands to satisfy goals.

There are also lots of research in the automatic generation of user profiles using machine learning techniques. But as indicated before, most deal with interests in the area of news, e-mail in win popes, filtering (for eg. Sheth, 1994, Edwards, et.al, 1996). These studies used such learning methods as term frequency/inverse-document frequency (tf/idf) Weighting Minimum Description Length (MDL), winnow, CN2, IBLP1, genetic algorithms, etc.

According to the foregoing review, machine learning approaches seem highly promising in implementing truly "personalized" alerting systems. However, there still remains a lot to be done as far as user interest profiling is concerned. In particular, there is much to be explored in the area of services based on document databases (particularly document databases describing/containing technical and scientific literature, which, traditionally,

were the focus of SDI systems) as most efforts were directed towards news and other web based documents.

Moreover, a work in this area is particularly pertinent as user interest profiling might differ from domain to domain. As stated by Foltz and Dumais (1992), "a factor that provides a good description of interests for world news, may not be effective for describing a person's research interests". Save such differences, so far, it is not yet resolved how best a user profile that captures user interests be generated in the form that can be used for searching as well as with minimum user interaction (Bloedorn, *et al*, 1996).

3.3.1. Why Agents in SDI systems ?

SDI lends itself to agent application for the following reasons:

1. Because it involves costly and routine tasks that can be delegated to some automated system. These expensive and tedious tasks have prevented SDI systems from wide implementation and have been sources of discouragement for libraries and information centers. The need to automate this process have been recognized since the early days of SDI (for example, Barker, 1972).

2. Because the current requirements for profile construction(extended user-intermediary discourse) doesn't go along with the cases of remote SDI services where it is very difficult for a user to reach the intermediary, especially qualified intermediaries. This is more aggravated by current development in Internet based SDI deliveries. Added to these is the fact that only limited number of such intermediaries exist and only well-off libraries and information systems can afford enough of them.

3. Because the current human-intermediary based approach puts too much requirements on user's time and effort, which in turn results in less motivation for many user to use the services. For successful profiles, users are required to devote time for extended interviews with intermediaries including the need to go to the intermediary, deliver feedback, report interest changes, etc. The utilization of networked approach has gone to some extent in resolving these with a lot remaining as far as interaction with intermediary is concerned. Hence, agent application can complement this endeavor of reducing the required user effort.

4. Because of the commonality of purpose between SDI systems (and particularly their interest profiling) and agents. In fact, some of what present agents are intended/claimed to do is what SDI systems were doing for the last 40 years or so - e.g. agency (searching and selecting information on behalf of user), personalization (knowing user and adapting to his behavior), notification, etc. This makes SDI systems ideal for agent application.

5. Even the need by intermediaries for assistance in profile compilation for some automated system to which they could delegate aspects in which they do not want to spend time or in which they are not fully conversant, like remembering user's past records, accessing some user related information, analysis and determination of the subjects as well as the standing of the inquiries within the structure of a given field, etc.

3.3.2 Review of Agents for SDI Systems

There are a range of SDI and filtering systems that applied agents, or attributed the name "agent" to their systems although some might be questionable in view of the foregoing definitions. In this section, attempt will be made to review some of these systems.

The *Personal Agent* that is included in the PLWeb software of the Personal Library Software company is described to monitor incoming information for web publishers and users and automatically select the most pertinent items, based on 'intelligent queries' called *agents*. Agents send e-mail alerts to users, referring by URL to a web site where the search results can be viewed. Personal Agent also monitors incoming data, suggesting new words and concepts for inclusion in queries (Library Technology, 1997).

The *Topic Agent Server Toolkit* released in 1996 by Verity Inc. is another filtering technology that allow Internet documents to be filtered against personal profiles in real time. Topic Agents use Verity concept-based search technologies to automate the process of monitoring information sources and delivering relevant information to the user. The Topic Agents analyze documents and prioritize them for delivery based on relevancy ranking (Library Technology, 1996)

Gibson (1997) describes *LineOne*, the web-based joint venture between News International and British Telecom, as being able to allow users to set up personal profiles of interest and have agents scan the web for relevant stories 24 hours a day. Subscribers can input a title and free text instructions for 25 personal agents, which then scan the web and retrieve information that matches the user's preferences. The agents can also be instructed to

search for other agents that have been set up by people with similar interests. Once one of these is found, it can be cloned by the user and adopted as part of their own profile.

3.4 Agent Development Approaches

The development of agents or agent based systems employs a number of Artificial Intelligence methods and procedures. Bentley (1995) describes three approaches to build Interface agents.

1) *Rule-based Approach* - which features a collection of user programmed rules for processing information related to a particular task.
2) *Knowledge based Approach* - building domain specific knowledge for the agent about the application and user to recognize plans and contribute to user's tasks.
3) *Machine Learning Approach* - This uses memory based reasoning combined with rules to model each user's habits. This approach is said to have a number of advantages over the previous two including: (1) requiring less initial work, and adapting over time, and (2) as a result , achieves a level of personalization.

More broadly, the various approaches that have been documented in the literature to have been employed in agent developments can be grouped into five categories as: Agent modeling, Agent oriented programming, Domain knowledge bases and representation, Machine learning and User modeling.

3.4.1 Agent Modeling

The Intelligent agent literature is full of various agent modeling philosophies and techniques. What follows is a brief review of few models, including the ones employed in this thesis.

Knoblock and Ambite (1997) suggest an agent architecture called SIMS that is mainly developed with particular reference to intelligent agents for information gathering. This architecture is designed to provide:

1) *Modularity* - in terms of representing an information agent and information source
2) *Extensibility* - in terms of adding new information agents and information sources
3) *Flexibility* - in terms of selecting the most appropriate information sources to answer a query ,
4) *Efficiency* - in terms of minimizing the overall execution time for a given query and
5) *Adaptability* - in terms of being able to track semantic discrepancies among models of different agents

As stated by Knoblock, *et al.* (1994), in this model each information agent is specialized to a single "application domain" and provides access to the available information sources within that domain.

IBM's (1997) Agent Building Environment Developer's Toolkit provides five frameworks or components that are grouped in three categories namely *skills*, *knowledge* and *Interface*. Skills describe what the agent can do or what the user can expect from the agent while knowledge refers to information about the user's preferences or needs held by the agent.

Interface includes both the user and the application interfaces. Under these come the basic components of the agent as shown below:

Table 3.2 : IBM's agent Modeling Frameworks

Knowledge • *Library Framework* - the persistent storage of knowledge • *Knowledge Framework* - Agent's current understanding of the user and the interactions received from the user. *Skills* • *Engine Framework* - the agent's brain to reason and learn *Interfaces* • *View Framework* - the visual presentation of the agent library to the user. Commonly referred to as the User Interface. • *Adapter Framework* - The agent's road map, used to connect to information. Commonly referred to as the Application Interface.

Sommaruga and Catenazzi (1995) proposed a stratified view for a cooperative agent model, which they claim to help a programmer identify and organize distinct levels of functionalities within an agent. These four levels are:

- *Real world level* - e.g. real-life problem the agent is supposed to interact with
- *Application level* - application dependent part of the system/ agent
- *Abstraction Level* - represents the competence abstraction, such as the definition of the agents' models through their skills and needs
- *Cooperation Level* - It identifies the level of cooperation, where the agent control resides.

Kendall, *et al* (1995) describe a methodology that recommends the use of workflow modeling, particularly the *IDEF* standard methodology for integrated computer aided

manufacturing (ICAM), and the *use case* driven object oriented software engineering (OOSE) for the design of agent based systems.

In Kendall, *et. al*'s (1995) methodology, the workflow analysis using IDEF[4] diagrams is used to identify the various functions in the system under study and give each one an ICOM (Input, control, output, and mechanism) representation. The use case modeling is used to describe how users interact with the system in a certain mode of operation. In use case modeling, a user in a particular role is known as an *actor* and each actor uses one or more use cases while a *scenario* is an instance of a use case.

On how to identify agents in a system, Kendall, *et al* (1985) state the following:

> *" Agents appear in applications when pro-active behavior - automated decision making - is required. Pro-active behavior is necessary when control decisions are required to augment or replace human decision making. Agents, therefore, arise in use cases to replace or assist actors".*

Once the agents have been identified, their *goals* and *plans* must be established, along with their *beliefs*. So as to deal with objects that exist in the external environment, agents need also have *sensors* and *effectors* which themselves can be implemented as objects. The sensor object monitors and filters output from objects in the application, alerting the agent when certain conditions exist. Effector objects can carry out an agent's intentions and actions, impacting other objects in the environment. These definitions by Kendall, et al (1995) match with those given IBM's (1997) ABE Developer's Toolkit except that the later uses the term *trigger* to refer to alerts to the agent of certain conditions in the environment while sensors are agent requests on environmental conditions. As the IBM's

[4] the acronym IDEF stands for ICAM Definition.

ABE Developer's toolkit is the one used for prototype development, further discussion will be given in chapter 5.

3.4.2 Agent Oriented Programming

The phrase *Agent oriented programming (AOP)* was coined in 1989 by Shoham to describe a new programming paradigm, one based on cognitive and societal view of computation (Shoham, 1997). AOP is an extension of object oriented programming with fundamental enhancements. In AOP agents are defined as entities whose state is viewed as consisting of mental components such as beliefs, capabilities, choices, and commitments. With this so called intentional description of a software system, AOP introduces a formal language with syntax and semantics to describe the mental states and an interpreted programming language, called AGENT-0 that has semantics consistent with those of the mental states and in which a programmer can program an agent.

3.4.3 Domain Knowledge Bases and Representation

An agent that features hard-coded logic is, in most ways, just a piece of interpreted code. It is the addition of knowledge bases and the inference engine that moves an agent up to the next step in the intelligence hierarchy. Each agent contains a model of its domain of expertise and models of the other agents and information sources that can provide relevant information (Knoblock, *et al*, 1994).

A variety of knowledge representation techniques have been developed in the Artificial intelligence field. Although most of these are employed for agent knowledge representation in different agent-based systems, the most widely used one is the *rule-based technique* and associated forward chaining inferencing. The fact that the standard agent

knowledge representation format called *Knowledge Interchange Format* (*KIF*) is of this category is a good evidence for the above statement. KIF is a prefix version of first order predicate calculus with extensions to support nonmonotonic reasoning and definitions. This language is intended, as a powerful vehicle to express knowledge and meta-knowledge. There were two different motivations behind the development of KIF: (1) creation of a *lingua franca* for the development of intelligent applications, and (2) creation of a common interchange format (Finnin, *et al*, 1997).

3.4.4 Machine Learning

As mentioned while discussing interface agents, the ability to learn is a core feature expected from most agents. This is possible using machine learning techniques (or algorithms) which have been developed for the last two decades and are applied in range of areas like data mining, pattern recognition, text classification etc. There are a range of machine learning techniques documented in the literature including decision-tree Induction (symbolic inductive learning algorithms), artificial neural networks, genetic algorithms, probabilistic learning algorithms (e.g. Naive Bayesian classifier), K-Nearest Neighbors, Instance-based learning, memory-based reasoning, etc. (Mitchell, 1997). Chen (1995) also provided a detailed discussion on some of these techniques.

Of particular relevance to the current work is the area of learning based on text documents. Basically, the learning in this area involves asking users to rank given documents and generating model of user interests based on user's ranking (Armstrong et al, 1995; Balabanovic et al, 1995, Pazzani, et al, 1996). So far these methods have been mainly used for documents written in the HTML format.

Many current systems that learn from text use the bag-of-words (word vector) representation using either *Boolean features* indicating if a specific word occurred in a document (e.g. Armstrong et al 1995; Pazzani, et al 1996) or *frequency of a word* in a given document (e.g. Balabanovic and Shoham 1995, Berry, et al 1995). One of the well established technique for learning over text in Information Retrieval is to represent each document as a TFIDF*-vector in the space of words that appeared in the training documents (Buckley, et al 1993), sum all the interesting document vectors and use the resulting vector as a model for classification based on Rocchio's (1971) relevance feedback method. This technique is extensively used in information retrieval agent experiments that learn from World Wide Web data (e.g. Armstrong, et al 1995; Lang, 1994; Balabanovic and Shoham, 1995; Berry, et al 1995; Ittner, et al 1995; Cohen 1996; Pannu and Sycara 1996).

3.4.5 User Modeling

Closely related to machine learning capability of agents is the capability to build a model of the user. User modeling is an approach which was being investigated in the areas of information retrieval and artificial intelligence for a long time (Saracevic et al, 1997). In the specific area of agents, the user model is an important component of an agent's knowledge which it uses to adapt to its individual users. As such, the user model is the basis for personalized agent-user interaction. To keep itself up-to-date about user's preferences, an agent should have a mechanism of constantly tracking and updating its current user model (Stefanuk, 1996). For this purpose machine learning techniques are reported in the literature to be helpful.

In general, the user model should include information about the user's knowledge, beliefs, goals and plans for achieving these goals, abilities, attitudes, and preferences. Saracevic, et al (1997) state that what Luhn called user profile in the SDI domain is also a user model that is composed of search statements expressing user interests. They also continue to state that "profiles in IR largely precede and outperform 'intelligent' agents and Knowbots in AI" and suggest the utilization of their achievements in agent systems.

* Term Frequency/Inverse Document Frequency

CHAPTER 4

PROPOSED AGENT-ORIENTED MODEL

4.1. Background

For the various reasons discussed in chapter 2, existing SDI systems are observed to be unable to provide complete services using human intermediaries only. Hence the need for automated assistance in both user directed and system-directed operations. Accordingly, recommendations for same have been repeatedly put forward by researchers in the area. In particular, the fast development of networking technology and the consequent wide area information retrieval models discussed in chapter 2 have made possible very useful solutions for some of these problems (Yan and Garcia-Molina, 1993). The potentials of agent technology in this connection have also been established in Chapter 3.

In this chapter, we present a model for the use of agent technology for the particular problems pertaining to user profile management with special emphasis to continuous updating of user profiles and their optimization (based on relevance feedback analysis) by taking a case, namely ILRIAlerts of ILRI. First a generalized agent-based SDI solution is discussed. Then, focusing on the client side, a detailed specification of constituent agents is presented. But before that, the following account attempts to briefly look at the factors and environments in which the proposed model was conceived.

As discussed in section 2.2, the user profile constitutes the critical component of SDI/ filtering systems. Moreover, most of the problems that have tremendous impact on SDI

systems relate to user profile management. Hence, it is felt that any viable model to SDI problems should address the efficient and effective management and utilization of the user profile component.

One promising approach in improving user profile management is the *distributed processing* of user profiles (Yan and Garcia-Molina, 1994). This approach builds upon the established model of using client/server (distributed) retrieval interfaces for online use. Its foundation is the assumption that computer networks, be it wide area networks (including Internet) or local area networks (including Intranets) are appropriate channels of SDI service delivery (Wyle, 1996).

Basically, *distributed processing* is an architecture where specialized portions of a certain application or system are performed on different machines (Feibel, 1995). According to Boar (1993), *client /server architecture* (or model) is an arrangement where a single application is partitioned among multiple processors (front- end and back-end) that cooperate (transparent to the user) in a unified manner to complete the unit of work as a single task. Such architecture is widely implemented as in the cases of EBSCO Information Service's System (Mountifield, 1995) and WebSPIRS* of SilverPlatter (SilverPlatter, 1997). With particular reference to agent-based systems, Nwana and Wooldridge (1996) stated that the client/server architecture and distributed object frameworks are the computing infrastructures that provide critical support for agent-based systems.

As discussed earlier, most current SDI/filtering services require human intermediaries that liaise between the information sources (databases) and the users of information. As such,

* This is based on SilverPlatter's ERL(Electronic Reference Library) open client/server architecture.

the operations performed by the intermediary can be seen as *user directed* and *system-directed.* Accordingly, using client/server architecture, the user-directed and system-directed portions of profile management can be partitioned between clients and a server respectively. The resulting client/server SDI environment would have a kind of structure depicted in figure 4.1.

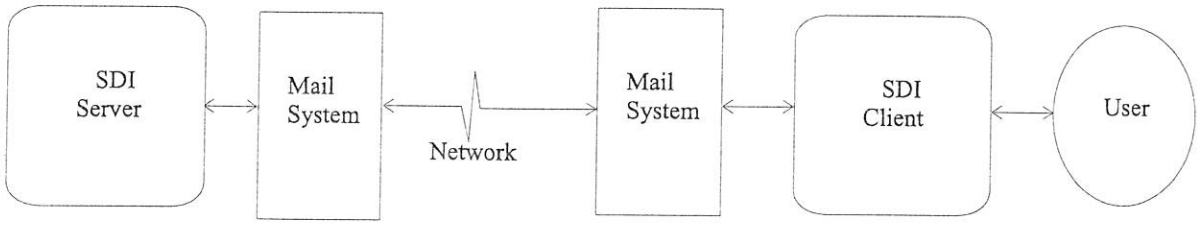


Figure 4.1 A Client/Server SDI Model

Unlike traditional SDI systems where almost all user profile management operations are concentrated on the server side, this model shares some portion of the task to the clients. In particular the client side is made responsible for providing the user an easy interface to the SDI system and close follow- up of user's information needs. It may also include such components as delivery format conversion facilities, ranked display of search results, efficient browsing of deliveries, feedback gathering and processing, maintenance of specialized personal databases from SDI search deliveries, detection of user's information interest changes and many other facilities that need close interaction with the user.

The SDI server is a component that is as important as the client side and is responsible for information filtering functions. Like any information server, it receives the user's information needs from the client in a continuous fashion and converts them into searchable expressions that are known to the databases or other information sources. It also conducts the search on the databases and delivers the results to the users.

The client and server components are tied together by e-mail system which acts as a communication channel between the two.

4.2. Agent Modeling Approaches

In many cases, agent oriented solutions are based on network of collaborating agents rather than isolated agents (for example Papazoglou, *et al* 1992; Knoblock, et al 1994; Lashkari *et al.* 1994). In this approach, each agent specializes in dealing with a specific type of task and provides information and expertise on its domain. Furthermore, the agents communicate and cooperate with each other in carrying out the tasks delegated to them based on their current collective knowledge (Thomson, 1995). In deed, these functions constitute the peculiar features of agents vis-à-vis other software systems. The agents developed in this manner can either form applications on their own or be embedded in other applications (Gilbert, 1997, Edwards, et al. 1996). The agents can also be integrated and presented to the user as a single agent (D'Aloisi and Giannini, 1995).

Similar approach has been adopted in this study in applying agents to SDI user profile management. Various agents specializing on specific domains at the client as well as server side have been identified and designed.

Another design option that needs to be taken into account by a work of this kind, that is in applying agents to hitherto human mediated information access situations, is whether a purely agent based solution or agents that complement the human intermediary are to be modeled. This study favors the latter approach which Nardi and O'Day (1996) called a

diverse information ecology where human agents and software agents collaborate to provide the required services to the users. Hence, the agents proposed in this work are thought to automate certain functions of the human intermediary while still mostly working in collaboration with a human intermediary.

For the purposes of systems specification and identification of agents, the *workflow analysis* and *use case modeling* methodologies proposed by Kendall, et al (1995) have been used as it was possible to get a fuller description on it from the literature. For the detailed modeling of the client side agents, Kendall et al's (1995) methodology as well as IBM's (1996) Agent Building Environment's (ABE) agent structure have been employed. The latter approach is chosen to include more concrete specifications in the model.

4.3. The Generalized Model

The Level 1 $IDEF_0$ diagram of the workflow analysis of SDI systems is given in fig. 4.2. As shown six basic functions can be distinguished : f*ormulate initial profile*, t*est & refine profile*, *search and deliver results*, *follow-up performance of profile*, *track user's new interests* and *update user's profile*. The level 1 $IDEF_0$ diagram is further analyzed as shown in figure 4.3 and 4.4.

Among the six basic functions identified from the workflow analysis, the last three, namely *follow-up profile performance*, *track user's new interests* and *update user's profile* involve continuous interaction with user and, hence, require closeness to user as well as access to user interest related information sources. Accordingly, this feature makes the client side the ideal position for automated systems that assist in these operations. Conversely,

formulate initial profile, *test and refine profile* and *search and deliver results* are technical operations involving databases and intermediary expert inputs and therefore systems assisting in these operations are better placed on the server side. However, such categorization is by no means considered exclusive. The server side operations require some degree of user interaction while client side operations are finally processed and put into effect by the server.

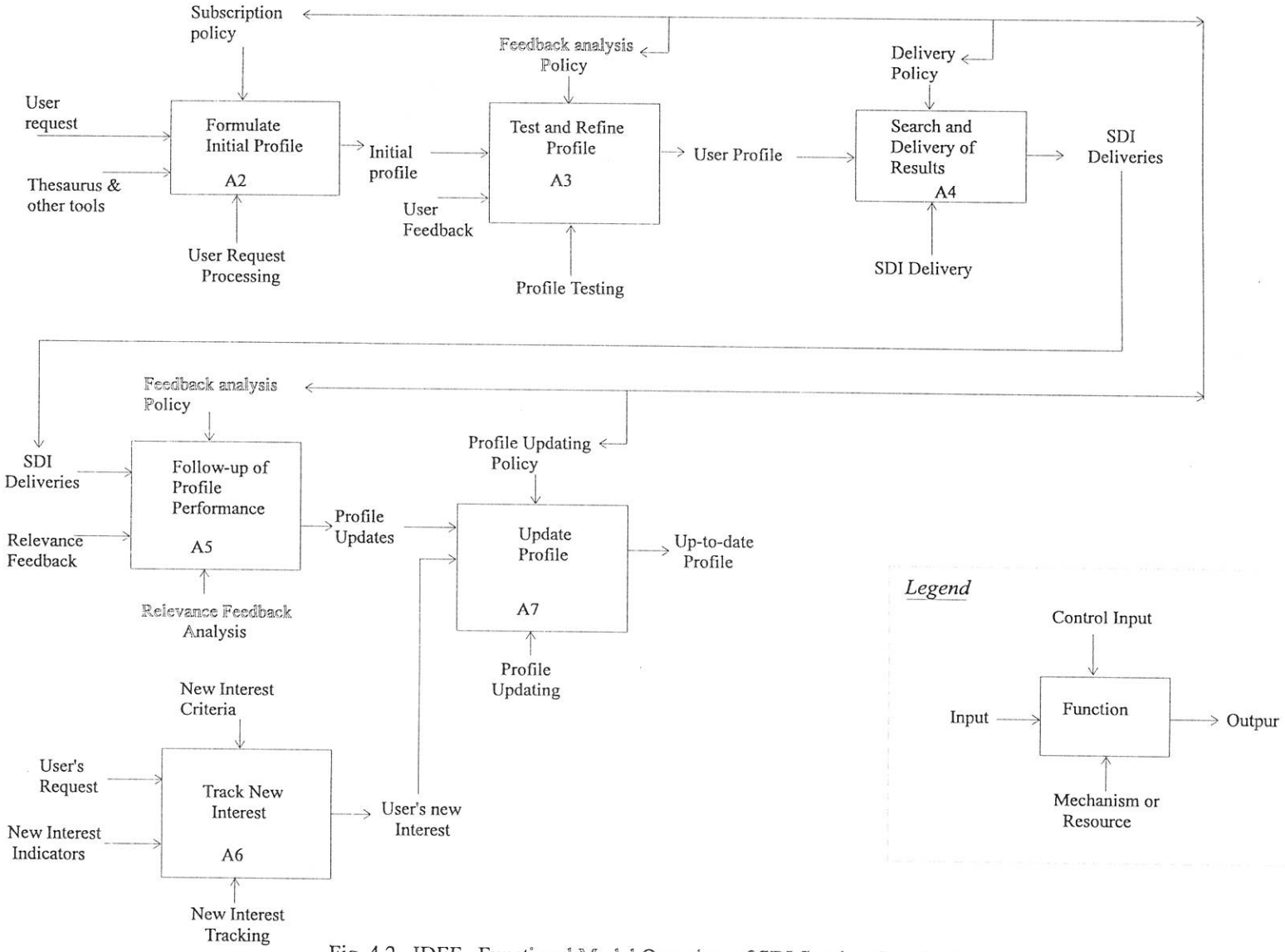


Fig. 4.2. IDEF$_0$ Functional Model Overview of SDI Services Level 1 Diagram

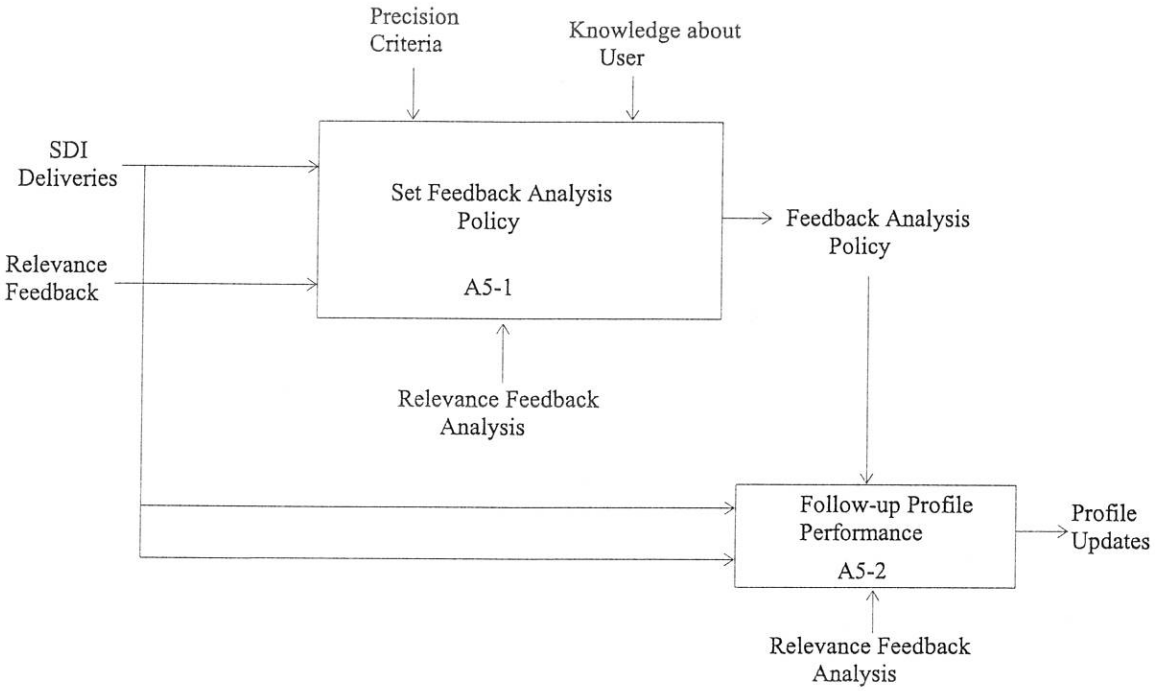


Figure 4.3. Follow-up Profile performance $IDEF_0$ Level 2 Diagram

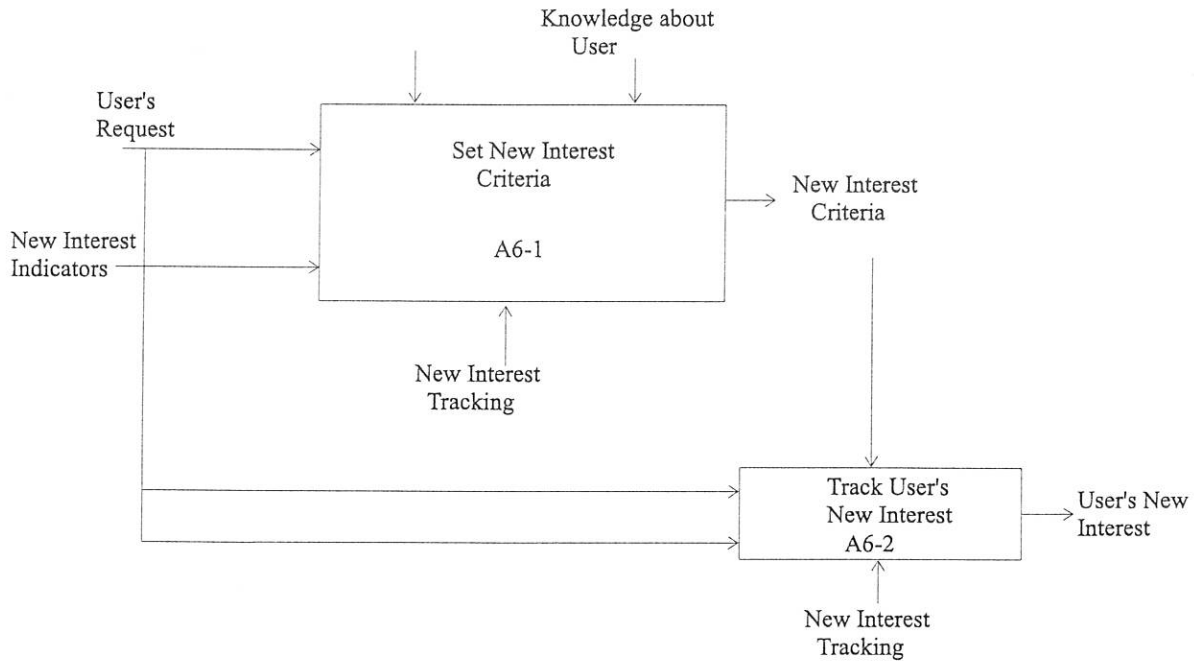


Figure 4.4 Track New Interest $IDEF_0$ Level 2 Diagram

Based on the above high level analysis, about five agents that could automate various decision making and monitoring operations were identified. These agents are named as *Browse and Feedback Agent* and *New Interest Tracking Agent* on the client side and *Search Profile Formulator*, *Filtering Agent* and *Delivery Agent* on the server side, as shown in figure 4.5. These basic agents may in turn be broken down into further specialist agents as appropriate.

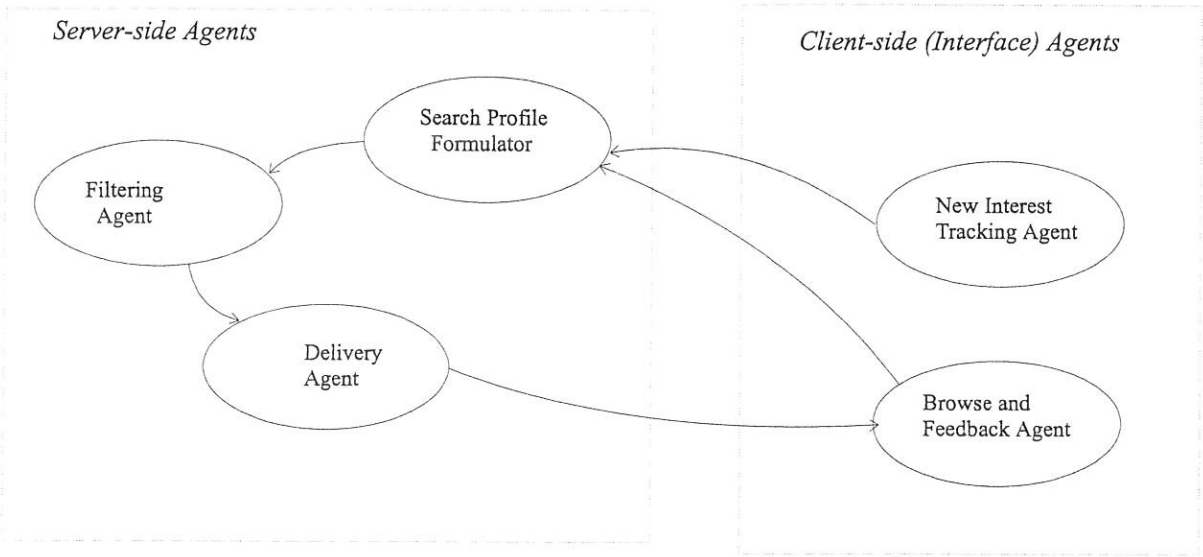


Figure 4.5. A Model for Agent Oriented SDI System

Although the client side agents are the main focus of the present study and thus dealt with in detail in the remainder of this report, a brief discussion of the server side agents is felt to be in order here. Although the client side agents are believed to address some of the aforementioned problems and enhance the contribution of the SDI service to the user's work, they do not actually filter and fetch the information. Rather this task is delegated to the server side which in turn could constitute the agents roughly sketched in Fig 4.5. As the actual work of filtering information based on user interest statements delivered by the client is done by this component, the whole functionality of the system partly rests up on it. There are currently a range of approaches and techniques that can be used to develop the

server side agents described above. It is believed that techniques employed by some of the existing systems (as reviewed in section 2.4), with the addition of automatic search formulation (for example, Yochum, 1996) and filtering methods (Foltz and Dummais 1992; Oard 1997), can be enhanced to the agents suggested. for the purpose of the current work, however, a human-intermediary-based server is assumed and all the outputs of the client-side interface agents are intended for utilization by the human intermediary.

4.4 The Client Side (Interface) Agent

4.4.1 Design Considerations

In the preceding few pages, the general framework for the implementation of agent based client/server SDI system was sketched. It has also been noted that the functioning of the client and the server depend on one another. However, due to the limited time available to the research, emphasis is given only to the client system and the agents that constitute it. Accordingly, this section provides a requirement as well as technical specification of the client system.

As shown in figure 4.5, two agents are identified on the clients side, viz. New Interest Tracking Agent, and Browse and Feedback Agent. The New Interest Tracking Agent functions in the domain of user's interest change tracking and predicts the likely interests of the user, as represented by keywords, through monitoring his/her job related documents. The Browse and Feedback Agent's domain is the provision of ranked display and user friendly browsing as well as feedback supplying interface and analysis of the relevance feedback thus obtained. The structure of the agents for the client system are shown in figure 4.6 below.

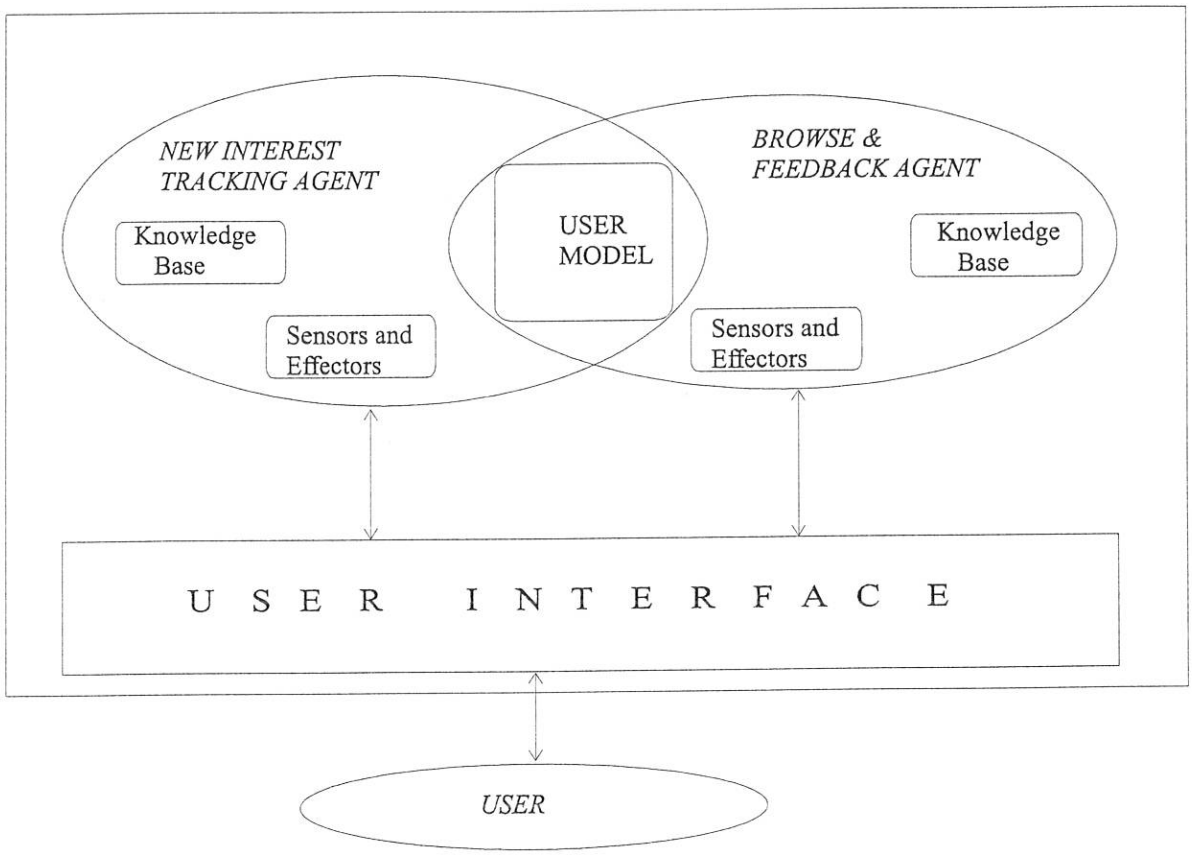


Figure 4.6. The Structure of the SDI Client

In general, the client system is an application that assists (and, as appropriate, acts on behalf of) an SDI service user in matters related to providing information needed by the SDI system to keep the profile representative of user's actual and current information needs. By doing so, it aims to reduce the burden of the user in keeping his/her profile optimized and up-to-date. By making the process of feedback provision and interest change reporting easy or automatic, it encourages or enables the user to make use of the SDI service more productively. By doing so such client also assists the SDI service providers: (1) by relieving the intermediary from the tasks of user-interest follow up, and (2) by providing input through scanning various sources affecting user-interest which are presently difficult to cover and hence abandoned. The client system can do these as it would be close to the user and will be able to draw information from various network

available resources related to users. Hence, the approach could be seen as "assigning one intermediary to work full-time with(or for) a user".

The philosophy of the SDI client extends the *passive subscriber model* of SDI systems that is proposed by Wyle (1996). Wyle describes a system based on the passive subscriber model as one that provides automatic information delivery and where

> *" the user passively receives information from the system, without having to re-submit queries every so often. The passive user model accommodates an interest drift, and a system employing the model automatically reminds its subscribers to update their profiles"*

In implementing the client system, agent oriented approach is considered since its functions involve pro-active behavior and automated decision making which qualify an application area for agent application (Kendall, et al 1995). Besides, the fact that the client is meant to serve as an assistant to users enables it to fulfill the agency criteria and would benefit if it is composed of agents each specializing in a specific area

The proposed agent-based client system is designed to include the following techniques and procedures which have been reported to perform better in various researches in information retrieval and software agents.

1) A *vector space model* based processing is employed for user interest modeling, user's interest tracking, ranked output etc. As such, the client is to serve as a vector space based interface to the existing Boolean based server. This approach is considered to take advantage of the various weighting and ranking schemes made possible using the vector space model. Although not exactly in the manner suggested in this work, vector space models have already been suggested and implemented in the SDI environment

(Wyle, 1996, Yan and Garcia-Molina, 1993). The review of the literature in the preceding chapter has also shown that representing profiles as word-vectors (bag-of-words) enables using machine learning and other advanced techniques.

2) Maintaining a *user model* on the client side is considered in addition to the search profile maintained on the server. The client side user model is designed to allow the maintenance of more extended information (such as user's current activities, preferences, etc. of the user) than kept in the present user profiles which are simply sets of search statements along with very few identification parameters like name and address. In the traditional SDI systems, much more additional information on the user is expected to be maintained in the minds of the intermediaries. As we are thinking of agents that perform most of these intermediary functions, they should maintain similar information on the user beyond the Boolean statements in search profiles. Indeed, the whole functionality of the agents is contingent on the availability of such model. Hence, a user model that is to be stored in the manner that the agents can make efficient use of is proposed. The agents are expected to keep the user model dynamically updated and make sure that the server side search profile is kept in synch with it. More will be said about the user model in section 4.4.4.

3) As has been pointed out in chapter two, one important function that is expected of the human intermediary, as far as user's follow-up and interest tracking is concerned, is proactivity and taking initiative. If the client system and its agents are to assist in the user's follow up and interest tracking functions, they too need to have these qualities. Hence, the agents are proposed to be *autonomous* and *proactive* in their functioning.

4) The proposed client system is also designed to include an *enhanced relevance feedback* analysis in addition to the presently practiced precision ratio calculations (as in the case

of ILRIAlerts). In particular, the relevant and irrelevant records will be further analyzed to optimize the user model, and hence, the search profile.

In addition to the above, the client system may also include many user supporting features like facilities for manipulation and printing of SDI results, for maintenance and retrieval of a mini-database by user out of SDI deliveries, etc.

4.4.2. The New Interest Tracking Agent

4.4.2.1 Agent Description, Goals and Plan

Although SDI services are meant to provide for a relatively stable and specific information need, this by no means implies a completely static user profile. In most cases, users are likely to have continuously changing interests in some specific areas while they may have a more or less stable interest in general information in other areas. If we take the interests of ILRI scientists, interview with some have revealed that their interests change from one research project to another and when research projects change their directions after their launching. Such change of interest has been well noted in the SDI literature. Wyle's (1996) Passive subscriber Model mentioned earlier also emphasizes that an SDI system should be able to track user's "interest drift".

The New Interest Tracking Agent is conceived based on the above thesis. This agent is designed to pro-actively and autonomously track user's interest changes with particular emphasis to newly growing interests prediction. To this end, just like the way done by human intermediaries, this agent employs the user's work (research) related documents as sources of new interest representing terms (key words).

By monitoring the various job-related documents (to which the user gave it the mandate), the New Interest Tracker Agent finds recurrent keywords, and depending on the threshold found in the user model and whether that keyword is not already in the users model, it suggests that keyword for the user. On the event that the user expresses an interest on the term, it automatically adds it to the user model (i.e. the agents model of the user). At the same time it also autonomously composes an e-mail and delivers it to the SDI server. On the other hand, if the user rejects a term that the agent suggests, then the agent takes note of that.

Although it is customary to set two types of confidence thresholds to agent, namely "the tell-me threshold" and "the do-it threshold" (Thomson 1995), the New Interest Tracking Agent is to operate based on "the tell-me threshold" model due to the inherent characteristics of term weighting process wherein, despite the effort to discard frequently occurring unnecessary terms, there still will remain terms that occur frequently but may not reflect new interest. This very nature of the domain will also prevent the agent from becoming completely trustworthy at any time in the future. But this does not mean that nothing can be done to enhance the trustworthiness of the agent. For instance, using a thesaurus that contains the likely keywords for topics that the user might be interested at all possible instances is assumed to be one way forward.

The New Interest Tracking Agent is based on the following basic assumptions.

- One obvious assumption is that the user of SDI system has either a personal computer or has access to a computer.

- It is also assumed that there exist job related information on the user's computer (or Local Area Network) like project proposals, project plans, user's publications, project databases, committee minutes, employee databases, etc. In the case studied in this thesis, that is ILRI , at least all internal users fulfill these requirements. Considering ILRI Information Service's decision that the SDI service will be offered only via e-mail after sometime, it is also highly likely that majority of users would prove this assumption tenable.
- The other basic assumption, which is also the basic plan of the agent, is a technical one that relates to what new interest means to the agent. Despite the effort made, no appropriate method that directly treats identification of terms/concepts that represent likely interests of a user from his/her documents has been found from the literature. Hence, it is hypothesized that a term is likely to represent new interest of a user if its occurrence frequency (and hence its frequency weight) is high in proportion to all the other terms found in user's job-related documents. Accordingly, the term frequency weighting scheme (Salton 1989) that is used for assigning indexing terms for a document is applied here for the whole collection of documents, as shown below, to predict new interest reflecting terms.

$$Weight_{ik} = \frac{Freq_{ik}}{NoWords_i}$$

$Weight_k$ is the weight of term k in document i, $Freq_{ik}$ is the frequency of term k in document i and NoWords is the number of terms in document i. Here, unlike the original frequency weighting that is used to determine indexing terms that represent a document, we determine terms that represent the whole collection rather than a document. And according to the above basic assumption, a term that represents the collection also reflects the users interest.

In addition, it is assumed that a term whose frequency of occurrence is highly (and/or evenly) distributed across larger number of documents is more likely to represent user's new interests than a frequency that is concentrated in few documents. However, the above frequency weight shows only the proportion of a term in the overall documents and doesn't show the distribution of its occurrence. For this purpose, the above weight is multiplied by the *document frequency(df)*. This approach is based on the term frequency/inverse document frequency (tf.idf) weighting but uses *df* (document frequency) factor, rather than *idf* (inverse document frequency) factor. This is because, unlike automatic indexing where "a high frequency term is acceptable for indexing purpose only if its occurrence frequency is not equally high in all documents of a collection" (Salton 1989), in our case, a term is acceptable for representing users interest if its occurrence frequency is equally high in all documents.

The usefulness of a term for our purpose increases with:

- The number of documents in which the term appears (df) contrary to the measure of the inverse document frequency where the interest is representing each document rather than all the documents relating to the individual as in this case. Here, the purpose is to assign high degree of importance to terms occurring evenly in most (or all) documents of the individual. This assumption seems to be valid as it is in accordance with the underlying assumption in probabilistic indexing theory which states that the best index terms are those that tend to occur in the relevant documents. In our case, the collection (set of user documents) consist of nothing but relevant documents for the user.

- The ratio of the proportion of the frequency of occurrence of the term to the proportion of the total frequency of terms occurring in each document.

Hence, the above formula becomes:

$$W_{ik} = \frac{f_{ik}}{\sum_{k=1}^{NoOfTerms} f_{ik}} * df_k$$

where W_{ik} is weight of term k in document i; $\sum_{k=1}^{NoOfTerms} f_{ik}$ is total number of terms occuring in document i; dff is number of documents in which term k appears anf f_{ik} is frequency of occurrence of tem k in document i.

One important consideration in implementing the above weighting scheme is the fact that, unlike the cases in automatic indexing and IR systems, the user's job-related documents are produced on a continuous basis. To handle this a *temporary term accumulator file* in which the agent accumulates potential terms found in previously scanned documents but have not yet met the New Interest Threshold (specified in user model) with their hitherto frequencies. The frequencies of terms in every new document are merged with this pending vector and resulting term weights are compared with the New Interest Threshold by the agent.

One problem this approach of employing temporary accumulators could run into is what can be called the *pending limit problem*. How long should a term be kept in the temporary accumulator to meaningfully reflect user interest ? One promising approach in addressing this problem is to study the average period of time required by the life cycle of user's interest, that is from first occurrence to maturity and then to obsolescence. For example, the average period of time a user will be working on a project can be considered. More

software based solutions like a facility that monitors how long a term has been in the temporary accumulator since its weight has last been updated can also help. In any case, it sounds logical to allow the user specify this duration and the agent execute it automatically.

The other important and at the same time challenging factor in the functionality of the New Interest Tracking Agent is how to determine a *threshold* that enables the agent trace user's new interest. For a user, a suitable threshold depends on a number of factors like the pattern of occurrence of interest representing terms, the amount of interest-related and non-interest related terms in documents, the user's tolerance level of irrelevant terms suggested, etc. In general, the use of an empirically identified fixed threshold weight or a dynamically updated threshold can be considered. One approach believed worth considering for this purpose is taking a certain variation from the *mean weight* of terms in the temporary accumulator.

Two approaches were identified for consideration in this connection

1. Term-Term similarity using the concept of term centroid suggested elsewhere (Salton, 1989) for automatic thesaurus construction
2. Simple statistical techniques based on the distribution of standard term weights

The first approach is based on a set of term vectors of the type shown in the matrix below

	T_1	T_2	T_3	.....	T_n	T_c (term centroid)	T_{sd}
D_1	w_{11}	w_{12}			w_{1n}	t_1	s_1
D_2					w_{2n}	t_2	s_2
.					.	.	.
.					.	.	.
Dn						t_n	s_n

where, D_i represents the various documents of the user, w_{ik} the weight assigned the the k-th term in document *i*.

Then it is possible to use a similarity function defined in below, to obtain a similarity measure between pairs of columns, reflecting the similarity between the terms.

$$Sim(T_k, T_h) = \frac{\sum_{i=1}^{n} t_{ik} * t_{ih}}{\sum t^2_{ik} + \sum_{i=1}^{n} t^2_{ih} - \sum_{i=1}^{n} t_{ik} t_{ih}}$$

To reduce the number of pair-wise similarity computation required, one may use the concept of centroid (as suggested by Salton, (1989)). A term centroid can be defined as average vector of the term vectors. That is T_c , the centroid whose element t_k is computed as the average value of all the values of term *k* in the individual documents for the person.

$$\bar{t}_k = \frac{1}{m} \sum_{i=1}^{n} t_{ik}$$

The similarity can now be computed as the similarity between each term vector and the centroid. Then, it is possible to use the similarity so computed to identify terms which are very similar to one another and the centroid (user interest representation). That is terms for which the similarity to the centroid is largest may be considered for suggestion to the user ranked by the similarity values. However, this method is not implemented here for reasons of avoiding unnecessary computation overhead, which was observed in the attempt made at the initial stage.

The second approach is based on applying simple statisitical techniques to the weights observed/computed for each term, as described below. This is done at steps as follows:

1. First for the purpose of computing the cummulative weight of each term in the entire collection (user's domain), the weights computed are transformed to standardized values by using Z score. That is, by applying the formula:

$$\bar{W}_{ik} = \frac{W_{ik} - \bar{t}_i}{S_i}$$

where w_{ik} is the computed weight of term k in document i

$\bar{t}_i$ is the mean, and

S_i is the standard deviation as shown.

$\bar{W}_{ik}$ - the standard weight.

This will result in the following matrix:

	T_1	T_2	.....	T_m
D_1				
.				
.				
D_n				
	tw_1	tw_2		tw_m

1. The standardized weight for term k are then totaled (as shown in the above figure) to give the cummulative standard weight of the term in the entire collection - the significance of the term for the user.
2. While simple ranking may then be used to determine the most significant terms, still for the purpose of systematizing the choice of the threshold beyond which terms are selected for suggestion to users, the mean and standard deviation of the cummulative standardized weight of the terms is computed. Once this is done, the threshold is determined by using the newly computed mean and standard deviation. For instance, the most obvious (thrivial) case is to consider those terms whose cummulative standard weight are 2 or 3 standard deviation away from the mean. The results of tests done using this method are reported in the next chapter.

4.4.2.2. Agent Sensors and Effectors

As has been shown in the agent architecture above, the new Interest Tracking Agent interacts with the user, job related documents and e-mail system. It interacts with the user through the client interface while it monitors the user job related documents by accessing

the computer files they are stored in. It also interacts with the e-male client so as to store to the mail boxes for delivery. The following are the suggested basic sensors and effectors.

Sensors: . Detect new file in directory

Effectors: . Scan file for potential key words
. Inform the keyword to user
. Update user model
. Compose e-mail
. Deliver e-mail

4.4.2.3 Agent Architecture

The architecture of the New Interest Tracking Agent that shows its capabilities, process flows and its interactions with the user as well as other systems is given in figure 4.7.

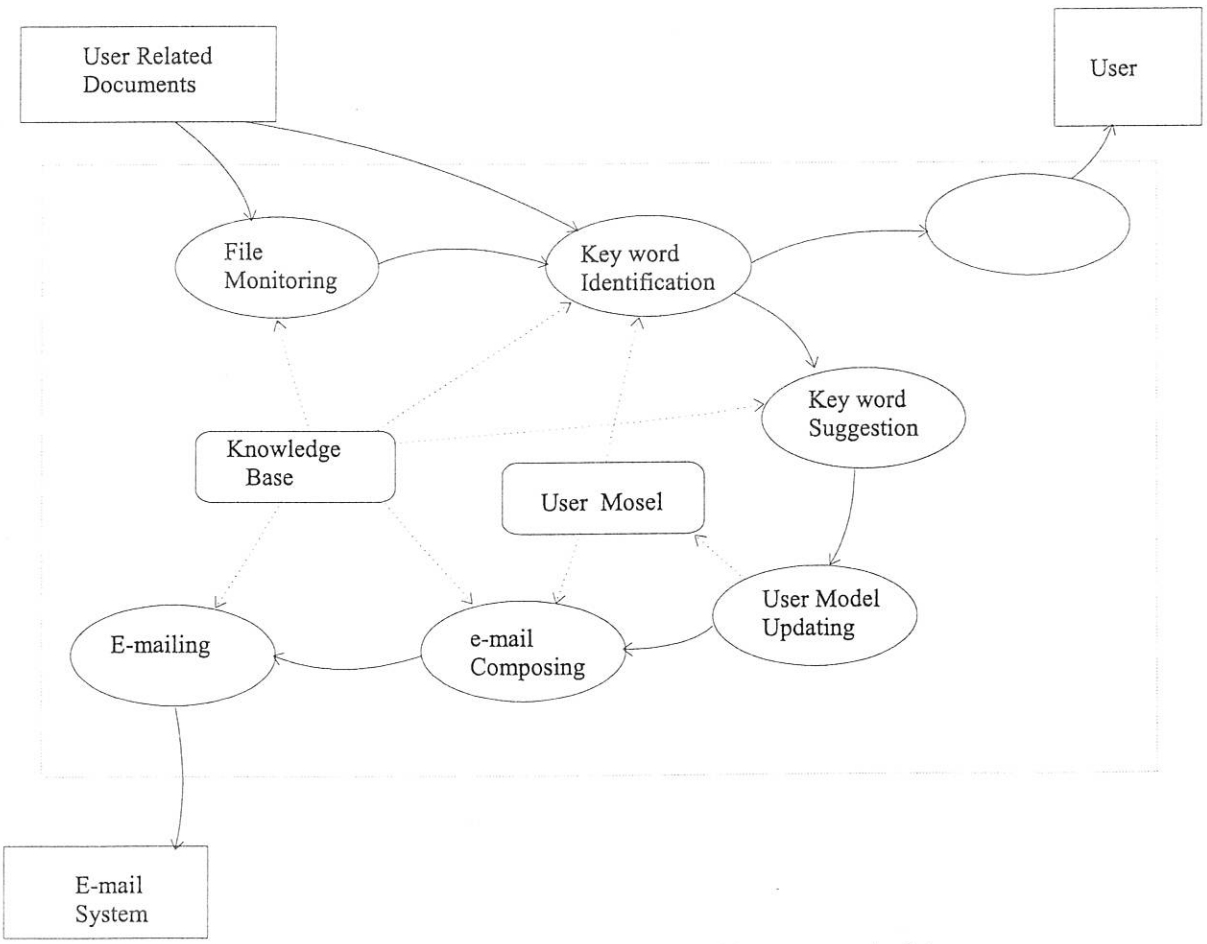


Figure 4.7. New Interest Tracking Agent Architecture

4.4.3. The Browse and Feedback Agent

4.4.3.1 Agent Description, Goals and Plans

On the importance of relevance feedback analysis in SDI systems, Lewis (1992) remarked that "Relevance feedback can be used in text routing [i.e. SDI] and has the potential for being more effective than in text retrieval since the information need persists over a long period of time." Accordingly, provision of feedback analysis component in the SDI client is a basic requirement.

The Browse and Feedback Agent is an interface agent that provides the user with friendly interface to browse and provide feedback on the SDI deliveries received from the server. This agent does what any online client database interface does and more. Mainly it uses the user model to rank the SDI deliveries according to relevance to user based on a ranking algorithm. It also receives users relevance feedback, analyze it and suggest modifications to the user model based on the analysis.

In particular, the Browse and Feedback Agent has three major goals:

1. Rank and display the SDI search results delivered by the server.
2. Analyze relevant document records and identify key words that can be used to optimize the user model. The optimization can be adding the new keywords, replacement of existing key words, or narrowing or broadening of the existing keywords, etc.

3. Analyze irrelevant records and identify keywords that must be removed from user profile or replaced by another keyword. In the same manner, it also identifies keywords and topics on which the user is no more interested in and informs the same to the SDI server.

(a) Relevance Ranking

There are various functions (algorithms) reported in published literature that may be considered for the purpose of ranking results periodically received from the SDI server. For the Browse and Feedback Agent, the statistical model that orders the records according to a statistical similarity between the user model and the vector representation of each record is proposed. This approach has been established by previous researches as relatively effective and is widely implemented in systems like wide-area information system (WAIS) world-wide web robots and online retrieval systems (Lee , Chuang and Seamons, 1995).

A common Similarity measure, known as the *cosine measure,* measures the angle between the document vector and the query vector (i.e. the user model) when they are represented in a V-dimensional Euclidean space, where V is the vocabulary size (Salton and McGill, 1983). This is one approach that can also be implemented in the Browse and Feedback Agent. Precisely, this ranking function defines the similarity between a document D_i (in our case a record in the SDI delivery) and a query Q (in our case the user model) as (Lee, Chuang and Seamons, 1995):

$$Sim(Q, D_i) = \frac{\sum_{j=1}^{v} w_{Q,j} * w_{i,j}}{\sqrt{\sum_{j=1}^{v} w^2_{Q,j} * \sum_{j=1}^{v} w^2_{i,j}}}$$

where $W_{Q,j}$ is the weight of term *j* in the user model which is defined in the same way as $W_{i,j}$ (i.e. weight of the term). The denominator, called *normalization factor,* discards the effect of record lengths on document scores. As the length of the records considered by the Browse and Feedback Agent doesn't normally vary greatly, the performance of this ranking function is expected to be relatively free from the problem caused by length variations.

There are various implementation methods for the above function as described by Lee, Chuang and Seamons (1995). For the present agent, one of them that approximates the effect of normalization, which has been reported to perform well for concept queries (to which the user models can be categorized), both in terms of their accuracy and computational efficiency by the same authors is considered. This normalization factor is also much easier to compute than the original one given above. The formula for this approach is given as:

$$Sim(Q, D_i) = \frac{\sum_{j=1}^{n} w_{Qj} * w_{ij}}{\sqrt{numberofterm \sin D_i}}$$

(b) Feedback Analysis

The Browse and Feedback Agent employs current methods of feedback analysis and feedback based learning. The use of relevance feedback algorithms for agent development has been employed by many researchers (Edward's *et al* 1996; Balbonovic and Shoam 1995; Pannu and Sycara ; Armstrong, *et al* 1995).

By analyzing the relevance feedback given to documents, the agent analyzes the relevant and irrelevant documents to identify potential additional keywords to the user model and keywords that should be discarded. To this end, the Cornell 'ltc' weighting that is commonly used with the vector space model of text retrieval is proposed. Besides, the Browse and Feedback Agent does a simple precision ratio calculation to supply for the server to be used for evaluation of the general performance of the SDI server. This weighting scheme, as given by Ittner, Lewis and Ahn (1995) is as follows:

$$w_{ik} = \frac{tf_{ik} * \log(N_D / n_k)}{\sqrt{\sum_{j=1}^{t} (\log(f_{ij}) * \log(N_D / n_k))^2}}$$

where N_D is the number of documents in the *training set* (based on which the agent learns the prototype vector), N_K is the number of documents in which term *K* appears, and tf_{ik} is:

$$tf_{ik} = \begin{cases} 0 & if \quad f_{ik} = 0 \\ \log(f_{ik}) + 1 & Otherwise \end{cases}$$

where f_{ik} is the number of occurrences of term K in document i.

Based on this weighting, the *Rochio algorithm* is suggested to learn from each SDI delivery and come up with a vector (called *prototype*) that is suggested to the user, modified and used to update the user model (which automatically entails its delivery to the server). This algorithm is chosen as it is widely used in the development of information retrieval agents (for example, Armstrong, et al 1995; Lang 1994; Edwards, et al ; Balabonovic & Shoham 1995) and as it is reported to work well. In our case, the records periodically received by the user are used as the *training set* from which good and bad

examples are taken for learning. The database on the server side on which the searching is done is considered as the *test set* for the prototype produced by Rochio algorithm. Hence, what is learnt from the delivered records is used to update the user profile (and search profile) and '*tested*' on the database. Therefore, each SDI delivery will be a new learning source for the Browse and Feedback Agent.

The Rochio algorithm as given by Ittner, Lewis and Ahn (1995) specifies that the weight of term K in the prototype Q_C for class C should be

$$tf_{ik} = \begin{cases} w_{ck}' & if \quad w_{ck}' > 0 \\ 0 & Otherwise \end{cases}$$

where

$$w_{ck}^{i} = \beta \frac{1}{|R_c|} \sum_{i \in R_c} W_{ik} \quad - \quad \gamma \frac{1}{\left|\bar{R}_c\right|} \sum_{i \in \bar{R}_c} W_{ik}$$

R_C is the set of training documents belonging to class c and $\bar{R}_c$ are the documents not belonging to class C. The parameters β and γ control the relative impact of positive and negative examples on the prototype. For these, the standard values β=16 and γ=4 are suggested by the same authors.

Although normally the prototype that results from the Rochio algorithm may directly be used to compare with the test set (i.e. the database), in the current agent, the terms instead are suggested to the user. This is made mainly because the matching with database is Boolean and because it is important that users approval should be received before a term is sent to the SDI server.

4.4.3.2 Agent Sensors and Effectors

The following are the suggested basic sensors and effectors.

Sensors: . Detect arrival of SDI delivery

Effectors:
. Rank delivery
. Display to user
. Analyze feedback
. Inform required updates to user
. Update user model
. Compose e-mail
. Deliver e-mail

4.4.3.3 Agent Architecture

The architecture of the Browse and Feedback Agent that shows its capabilities, process flows and its interactions with the user as well as other systems is given in figure 4.8.

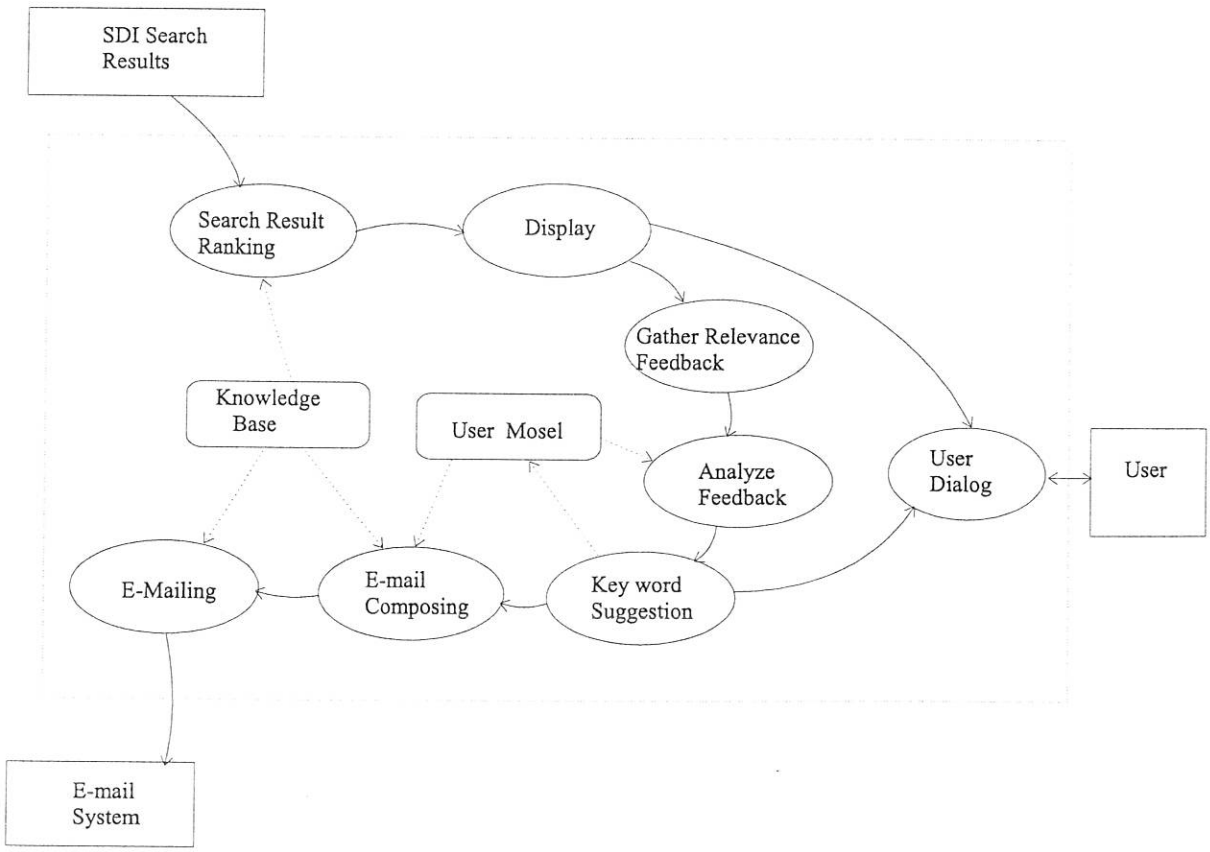


Figure 4.8. Browse and Feedback Agent Architecture

4.4.4. The User Model

The user model is the core of the interface agents functions. It is from where the agents get the preferences of their user. As such, the user model constitutes the agents' basic knowledge that they use for inferencing.

As has been mentioned earlier, the client side user model is to contain more information on the user than simple Boolean statements (as is the use with current SDI systems). The aim of this model is for use in agent inferencing rather than matching with database (for which the Boolean search profiles are to continue being used). Hence, the content and format of the model is to be determined by its level of support of inferencing operations of the agent.

The types and extent of information to be maintained in information retrieval agents in general and interface agents in particular is a subject of intense research now days. Particularly in the SDI and filtering area, there are works like that of Stadnyk and Kass (1992) which are specifically devoted for this subject. Consequently, a number of suggestions have been proposed. For the presently proposed SDI client, too, coming up with a complete user model is a task that probably requires one or more full-fledged researches. Hence, in this thesis, only a minimal user model that is thought to meet the requirements of the prototype to be developed is suggested. Broadly, this user model would have the following categories of information:

1. User's activity (General and Specific)
2. User's workstation (system)
3. Agent Services requirements
4. Vector of interest representing terms

Keeping the user model dynamically updated is the responsibility of both agents. Hence, it is accessed by both agents. Any addition, deletion or modification to the user model is mainly under the approval of the user although the agents also could autonomously make some updates like changing the weight of terms.

4.4.5. Interaction With User

It is important that the interface agents are transparent to the user. To this end, appropriate interfaces should be provided for the user to monitor agent operations and modify information held by them.

CHAPTER 5

THE EXPERIMENT

5.1 General

As has been discussed in the preceding chapter, the agent-oriented modeling has adopted many procedures and algorithms from previous works and has suggested some novel approaches. This chapter reports the attempts made to experience and demonstrate how some of these techniques can be implemented and test the performance of same.

5.1.1. Experimental Objectives and Scope

The objective of the experiment was to demonstrate and test the viability of the agent-based SDI client model described in the previous chapter. In particular it focuses on implementing and testing the fundamental components and algorithms of the agents rather than developing a fully functional agent. Accordingly, minimal interfaces and features are provided. Due to time limitation, only the New Interest Tracking Agent has been prototyped. This agent was selected because its design involves numerous assumptions that require empirical justifications unlike most of the techniques employed by the Browse and Feedback Agent which were the subject of a number of researches. This thesis does not aim to conduct a complete experiment on the proposed procedures and assumptions. However, it has the interest of uncovering some indicators on whether they are promising or not. Since the whole research work is primarily an academic exercise, the major focus of the experiment was on learning how agent methodologies can be applied to solve an information problem in general and SDI user profile management problems in particular.

5.1.2. Procedures and Tools

The prototype for the New Interest Tracking Agent was developed using IBM's Agent Building Environment (ABE) Developer's Toolkit, Level 6, (IBM, 1997). For the development of the prototype, the approach that Bischofberger and Pomberger (1992) referred to as *Experimental Prototyping* whose purpose is to "experimentally validate the suitability of system component specifications, architecture models, and ideas for solutions for individual system components" is employed.

Once the prototype agent is developed, various tests were made using a test collection as will be described in section 5.4. So as to assist further analysis of agent results, procedures that may otherwise not be needed in an agent that serves a user are added. Furthermore, statistical methods and applications have been used to analyze the properties of the test collection to compare the performances of different term weight thresholds and to analyze the suggestions of the agent.

5.2. Overview of the ABE Developer's Toolkit

IBM's Agent Building Environment (ABE) is a toolkit for software developers that is meant to be used to build an application based on intelligent agents or to add (embed) agents to an existing applications. It is basically aimed at providing a standard architecture and platform for agent development and consequently is based on internationally accepted components.

Basically, ABE provides a structure to build agent components that are constructed at run-time based on configuration selections in a configuration file as well as some basic components of agents. The generic architecture of the ABE includes components for receiving events, sensing specific conditions based on interactions with external resources and applications, and then taking autonomous actions. For this, the toolkit's architecture encompasses the following three elements:

1) *Adapters* - Adapters provide the direct interfaces of the agent with applications. Adapters serve as eyes, ears, and hands for specific agents.

2) *Engine* - An agent engine processes the incoming information according to predefined criteria and produces specific results. The ABE level 6 includes a rule-based forward chaining inference engine called the RAISE inference engine, which is based on technology from the T. J. Watson Research Lab of IBM.

3) *Knowledge* - The knowledge represents the rules, observations, conditions, and actions that the agent engine will perform. Such information are stored in individual Libraries that allow the persistent storage of rules and facts in hierarchical structures. The engine uses these rules and facts to intelligently guide the agent.

Development or embedding of agents using IBM basically involves development of *adapter* and *agent control* using either C++ or Java and building the inference materials (i.e. *rules* and *long-term facts*). Using the internationally standard format for storing agent's internal knowledge called *Knowledge Interchange Format* (*KIF*). Then these

components are integrated with the engine and, if necessary, with the basic adapters provided with the toolkit to form the agent.

5.3 The prototype

Based on the model given in the previous chapter, the prototype New Interest Tracking Agent was developed to monitor specific directories on user's machine, automatically trace when new document files are added and scan those files for terms that are likely to represent new interest of the user. This agent determines whether a term is likely to represent new interest based on a threshold which, for this experiment, is found by analyzing a set of documents as will be described in section 5.4. Then, when the term it suggested is approved, it composes and delivers an e-mail to the server.

This agent, when implemented using the Agent Building Environment, takes the following structure:

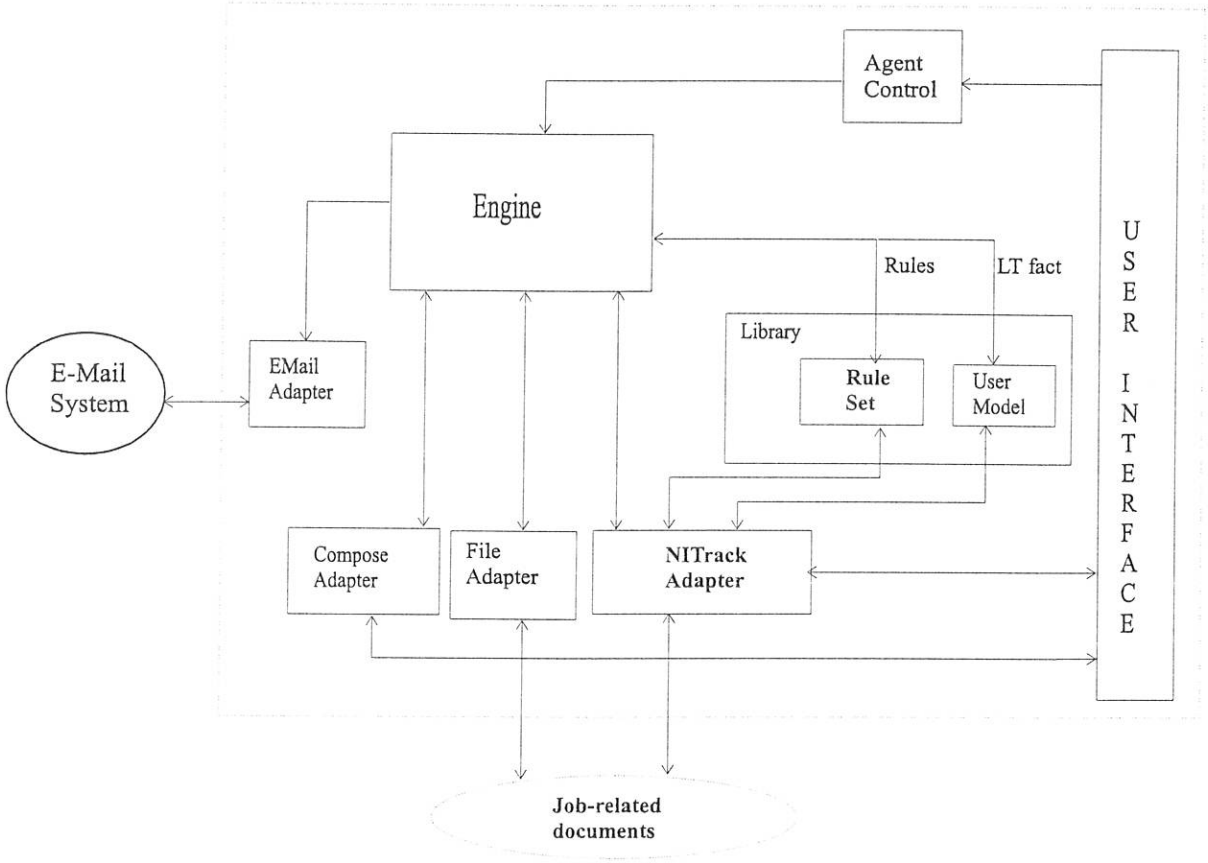


Figure 5.1. The New Interest Tracking Agent Inplementation Structure

From the components forming the above structure, some are provided by ABE while the others are developed. A brief description of ABE provided components that are integrated into the New Interest Tracking Agent follows:

1. *The Engine* - The RAISE inference engine is a forward-chaining inference engine that can be configured in an agent to attach to any number of adapters. Adapters exist outside of RAISE. The RAISE performs inferencing using the supplied conduct sets that consist of rules (RuleSet(s) and optionally long term facts (LTFactSet(s)).

2. *The File Adapter* - The File adapter allows an agent to observe and manipulate files to which it has access. This can include files that reside on the same system as the agent, and also files on any file servers to which the agent has access. In the New Interest

Tracking Agent, this adapter has been integrated to automatically scan the directories where user stores his job-related document files on his machine.

3. *The Time Adapter* - The time adapter deals with the domain of time. It provides information about time and performs operations with time such as ordering and arithmetic operations. It provides the capability of setting timers or alarms which will subsequently generate Trigger Events. This adapter has been integrated to control the interval at which the New Interest Tracking Agent should scan directories.

4. *The Mail Adapter* - It is a very basic sample adapter that sends email. It establishes a TCP/IP SMTP (Simple Mail Transfer Protocol) dialogue with a relay host to accomplish this . This Adapter has been integrated to automatically deliver the e-mail that contains the terms user has indicated relevant to the server.

The remaining components, namely the Agent control, rules sets and user model, the NITrack Adapter, Compose Adapter and a simple user interface have been developed. Each of these provide just basic functionalities as described below.

5.3.1 Agent Control

Although ABE provides a general purpose Agent control component that is written with Visual Age C++, writing a separate simple control was preferred for the sake of flexibility and extendibility. Besides, developing a new Agent control was opted as it is not possible to recompile the given agent control with additional components as it uses libraries which are not given with ABE Developer's Toolkit.

The Agent control is inherited from the *IAAgent* class of ABE and is responsible for the initialization and overall control of the running agent. By reading and parsing configuration selections in a configuration file, it builds the agent at run time by loading, integrating and initializing each component. The configuration file shown in table 5.1 includes statements that define the parts constituting the agent which are the engine, adapters, library and conduct sets (i.e rules and long term facts for use in inferencing). Each component is described by a name that uniquely identifies that component and the name of the dynamically loaded DLL module containing the executable code of that component. Accordingly, as shown in the table , the NITrack and Compose adapters written for this experiment are compiled as DLL and specified in the configuration file.

Table 5.1.: Agent configuration file

```
* Configuration file for NITA
*
* Engine domain and DLL name
Engine    RAISE iagerais
*
* Adapter domains and DLL names
Adapter   Time iagatime
Adapter   SampMail sampmail
Adapter   File iagafile
Adapter   NewInterest nintrest
Adapter   Compose mcompose
*
* Library loadable module name, map, and top collector name
Library file iaglibf . .
*
* Engine type and conduct set name to associate with it
Conductset RAISE nitrack
Conductset RAISE usermodl
```

The full source code and header file of the Agent control is attached as Annex 1.

5.3.2 The Rule Sets and User Model

Once the agent is loaded and initialized by the agent control, it is the inference Engine which takes control of the agent. First the engine requests each adapter to register its effectors and sensors with it calling their *identify()* member function. Then it receives triggers from adapters and uses the rule set and long term facts, called *conduct sets*, to do inferences and initiate the relevant actions to be performed by the adapter by calling their *performAction()* member function for effectors and *testCondition()* function for Boolean sensors (the codes for these function in the NITrack adapter and compose Adapter will be discussed in the following sections). In other words, the engine, based on the rules available to it, coordinates the various components of the agent. In addition to the rules, the engine uses loaded long term facts for inferencing. The Rule set that is used by the engine of the New Interest Tracking agent is the following:

Table 5.2. Rule set for New Interest Tracking Agent

```
1
ntracker1
A

SetAlarm
A
(=> (EventName "AGENT:STARTING") ("SetIntervalAlarm" 60  "minutes"))

ConfigureMail
A
(=> (AND (AND (EventName "AGENT:STARTING") (RelayHost ?host))
      (EMailfrom ?from)) (AND (EMailRelayHostIs ?host) (EMailFromAddressIs
      ?from)))

ScanDirectories
A
(=>(AND (EventName "Time:IntervalAlarm") (Interval 60 "minutes"))
     CheckDirectory ?directory "*.*" "ALL"))

InitiateDocScan
A
(=>(AND (AND (EventName "File:FileCreated") (ShortFileName ?fname))
     (DirectoryName ?directory)) (NITracker:WeightTerms ?fname))

ComparisonToThreshold
A
(=>(AND (EventName "NITracker:WeightingFinishedEvent")
    (NewInterestThreshold ?threshold))
     (NITracker:FindTermOfInterest ?threshold))

SuggestiontoUser
A
(=> (AND (AND (AND (EventName "NITracker:TermFoundEvent")
     (TermName ?term ?weight)) (CheckInUM ?term "notfound"))
     (NITracker:ImplementDecision "suggest"))

DontSuggestToUser
A
(=>(OR (AND (AND (EventName "NITracker:TermFoundEvent")
    (TermName ?term ?weight)) (CheckInUM ?term "found"))
     (NITracker:ImplementDecision "dontsuggest"))

InitiateComposer
A
(=>(AND (EventName "NITracker:UMUpdatedEvent") (TypeOfUpdate ?type))
    (Composer:Compose ?type))

InitiateEMail
A
(=> (AND (EventName "Composer:MessageComposedEvent") (TestIs ?text))
     (EMailWithsubject "ILRI&telecom.net.et" ?text "Newinterest from user" ?from))
```

As can be easily observed from the above rules, most of the functions that are done using rules in this prototype (for example, checking whether a term already exists in the user

model) can also be done using common procedural approach. This is due to the fact that the current agent is a prototype that implement few functions.

It has been mentioned in the previous chapter that the agent maintains a user model that it uses to store learnt user model. This user model is implemented here as a long term fact set although in full-fledged agent a model that also includes rules is envisaged. A sample user model that is used by the New Interest Tracking Agent is given in table 5.3

Table 5.3. Sample User Model

```
1
usermodel
User Model Persistently Stored as Long Term Fact Set
A
UserName
A
(UserName "Dr. W. Thrope")

GeneralInterestName
A
(GeneralAreaOfInterest "Animal Science" "Livestock Production")

CurrentProjectsName
A
(CurrentProject "Research methodologies for dairy systems" "Constraint analysis for
dairy
   systems")

PlannedProjectNames
A
(PlannedProject "Cattle Research Network Coordination (CORAF) Livestock Network")

DirectoryToMonitor
A
(DirectoryName "c:\myfiles\projects\" "c:\myfiles\plans\")

InterestThreshold
A
(NewInterestThreshold  0.0024)

RelayHostforEmailAdapter
A
(RelayHost "telecom.net.et")

EmailAddress
A
(EMailFrom "w.thorpe.@cgnet.com")

SDIProviderAddress
A
(EMailto "ilri-information@cgnet.com")

TermsOfInterest
A
(WeightOf  "cattl" (real 0.0109664))
(WeightOf  "dairi" (real 0.0452365))
(WeightOf  "farm"  (real 0.0109664))
(WeightOf  "livestock" (real 0.0219328))
(WeightOf  "market"  (real 0.00959561))
(WeightOf  "methodologi" (real  0.0123372))
(WeightOf  "milk" (real 0.0109664))
(WeightOf  "polici" (real 0.013708))
(WeightOf  "product" (real 0.0246744))
WeightOf  "smallhold" (real 0.0246744))
```

5.3.3 The NITrack Adapter

The NITrack adapter whose source code is attached as annex 3 is the core of the New Interest Tracking Agent that actually scans users job related documents for terms representing user's information interest. As indicated in the rules above, one effector of this adapter (i.e *WeightTerms*) is called when the *InitiateDocScan* rule is fired by the *FileCreated* event from the File Adapter. In addition, this adapter provides the *FindTermOfInterest* and *ImplementDecision* effectors as well as the *CheckInUM* sensor. This adapter also includes a procedure to register its effectors and sensors to the engine called *identify()*. Its also includes the *performAction()* function which the inference engine calls to pass effector and sensor service requests.

As mentioned before, when a new document file is detected by the File Adapter, the *WeightTerms* effector is called. This in turn causes the adapter to launch its *weightTerms()* function. The *weightTerms()* function, as is common in vector space based systems, first removes the stop words (words so common as to be useless as new interest keywords, like *the, and*) from the document and then stems the remainder of the words. The later reduces words to their "stems" (or 'roots') and thus diminishes redundancy. For instance, *farmer*, *farmers*, *farming*, farm, *farmed* all reduce to *farm*.

In this prototype, Frakes (1992) implementation of the porter suffix striping algorithm (Porter 1980) has been used. This algorithm was selected as it was reported to be used in related studies (for example Balabanovic & Shoham 1995). For stop word removal, a stop list filter finite state machine generator implemented by Fox (1990) as well as the stop word list provided with it have been used. The ANSI C implementations of these algorithms were integrated to the NITrack adapter with some effort and modification.

Once the terms in the stop words are removed and remaining terms stemmed, the frequency of occurrence of each term is counted and a term frequency vector generated. Then, as discussed in the agent model in chapter 4, this term frequency vector is merged with previous vector (tempacum.dat), if any, that is composed of terms found from previously scanned documents but have not yet met the required threshold. For the merging of these two vectors, the single loop consequential merging algorithm (Folk and Zoellick, 1992) has been implemented. Along with the merging, the weight of each term is recalculated using the weighting formula from the preceding chapter.

Upon the merging process (or updating of temporary accumulator) *df* is added by one, the two *tf* values are added and the total number of words in new vector are added to the wither to total words maintained at the first line of the tempacum.dat vector). The following code block performs these processes:

```
while (MORE_VECS) {
        if (strcmp(docVector->term, accumVector->term) < 0) {
                docVector->df += 1;
                WriteAcumVec(docno,aFile, docVector, tptr);
                MORE_VECS = ReadVecElement(dFile, docVector);
        }
        else if (strcmp(docVector->term, accumVector->term) > 0) {
                WriteVecElement(aFile, accumVector);
                MORE_VECS = ReadVecElement(cFile, accumVector);
        }
        else {
                // add frequency,recalculate weight and write to accumFile
                accumVector->doctf += docVector->doctf;
                accumVector->df += 1;
                WriteAcumVec(docno, aFile, accumVector, tptr);
                MORE_VECS = ReadVecElement(dFile, docVector);
                MORE_VECS = ReadVecElement(cFile, accumVector);
        }
}
```

When done with merging, the NITrack adapter generates the *WeightingFinished* event to the engine which, consequently, fires the rule that gets the *NewInterestThreshold* from the user model and calls the *FindTermOfInterest* effector of the NITrack adapter. This causes a call to the *findTermOfInterest()* member function. This function compares weight of each vector element in temporary accumulator (in tempacum.dat) with the threshold and, if it finds term (s) that meet the threshold, it causes the generation of the *TermFound* event by the NITrack adapter to the engine. This, in turn, causes a call to the *CheckInUM* sensor of the NITrack adapter. This sensor call causes the NITrack adapter to start its *checkInUM()* method that checks whether the term is already available in the user model. After checking, it responds "found" or "not found". If the term is not found in the user model, the rule that causes a call to *ImplementDecision* effector that suggests the term to the user is fired. The effector is also provided by the NITracker through its *ImplementDecision()* method.

In cases when the user accepts a term suggested by the New Interest Tracking Agent, the NITrack adapter adds that term to the user model and triggers the *UMUpdated* event. This event fires the *InitiateComposer* rule that causes the engine to send an effector call to the Compose adapter.

5.3.4. The Compose Adapter

The Compose Adapter composes an e-mail based on terms that the user approved as relevant to his information interest. Currently this facility is included just for demonstration purpose and does not do much enhancement on the terms except adding some common headings, etc. This component is designed as a separate adapter so as to allow future enhancements to the terms that might be needed by the server.

When done with merging, the NITrack adapter generates the *WeightingFinished* event to the engine which, consequently, fires the rule that gets the *NewInterestThreshold* from the user model and calls the *FindTermOfInterest* effector of the NITrack adapter. This causes a call to the *findTermOfInterest()* member function. This function compares weight of each vector element in temporary accumulator (in tempacum.dat) with the threshold and, if it finds term (s) that meet the threshold, it causes the generation of the *TermFound* event by the NITrack adapter to the engine. This, in turn, causes a call to the *CheckInUM* sensor of the NITrack adapter. This sensor call causes the NITrack adapter to start its *checkInUM()* method that checks whether the term is already available in the user model. After checking, it responds "found" or "not found". If the term is not found in the user model, the rule that causes a call to *ImplementDecision* effector that suggests the term to the user is fired. The effector is also provided by the NITracker through its *ImplementDecision()* method.

In cases when the user accepts a term suggested by the New Interest Tracking Agent, the NITrack adapter adds that term to the user model and triggers the *UMUpdated* event. This event fires the *InitiateComposer* rule that causes the engine to send an effector call to the Compose adapter.

5.3.4. The Compose Adapter

The Compose Adapter composes an e-mail based on terms that the user approved as relevant to his information interest. Currently this facility is included just for demonstration purpose and does not do much enhancement on the terms except adding some common headings, etc. This component is designed as a separate adapter so as to allow future enhancements to the terms that might be needed by the server.

5.3.5. The User Interface

A simple interface that can be used to initiate the agent and browse agent messages (displays) is included. This component is intended to be enhanced to an interface similar to any text retrieval system especially when the Browse and Feedback agent is added.

5.4. The Test

A number of tests were made to test how well can the New Interest Tracking Agent trace the new interest of a user. Specifically the following aspects were considered:

- monitoring of new user documents
- the performance of stop word removal and stemming algorithms
- the performance of the suggested weighting method in identifying terms representing user's interest
- comparison of the performances of various thresholds

5.4.1. The Test Case

For the test, 12 ILRI scientists (out of about 20) who are also active users of ILRIAlerts were selected from the existing user profile. Then the plans for projects these scientists are working on were selected from ILRI's 1997 Project Work Plans (ILRI 1997). Table 5.4 shows the number of project plans selected for each user.

Table 5.4. : Number of project documents used for experiment

Users (profile ID.)	No. of documents
021-T	4
087-T	4
226-T	4
007-T	4
012-T	3
104-T	3
005-T	3
010-T	3
022-T	2
009-T	2
152-T	2
159-T	2
Total	36

Each project plan contains the program of which the project is a part, the project title, scientists and other staff involved, background and justification of the project, objectives, work plan for 1997, expected outputs (publications and others), a table showing the project's logical framework with headings like narrative summary, measurable indicators, means of verification, important assumptions, etc.

The 36 documents had a total of 35, 779 terms and 5002 distinct terms in 34 documents.

5.4.2. Monitoring of New User Documents

The New Interest tracking Agent has been tested for capabilities of tracing new documents stored in any directory as long as the full path of the directory or directories are indicated to it. Different directories at different hierarchies were indicated to the agent by storing it in the User model. Likewise, varying time periods at which these directories have to be checked for new files were specified in the user model. Then files from the test cases were saved in the directories at random intervals. The agent has managed to recognize all the new files and scan their contents. The above tests have indicated that the agent can be

developed as memory resident program that autonomously monitor specified directories without the user having to initiate it.

5.4.3. Performance of Stop Word Removal and Stemming Algorithms

While the agent is processing a document, intermediary results like term list after stop word removal and list of stemmed terms were kept for all of the test cases. The finite state machine perfectly removed terms similar to those found in the stop word list. Frakes (1992) implementation of the Porter suffix-stripping algorithm also reduced most word appropriately while there were also term which were overly stemmed to the extent that they lost their meanings. Some of these terms like *feed, breed, seed, crossbred*, etc. which were important to the documents studied were stemmed to *fe, bre, se, crossbr*, etc. In order to remove the effect of these terms on other factors considered in this experiment, they were first modified to regain their proper meaning before passed over to the weighting component. On average the stemming algorithm has properly stemmed about 93% of the terms.

5.4.3. Agent Performance

Basically the agent's performance depends up on its term weighting schemes and the threshold it employs. Tests were done to see how the methods for these two aspects suggested in the model perform. To this end, with the support of a livestock research associate and an indexer (both from ILRI), groups of single and hyphenated terms that approximately represent each of the 12 users were identified based on contents of the project plans.

The training document sets for each user were presented to the agent one after another so as to generate their cumulated vectors (term-weight pairs stored in the temporary accumulator). Then the standard mean weight of each vector as well as the estimated standard deviations (i.e., 1s.d., 2s.d., and 3s.d.) were calculated so as to determine thresholds based on the approach discussed in the previous chapter (section 4.4.2.1).

$$W_{ik} = \frac{f_{ik}}{\sum_{k=1}^{NoOfTerms} f_{ik}} * df_k$$

In order to obtain the threshold with better performance, the mean (which is 0) and the standard deviation values for each user were supplied to the agent as thresholds to apply on the test documents and the resulting terms were compared with the list of terms previously identified as descriptors of user interests. Table 5.5 shows the accuracy of these threshold values (accuracy measures as the proportion of relevant terms generated by agent out of the total number of terms generated). The obtained data are plotted in figure 5.2 so as to see the accuracy performance of the four threshold values.

Table 5.5 : Accuracy rates of four thresholds

		mean				s.d. 1				s.d. 2				s.d. 3			
Users (p. ID)	No. of terms	terms produced	% (terms in docs.)	Rel. terms	accuracy	terms produced	% (terms in docs.)	Rel. terms	accuracy	terms produced	% (terms in docs.)	Rel. terms	accuracy	terms produced	% (terms in docs.)	Rel. terms	accuracy
005-T	405	72	17.8%	20	27.8%	26	6.4%	10	38.5%	14	3.5%	8	57.1%	9	2.2%	6	66.7%
007-T	726	64	8.8%	22	34.4%	48	6.6%	16	33.3%	32	4.4%	12	37.5%	18	2.5%	6	33.3%
021-T	777	169	21.8%	41	24.3%	69	10.9%	30	43.5%	25	4.0%	13	52.0%	15	2.4%	8	53.3%
012-T	617	99	16.0%	29	29.3%	58	9.4%	24	41.4%	24	3.9%	11	45.8%	15	2.4%	10	66.7%
022-T	293	200	68.3%	18	9.0%	19	6.5%	8	42.1%	5	1.7%	3	60.0%	4	1.4%	3	75.0%
010-T	506	151	29.8%	20	13.2%	40	7.9%	15	37.5%	16	3.2%	6	37.5%	11	2.2%	6	54.5%
104-T	574	180	31.4%	45	25.0%	58	10.1%	25	43.1%	28	4.9%	18	64.3%	12	2.1%	10	83.3%
087-T	602	57	9.5%	15	26.3%	35	5.8%	11	31.4%	17	2.8%	7	41.2%	9	1.5%	5	55.6%
009-T	309	120	38.8%	23	19.2%	19	6.1%	12	63.2%	12	3.9%	9	75.0%	8	2.6%	8	100.0%
226-T	666	136	20.4%	34	25.0%	53	8.0%	25	47.2%	23	3.5%	13	56.5%	13	2.0%	8	61.5%
152-T	262	166	63.4%	15	9.0%	26	9.9%	10	38.5%	8	3.1%	5	62.5%	3	1.1%	3	100.0%
159-T	278	73	26.3%	10	13.7%	15	5.4%	9	60.0%	9	3.2%	5	55.6%	8	2.9%	6	75.0%

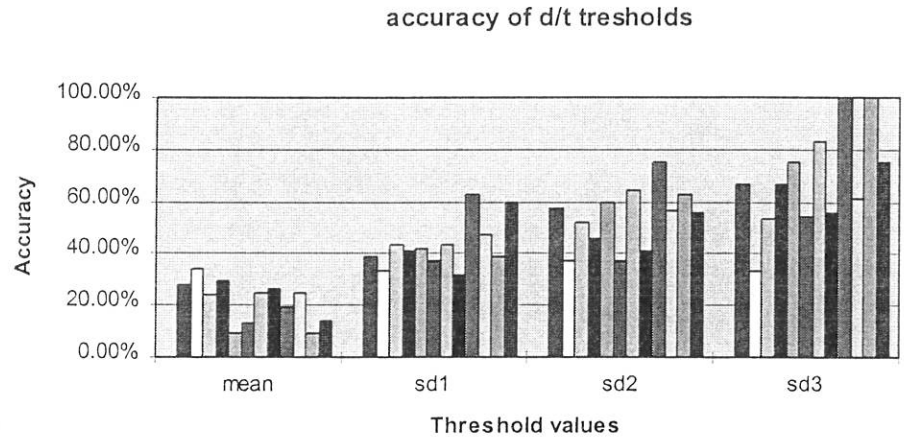


Figure 5.2. The performance of four threshold values

The above figure shows a general tendency of increasing accuracy as the threshold goes further away from the mean. However, as figure 5.3 shows, the number of terms relevant to the users interest kept on decreasing as we set the threshold further from the mean. This is shown more clearly in figure 5.4 which plots the error rate of each threshold as measured by the ratio of irrelevant terms produced.

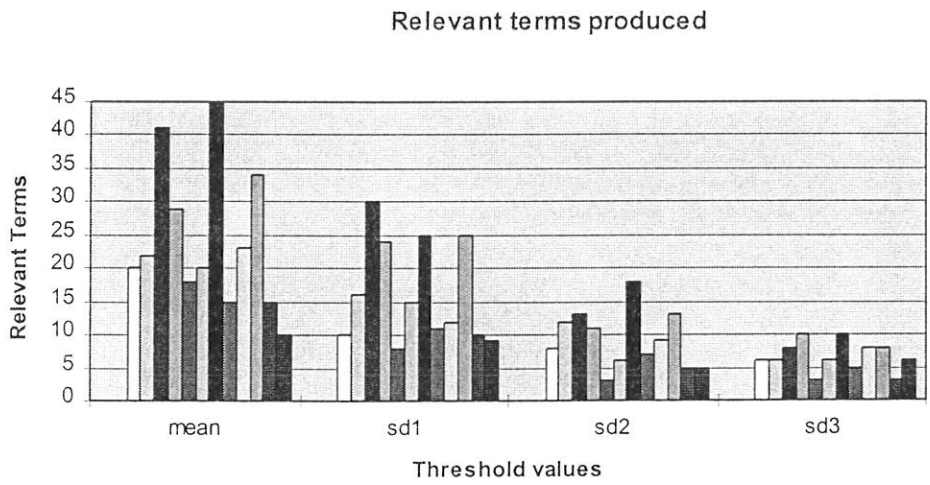


Figure 5.3: No. of terms generated by four threshold values

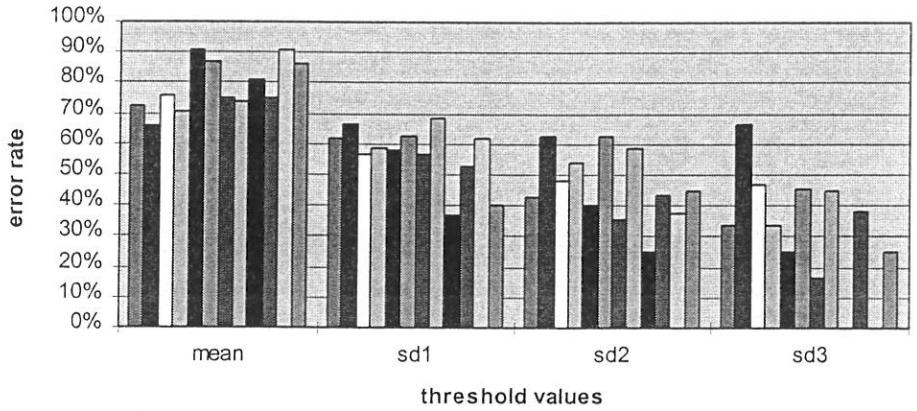


Figure 5.4.: Error rate of the four thresholds

5.5. Discussion

The above results, although they are based on a small number of test cases which can hardly be considered as representative of any sort, have provided some indications on the factors affecting the performance of the New Interest Tracking Agent. The functional components of the prototype agent, namely the user document file tracking, e-mail composing and delivery, and procedures like stop-list removal and stemming algorithms have shown good performance. However, it has been noted that some important additions and modifications are in order as regards the stop word lists and the stemming rules. Not making such changes are also observed to have critical impact on agent's performance.

The experiment made to see the performance of varying thresholds based on the basic hypothesis of using the mean as a staring point has indicated some general patterns although no clear cut information were found which, again, may be due to the small sample size considered. The first observation made is that the number and size of the documents appear to have an impact on the results obtained. It has been observed that terms related to user's interest continue to stand out in cases where larger number of documents were considered compared to cases where fewer documents were considered. This is probably

due to the fact that the weights assigned are directly proportional to the occurrence of terms across documents. This, in turn, has been interpreted as indicative of continuous improvement of the performance of the threshold determination process as the number and size of the documents increases.

Another factor that is thought to have influence on the test results is the fact that only one type of documents, namely project plans, are considered. These documents have more or less similar structure and length. Although this has been advantageous for controlling the impact of the threshold variations in this experiment, it is also highly likely that it resulted in biased results. For example, as the documents were describing the basic ideas of each project in brief, they might have resulted in large number of keywords which made smaller thresholds have good performance.

The other dimension from which the results obtained were considered is whether or not the weighting scheme and threshold determination methods perform in good harmony. While the weighting scheme values a term based on its frequency of occurrence in a document and its occurrence distribution across documents, the threshold determination method was trying to interpret the weights based on the theory of centroid. As the weight is directly proportional to frequency of occurrence, the threshold determination that considers the weight distribution as frequency distributions is observed to be appropriate. One demonstration of this is that the mean weight starting from which the thresholds are determined always moves with the concentration of larger weights.

One observation which stood out clearly is the impacts of varying thresholds which in turn showed the accuracy/coverage contrasts that these variations bring about. It has been noted

that as the threshold increases (goes far away from the mean), the number of terms suggested by the agent (that is coverage) decreases while the accuracy improves. Conversely, the number of terms suggested increases as the threshold decreases and, concomitantly, the accuracy goes down. This phenomena is illustrated in figures 5.2 and 5.3. Furthermore, it has also been noted that higher thresholds continue to drop relevant terms as well as irrelevant terms. Justifying the assumption that more important terms are likely to fall around the middle, the number and importance of terms that are dropped by higher thresholds has been found to be disastrous.

On the other hand, the number of terms that are generated by lower thresholds are found to be very large making it not reasonable to use them in an agent that is meant to assist an end user. As also noted by Edwards, *et al.* (1996) and Armstrong, *et al.* (1995) an agent system is more beneficial to a user if its assistance is infrequent but correct, rather than frequent but inaccurate. Using lower thresholds has been observed to result in agent suggestions that contradict with this belief. Also, commonly used irrelevant terms like *increase, impact, project, identify, research*, etc. have been found to result even when higher thresholds are used.

Viewed in light of these results, the simple weighting and threshold determination methods alone do not appear to effectively handle the accuracy/coverage tradeoff. Based on this observation, the need for considering other methods that deal with this weakness has been felt. One obvious approach that could alleviate this situation is strengthening the stop word list while the other one is filtering likely relevant terms using a thesaurus before weighting of terms is done. The latter approach is expected to result in better result as the language of a user's domain of interest is far smaller than the whole language which, in turn, means it

is more effective to extract potentially relevant terms than removing non-relevant terms. Hence, by adding a thesaurus, which is another knowledge base for the agent, we can specialize the agent to a certain domain of interest thereby improving its performance.

CHAPTER 6

CONCLUSION AND RECOMMENDATIONS

6.1. Conclusion

The work described in this thesis has tried to investigate into the problems pertaining to SDI user profile management and has attempted to demonstrate how agent technology can be used to address some of the problems. So as to have the required insight into the SDI user profile management processes and problems, a case SDI system, ILRIAlerts (maintained by ILRI), has been studied.

From the literature, it has been learnt that agent technology is being applied to a range of problems, from which most have fundamental commonality with problems of SDI user profile management in particular and SDI system in general. Hence, an attempt was made to see in further detail how this technology can be applied to user profile management. This has motivated the development of a generalized agent-oriented model that is based on the established client/server architecture. This was followed by an attempt to come up with a more detailed specification of the client component with particular emphasis to the two agents suggested in the generalized model. These two agents that are named as New Interest Tracking Agent and Browse and Feedback Agent were identified based on a careful analyses of the SDI user profile management domain. The detailed specification of these agents have tried to identify the appropriate development options and algorithms mainly based on established methods in agent technology and information retrieval.

However, for the New Interest Tracking Agent, an original method of predicting user's likely subject interest based on his/her job-related documents has been suggested in this study for lack of previous work in this area. This method was based on the term frequency weighting and document collection considerations in the vector space model. The weighting method that is used to produce a vector that distinguishes a document from others in a collection has been reversed and used to produce a vector that describes a collection of job-related documents of a user. This done based on the assumption that the vector representing the collection also represent the user's interest. For the implementation of this method, procedures of term frequency accumulation and vector merging methods have also been suggested. With the aim of experiencing as well as demonstrating how the suggested model can be implemented, a prototype agent has been developed using IBM's Agent Building Environment Developers Toolkit and Microsoft visual C++. Using this prototype, testing of the performance of the adopted and suggested agent methods and procedures have been tested. Although the test results need to be confirmed by more complete experiments, results obtained in the current small-scale tests have shown that the general approach is promising.

The tests were done on 36 project plan documents in which twelve ILRI Scientists who are also users of ILRIAlerts are working on. The agent has managed to autonomously trace job related documents and automatically start scanning their contents. The Porter stemming algorithm which has been used to reduce terms to their roots prior to the term weighting process has properly stemmed 93% of the terms. The experiment has also tried to see the effectiveness of a term weighting and threshold determination schemes that were suggested in the model by comparing four threshold values. This experiment has uncovered, *inter*

alia, the problem of accuracy/coverage tradeoff that needs to be dealt with if the weighting and threshold determination schemes should perform well. One option to this end, that is the use of a thesaurus, has been tested and general improvements on the previous results obtained.

Overall, this study has found that agent technology can be applied to alleviate some of the basic problems with SDI user profile management. This technology has been found to be particularly promising in developing client side applications that assist in functions like interest change tracking and feedback processing. In particular, using an appropriate term weighting and threshold determination schemes in combination with a relevant and complete thesaurus has been found to be promising in implementing a client side agent that autonomously traces user's new interests and carries out the routine processes of informing the SDI server of these interests.

Besides, this study has resulted in a good understanding of the problem area and has helped to conceive the research to take into account practical problems in addition to carrying out the research for pure academic purposes.

6.2 Recommendations

This research has uncovered that agent technology holds a great potential in improving the SDI user profile management process. It was also learnt that a lot of more research and developmental efforts need to be conducted to enable the full exploitation of this technology and the realization of systems employing the agent-oriented model suggested

in this thesis or a modification of it. In particular, the following areas were identified as deserving further research work:

1. A more complete testing of user interest tracking and threshold determination methods suggested in this research using a comprehensive test collection composed of various kinds of user documents.

2. Empirically testing the Rocchio algorithm suggested for agent learning from feedback and comparing its performance in the SDI domain with other machine learning algorithms. Further research can also consider including additional learning sources for the agent like networks (for example, employee databases, etc.) and observation of user computer actions.

3. This study has precluded the server side agents completely. But the proper functionality of the whole agent-based system requires efficient server side agents . Further work in this component can build on works in Intelligent knowledge-based information retrieval, automatic search formulation and reformulation heuristics, filtering techniques and the like.

4. With particular reference to Information Systems and Services in developing countries, the cost-effectiveness and possible advantages of agent-based SDI systems need to be considered in further detail. In particular, coupled with Internet, the potential such systems hold in enhancing the accessibility of the large amount of indigenous development information in the databases of African information systems need to be given enough consideration by future works.

5. Determining domain knowledge that could enhance the reasoning capability of the agent is also very important in improving the performance of client-side agents. In particular, the adoption and integration of a thesaurus that has been tried in this study at a small-scale level needs further study. In particular, system performance issues that result from the size of thesauri and mechanisms of keeping them up-to-date needs particular emphasis. In this connection, another area to be considered is using a machine learning methods in customizing standard profiles (i.e. profiles provided by database producers to represent interests on a given subject) which in turn allows the agent to acquire an initial user model which it can improve upon in time.

6. Adding mobility to the client agents is also recommended to enable the agent learn from other agents of users with similar interests. The SDI server is envisaged to serve as a good meeting place for such agents to learn from one another. The mobility feature also enables accessing additional learning sources from networks like organizational LANs and Intranets.

REFERENCES

Allan, James.1995 "Relevance Feedback with Too Much Data." Proccedings of SIGIR' 95, 337-347.

Akoulchina, Irina and Ganascia, Hean-Gabriel. 1997. "SATELIT-Agent:An adaptive interface based on learning interface agent technology". in UM97: The Sixth International Conference on User Moodelling China Laguana, Sardinia, Italy, June 2-5 1997.

Amy, Doan. 1997. "Pushing Back". InfoWorld. 19(31): 1-2

Armstrong, et al. 1995. "Webwathcher: a learning apprentice for the World Wide Web". In Proceedings of the 1995 AAAI Spring symposium on Information Gathering from heterogeneous, Distributed environments, Stanford, CA, 1995: AAAI Press.

Balabonovic, Marko and Shoham, Yoav. 1995. Learning information retrieval Agents: Experiments with Automated web-browsing. In Proceedings of the 1995 AAAI Spring symposium on Information Gathering from heterogeneous, Distributed environments, Stanford, CA, 1995: AAAI Press.

Barker, F. H. *et al.* 1972. "Towards Automatic Profile Construction". Journal of Documentation. 28: 44-55.

Bentley, Mark. 1995. "Intelligent Agents." in 1995 Student Reports (at http://ksi. CpSc. Ucalgary.ca/courses/547-95/bentley/ 547 talk.html

Berry, M.W. Dumais, S.T, Obrein, G.W. 1995 "Using Linear Algebra for Intelligent Information Retrieval". SIAM Review, 37(4): 573 - 595.

Bigus, Joseph p. 1996. Data Mining With Neural Networks: Solving Business Problems from Application Development to Decision Support, New York Mc Graw-Hill.

Birungi, P. K. 1995. "Improved strategies for Employment and Human Resources Utilization in the Information and Documentation Sector". In Strategies for Human Resources Development for Information Management in Africa , Ed. Francis Inganji, 49-57. Addis Ababa: UNECA,PADIS.

Bischofberger, W. and Pomberger, G. 1992. Prototyping Oriented Software Development: Concepts and Tools. Berlin: Springer-Verlag

Bloedorn, Eric, *et al.* 1996. "Representational Issues in Machine Learning of User Profiles". In Proc. of AAAI Spring Symp. On Machine Learning in Information Access. Stanford, March 25-27.

Boar, Bernard H.1993. "Implementing Client/Server Computing: A Stategic Perspective". New York: McCraw Hill, Inc.

Bradshaw, Jeffrey M. 1997. An Introduction to software agents. In software agents ed J.M Bradshaw. Menlo park, Calif.: AAAI press:3-46

Buckley, Chris, Salton, Gerard & Allan, James. 1994. " The Effect of Adding Relevance Information in a Relevance Feedback Environment." 17th Int'l ACM/ISIGIR Conference on Research and Development in Information Retrieval, Dublin, Ireland, June, 1994: 292-298.

Butler, John T. 1993. A Current Awareness Services Using Microcomputer Databases and Electronic Mail. College & Research Libraries, 54(1), 115-123.

Chen, Hsinchun. 1995. "Machine Learning for Information retrieval: Neural Networks, Symbolic Learning, and Genetic Algorithms". JASIS. 46(3): 194-216.

Cohen, william W. 1996. Learning rules that Classify E-mail.

Cortouts, Jan and Philips, richard. 1996. The SDI service of VUBIS-Antwerp: User Manual. Belgium: university of antwerpen.

Daniels, P. J. 1986. "Cognitive Models in Information Retrieval - An Evaluative Review". Journal of Documentaiton. 42(4): 272-304.

D'Alossi, Daniela and Giannini, Vittorio. 1995. "The InfoAgent: An Interface for Supporting Users in Intelligent Retrieval", November 1995.

DNmmond, C, Ionesuc, D, and Holte, R. 1995. "A Learning Agent that Asssts the Browsing of Software Libraries, Technical Report TR-95-12". Computer Science Department, University of Tawa.

Edwards, Peter, et al. 1996. "Experience with Learning Agents which Manage Internet-Based Information".

Ellis, Davis. 1990. New Horizons in Information Retrieval. London. London:The Library Association Pub.

Erickson Thomas 1997. "Designing agents as if people Matter." In software agents ed J.M Bradshaw. Menlo Park, Calif.: AAAI press :79-96

Etizioni, O.Weld, D. 1994. "A Softbot-Based Interface to the Internet". Communications of theACM, 37(7): 72-79

Feibel, Werner. 1995. Novell's Compute Encyclopedia of Networking. Alameda: Novell Press.

Finin, Tim, Labrou, Yannis & Mayfield, James 1997. "KQML as an Agent Communicaiton Language". In software agents ed J.M Bradshaw. Menlo Park, calif. AAAI press : 291 - 316

Folk, Michael T. and Zoellick, Bill. 1992. File Structures. 2nd ed. Reading, Massachasetts: Addison-Wesley Pus. Co.

Foltz, Peter W.and Dumais, Susan T. 1992. "Personalized Information Delivery: An Analysis of Information Filtering Methods". Communications of the ACM. 35(12): 51-60

Frakes, W.B. 1992. "Stemming Algorithms". In Information Retrieval Data Structures and Algorithms, Frakes, W.B. and Bacza-Yates,R., eds.

Gauch, Susan. 1992. Intelligent Information Retrieval: An Introduction. JASIS. 43(2): 175-182.

Genesereth, Michael R, 1997 An agent based framework for Interpretability. In In Software Agents ed J.M Bradshaw. Menlo Park, Calif.: AAAI press : 317-345

Gibert, Don et al. 1995. IBM Intelligent Agent strategy, IBM Corporation.

Gilbert, Don. 1997. Intelligent Agents: The Right Information at the Right Time. IBM Corporation, Research Triangle Park: USA.

Guinchat, Claire and Menou, Michael. 1983. General Introduction to the techniques of Information and Documentation Work. Paris: UNESCO

Hamilton, Feona. 1990. Information publicity and Marketing of ideas for the information profession. England: Gower Pub.

Hamilton, Feona. 1995. Current Awareness, Current Techniques. Aldershot: Gower.

Haravu, L. J. 1995. General Purpose of SDI Program for use with Micr-CDS/ISIS Databases: NEWSDI.PCD Version 2.0. (A readme file attached with the NEWSDI program)

Haygarth-Jackson, A. R.1972. "Utilization of mechanized services". Information Scientist, vol. 6:132-138.

Hielmann, Kathryn, et al. 1995. Intelligent Agents: A Technology and Business Application Analysis. USA: Intelligentia, Inc.

Hiltz, S. and Turoff,M. July 1995. "Structuring Computer-Mediated Systems to Avoid Information Overload." Communications of ACM, 28(7), pp 680-689

Holte. R.C. and Drummand, C.1994. "A Learning Apprentice for Browsing". In AAAI Spring Symposium on Software Agents, 1995.

IBM. 1997. Agent Building Environment (ABE): Technical Overview. (at Intelligent Agents Home Page, www.networking.ibm.com/iag/iaghome.html.)

IBM. 1997. IBM Agent Building Environment Developer's Toolkit. IBM Corporation.

IDRC. 1985. MINISIS SDI: A Guide to the MINISIS SDI Facility. Canada: IDRC.

ILCA. 1991. Information Policy and Procedure Manual: First Draft. Addis Ababa: ILCA, July .

Ittner, David J., Lewis, David D and Ahn, David D. 1995. " Text Categorization of Low Quality Images". In Fourth Annual Symposium on Document Analysis and Information Retrieval, Las Vegas, NV, April 1995, ISRI; Univ of Nevada, Las Vegas : 301-315.

Jacques Delors Information centre. 1997. Information services at Jacques Delors Information centre (at: www.cijdelors.pt/inform/difusaoi,html)

Katz, William A. 1997. Introduction to Reference Work: Volume II' Reference Services and Reference Processes. 7th ed. New York: McGraw-Hill.

Keenan, P.M and Mountgomery, C. H. 1994. "An SDI program for Distributing Funding Opportunities using electronic mail". J. of the American Medical Informatics association. No. ss, p. 982.

Kemp, D.A. 1979. Current Awareness Services. London: Clive Bingley

Kendall, Elizabeth A. et al. 1995. "A Methodology for Developing Agents Based Systems". In Distributed Artificial Intelligence: Architecture and Modelling, First Australian Workshop on DAI, Canberra, ACT, Australia Nov. 1995 proceedings: 85-99

Knoblock, Craig, A. and Ambite, Jose' Luis. 1997. ... In software agents ed J.M Bradshaw. Menco para,calif AAAI press : 347-373

Knoblock, Craig A., Arens, Yigal and Hus, Chun-Nan. 1994. "Cooperating Agents for Information Retrieval". In Proceedings of the Second International Conference on Cooperative Information Systems, University of Toronto press, Toronto, Ontario, Canada

Krulwich, Bruce. 1995. "Learning user interests across heterogenous document databases." Proc. 1995 AAAI Symp. on information gathering from heterogenous, Distributed environments, Stanford, March 1995. AAAI Press

Lang, Ken. 1995. "NewsWeeder: learning to filter netnews. In Machine Learning: Proceedings of the 12th International Conference, Lake taho, California, 1995.

Lary, K. 1995. "News Weeder: Learning to filter Netnews". Proc. of the 12th International Conference on Machine Learning ICML 95-1995.

Lashkari, Yezdi, Metral, Max & Maes, Pattie. 1994. "Collaborative Interface Agents". In Proceedings of AAAI'94 Conference, Seattle, WA, July 31-Aug 4, 1994 : 1-6.

Laurel, Brenda 1997. "Interface Agents Metaphors with Character." In software agents ed J.M Bradshaw. Menlo Park, Calif. : AAAI press : 67-77

Lee, Dik L., Chuang, Huei and Seamons, Kent. ???? "Effectiveness of Document Ranking and Relevance Feedback Techniques"

Leggate, P. 1975. "Computer-based Current Awareness Service". Journal of Documentation. 31 (2):93-115.

Library Technology. News. April 1996, 1(2): p. 29.

Library Technology 1997, 2(1): P.4.

Library Technology 1996, 1(2): P.27

Gibson, Paul. 1997. "LineOne gets a face lift". Information World Review. October 1997, no. 129 p. 14

Luhn, H. P. 1958. "A Business Intelligence System". IBM Journal of Research and Development, October 1958:314-319.

Luhn, H. P. 1961. "Selective Dissemination of new scientific information with the aid of electronic processing equipment", American Documentation, April 1961: 131-138.

Maes, Pattie. 1994. Agents that Reduce Workload and Information Overload. Communications of the ACM , 37(7 July): 31-40.

Maes. Pattee and Kozierok, Robyn. 1993. "Learn Interface Agents". In Proceedings of AAAI 1993 conference washington, D.C. N/Y 1993: 459-465

Marion, William. 1994. Client/Server Strategies: Implementation in the IBM Environment. New York : Internex Publications & McGraw-Hill, Inc.

Mauerhoff, George R. 1974. "Selective Dissemination of Information". Advances in Libraries. 4: 25-62.

Mitchell, T.M. 1997. Machine Learning. The McGraw-Hill Companies, Inc.

Mondschein, Lawrence G. 1990. SDI Use and Productivity in the Corporate Research Environment. Special Libraries. 81(4):

Mountifield, H. M. 1995. "electronic current awareness service: a survival tool fore the information age". The electronic Library 13(4, August): 317-324.

Mugasha, James. 1995. The Role of Appropriate Labour Market Information to Manpower Planning. In Strategies for Human Resources Development for Information Management in Africa , Ed. Francis Inganji, 67-72. Addis Ababa: UNECA, PADIS.

Nageri, M. W. 1994. "The need for African Science and Technology Databases: Critical areas of Concern," in Workshop on databases: the needs and contributions of African researchers, Addis Ababa, Ethiopia, 10-12 October 1994. Addis Ababa: UNECA, PADIS.

Nardi, Bonnie A. and O'Day, Vicki. 1996. Intelligent Agents: What We Learned at the Libraries. Libri. 46: 59-88.

Nganga, James M. 1995. Marginalization of Personnel in Information and Documetation Sector in Africa. In Strategies for Human Resources Development for Information Management in Africa , Ed. Francis Inganji, 89-93. Addis Ababa: UNECA, PADIS.

Nwana , Hyacinth s. 1996. "Software agents: An overview" . The Knowledge engineering Review. 11(3) : 205-244

Nwana, H.S. and Wooldridge, M. 1996. "Software Agent Technologies" BT Technology Journal, 14 (4): 68-78

Oard, Dough. 1997. Information Filtering Defined. (at Information Filtering Resources, http://www.glue.umd.edu/~oard/)

Oard, D.W. 1997. "The state of the Art in Text Filtering." User Modeling and User-Adapted Interaction, 7(3) : 141 - 178.

Orfali, Robert, et al. 1994. Essential Client/Server survival Guide. USA: Van Nostrand Reinhold.

Pannu, Anandep S. and Sycara, Katia. 1996. Learning Text Filtering Preferences.

Papazoglou, Mike P., Laufman, Steven C. & Sellis, Timos, K. 1992. "An organizational Framework for Cooperating Intelligent Information Systems." International Journal of Intelligent and Cooperative Information Systems. 1(1): 169-202.

Pao, Miranda Lee. 1989. Concepts of Information Retrieval. Englewood, Colorado: Libraries Unlimited

Pazzani, M., Muramatru, J., Billsus, D., Sy Skill and Wesert 1996. "Identifying interesting Web Site". In Proceedings of the Thirteenth National Conference on Artificial Intellegence AAAI 96: 54-61.

Porter, M. 1980. "An algorithm for suffix stripping". Program 14(3): 130-137

Ravi, A. Sreenivasa and Indira, B. C. 1994. "An e-mail-based bibliographic information Server". Journal of Information Science, 20(4): 295-299.

REPIDISCA. 1997. What is REPIDISCA ? (at http://www.ids.ac.uk/eldis/data/d013/e01302.html)

Reva, Basch. 1997. "Special Delivery: Data Pushing services may change the Web". Computer Life. 4(4, April): 50-52

Rocchio, J. 1971. "Relevance Feedback in Information Retrieval". In the SMART Retrieval System: Experiments in Automatic Document Processing, Chapter 14. Prentice-Hall, Inc. 313-323.

Rogers, Jane. 1985. "Present-day Developments in Current Awareness services". In Practical Current Awareness Services from Libraries, ed. by Tom Whitehall. USA and England: Gower Publishing Co. :74-87

Rowley, Jennifer, 1993. Computers for Libraries. 3rd ed. London: Library Association Publishing

Ryan, B. 1991. The Data Swamp, Byte 16(5): 153-156

Salton, G. and McGill, M. 1983. Introduction to Modern Information Rettrieval. New York: McGraw- Hill.

Salton, G. 1989. Automatic Text Processing.

Saracevic, Tefko and Wood, Judith B. 1981. Consolidation of information: A handbook on Evaluation, Restructuring and Repackaging of Scientific and technical information (Pilot edition). PGI,UNESCO: Paris.

Saraceric, Tefko; Spink, Amanda and Wu, Mei-Mei. 1997, "Users and Intermediaries in Information Retrieval. What are they talking about ?" In user modeling: proceeding of the sixth Internation Conference, UM97, Anthony Jameson, Cecile Paris and Carlo tasso, eds. Vienna, New York: Springer Wein, New York.

Sheth, B. 1994. A learning Approach to Personalized Information Filtering, Master's Thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology.

Sheth, Beerud & Maes, Parrie. 1993. "Evolving Agents for personalized Information Filtering". In Proceedings of the Ninth IEEE Conference on Artificial Intelligence for Applications, 1993: 1-7

Shoham, Yoav. 1997. " An Overview of Agnet-Oriented Programming". In software agents ed J.M Bradshaw. Menlo Park, Calif.: AAAI press : 271-290

SilverPlatter Infomation Ltd. 1997. ERL Technology: "The choice of Accesing Infomation". SilverPlatter News Africa. p. 7.

Sommaruga, Lorenzo and Catenazzi, Nadia. 1995. "From Practice to Theory in Desining Autonomous Agents". In Distributed Artificial Intelligence: Architecture and Modelling, First Australian Workshop on DAI, Canberra, ACT, Australia Nov. 1995 proceedings: : 130-143

Sprague, Robert J. and Freudenreich, L.Ben. 1978. "Building Better SDI profiles for users of large, multidisciplinary Databases". JASIS 29(1): 278-282

Stefanuk, Vadim L. 1996. "Embedding user Models in Intelligent Interfaces. In Fifth International Conference on user modelling (UM96). Kailua-Kona, HI, january 2-5, 1996.

Sycara Katia etal. 1996 "Distributed Intelligent agents IEE Expert December "1996; 36-45. Maea, patile 1994 Agent That Reduce Worth and Information Overload, *Communications of the ACM.* 7:31-409

Thomson, Angela. "Building Agents". in 1995 Student Reports (at http://ksi. CpSc. Ucalgary.ca/courses/547-95/bentley/ 547 talk.html

Torkelson, N.R. 1973. SDI as a control of information overload. Proceedings of the American Society for Information Science, 10, 231-2

UNECA. 1994. Techniques of Marketing Information/Information services and Products. in Training Workshop for University Lecturers Teaching Library and Information Science in African Universities and Institutions of Higher Learning held in Gaborone, Botswans 5-16 December 1994, Addis Ababa:UNECA, PADIS.

UNESCO. 1980. Guide for the establishment and evaluation of services for Selective Dissemination of information. Paris:UNESCO.

Weaving, Edmond. 1991. Current Awareness Services and the Information Center. Aslib Proceedings . 43 (10, October): 299-303

Whitehall, Tom (ed). 1986. "Alternatives for Current Awareness Service (CAS)". In Practical Current Awareness Services from Libraries, ed. by Tom Whitehall. USA and England: Gower Publishing Co. :1-14

Whitehall, Tom. 1997. "Dissemination of Information". In International Encyclopedia of Information and Library Science. ed by John Feather and Paul Sturges. London & New York. Rastledge.

Wooldridge , M.J, and Jennirgs N.R 1995 "Agent Theories, Architectures and Languages" *tutorial.* first International conference on multi-agent systems.

Wyle, Michell F. 1996. Effective dissemination of WAN information. Ph.D. disseration, Lasalle University.

Yan, Tak W. and Gascia-Molina. Index Structures for Information Filtering Under the Vector Space Model.

Yan, Tak W. Garcial-Molina, Hecotr. 1993. Index Structure for Selective Disseminaton of Information Under the Boolean Model. Technical Report, Stanford Computer Science Stan-CS-92-1454, 1-33.

Yan, Tak W. Garcial-Molina, Hecotr. 1994. "Distributed Selective Dissemination of Information. In Proceedings of the Third International Conference on Parallel and Distributed Information Systems," Sep. 28-30, 1994, Austin Texas. IEEE Computing Society Press, Los Alamitos, LA, 89-98.

Yochum, Julian A. 1996. "Research in Automatic Profile Generation and Passage-Level Routing with LMDS". In the 4th Text Retrieval Conference (TREC-4), NIST SP:

Yochum, Julian A. Research in Automatic Profile Generation and Passage-Level Routing with LMDS..... Fourth Text Retrival Conference (TREC-4) National Institute of standards and Technology.

Annex 1

```
/*************************************************************************
**    File:       nita.cpp
**    Description:  Implementation of the prototype NITA (New Interest
**    Tracking Agent) control.
**    Main agent program to read configuration file,
**    connect engines and adapters, and start the agent running.
**
*************************************************************************/

#include <iostream.h>
#include <fstream.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <signal.h>
#include <ctype.h>
#include <locale.h>
#include <time.h>
#include <errno.h>

#include <iatk/agent.h>
#include <iatk/iaerrors.h>
#include <iatk/handles.h>
#include <iatk/atom.h>
#include <iatk/library.h>
#include <iatk/collect.h>

char *cfgFileName;  // to hold cfg file name for all functions
istream * cfgFile;
char *linebuf;

class NITAgent : public IAAgent {
 public:
  NITAgent(const char *fn= "pocuagnt.cfg");
  ~NITAgent(){}
        bool addEngine(IAEngineHandle );
   bool addLibrary(IALibraryHandle );
   bool addAdapters();
        bool loadConductSet();

 private:
        bool ConfigureAgent();
};

static NITAgent ProtoTypeAgent; //IAAgent implementation;
IAEngineHandle engineHandle; // Handle for inference Engine
IALibraryHandle libHandle;  // Handle for library


NITAgent::NITAgent(const char *fname): // throw Exception
        IAAgent(fname)
{
   cfgFileName = new char [strlen(fname) + 1];
        if (!ConfigureAgent())
                throw ("Agent configuration failed"); // Setup agent

}
```

```
void main(int argc, char **argv)
{
        try {
                        try
                        {    // start the agent
                                ProtoTypeAgent.start();
                        }
                        catch (IAError &e)
                        {
                                cerr << e.errormsg() << endl;
                                cerr << "Error starting agent." << endl;
                        }
        } catch (...) {
                exit(1);
        }
}

bool NITAgent::ConfigureAgent()
{
    char * keyword;
     int adaptercount = 0;
     int i;

     try {
            ifstream infile;
            infile.open(cfgFileName);
            if (!infile) {
                     cerr << "Unable to open configuration file" << endl;
                     exit (3);
            }
            cfgFile = &infile;
            if (!addEngine(engineHandle)) return false;
            else cout << "Engine Loaded ... " << endl;
            do {
                     cfgFile->getline(linebuf, sizeof(linebuf));
                     if (!strlen(linebuf) || linebuf[0] == '#' || linebuf[0] == '*')     continue;
                     if (isspace(linebuf[0]))
                         cerr << "Keyword must start in column 1" << endl;
                     for (i = 0; isspace(linebuf[i]) != 0; i++) {
                              keyword[i] = linebuf[i];
                     }
                     if (keyword == "library") {
                              if (addLibrary(libHandle))
                                       cout << "Library Loaded ..." << endl;
                     }
                     if (keyword == "adapter") {
                              ++adaptercount;
                              if (addAdapters())
                                       cout << adaptercount  << "adapter loaded ..." << endl;
                     }
                     if (keyword == "conductset")
                              loadConductSet(); // Load conduct sets to engine
                     else cerr << "Improper specification in" << infile << endl;
            } while (!infile.eof());
    } catch (...) {
        cerr << "Setting up agent exception ";
        return false;
    }
    return true;
}
```

```
bool NITAgent::addEngine(IAEngineHandle engineHandle)  // ADDING THE ENGINE
{
    cout << "Adding engine RAISE"<< endl;
    try {
          IAAgent::addEngine("RAISE","iagerais","",engineHandle);
    } catch (IAError &e) {
      cerr << e.errormsg() << endl;
      cerr << "Engine setup error." << endl;
      return false;
    }
    return true;
}

bool NITAgent::addLibrary(IALibraryHandle libHandle)  // Declare library parms
{
    char *cfgword[6];
    char *word;
    int i, j, w = 0;
    char *map; // Map to library within file system
    char *topCollector; // Name of top collector in library (top directory)
    char *libType; // Library implementation is a file system
    char *library_implementation_dll; // Dll containing implementation of library
    int NoOfLevels; // Number of Collectors levels in the Library

    try {
          // parse linebuf
          for (i = 0; linebuf[i] != '\n' || linebuf[i] != '\r'; i++) {
                for (j = 0; isspace(linebuf[i]) != 0; j++) {
                      word[j] = linebuf[i];
                }
                cfgword[w] = word;
                w++;
          }
          if (cfgword[0] == "library") {
                 // get library parms from cfg file.
                 libType = cfgword[1]; // library type
                 library_implementation_dll = cfgword[2]; // library module name
                  map = cfgword[3];  // library map
                 topCollector = cfgword[4]; // library top collector
                 if (cfgword[5])
                        NoOfLevels = atoi(cfgword[5]); // no. of Collector levels
                 else NoOfLevels = 2;
                 cout << "Adding library";
                 IAAgent::addLibrary(map, topCollector, library_implementation_dll,
                        libType, NoOfLevels, libHandle);//Add library to the engine
                 IALibInferenceCollector *appTopCollector;
                 IALibrary appLibrary; // Create library object for adapter
                 appLibrary.attachImplementation(map, libType,  library_implementation_dll,
                      topCollector, NoOfLevels);

                // Create and access the top collector object
                appTopCollector = appLibrary.getTop();  // returns pointer to

                 //IALibInferenceCollector
                appTopCollector->get();
          } else {
              cout << endl <<"Improper specification of: " << cfgFileName;
              return false;
          }
    }
```

```
    catch (IAError &e) {
      cerr << e.errormsg();
                    cerr << "Library setup error.";
      return false;
    }
    // Catch fileNotFoundException, NumberFormatException, and IOException
    catch (...) {
      cerr << "Library setup exception. ";
                    return false;
    }
    return true;
}

bool NITAgent::addAdapters()  // ADDING THE ADAPTERS.
{
    IAAdapterHandle aHandle;
    char *domain;
    char *moduleName;
    char *parms;

    char *cfgword[3];
    char *word;
    int i,j, w = 0;

    try {
          // parse linebuf
          for (i = 0; linebuf[i] != '\n' || linebuf[i] != '\r'; i++) {
                for (j = 0; isspace(linebuf[i]) != 0; j++) {
                      word[j] = linebuf[i];
                }
                cfgword[w] = word;
                w++;
          }
          if (cfgword[0] == "adapter") {
                // get adapter parms from cfg file.
                domain = cfgword[1];  // adapter domain
                moduleName = cfgword[2]; // adapter module name
            if (cfgword[3])
                    parms = cfgword[3]; // parms for the adapter (if they exist)
                                                        // only one parameter is assumed
            else parms = "";
        // add the adapter and connect it to the engine
        IAAdapterHandle aHandle;
        IAAgent::addAdapter(domain,moduleName,parms,aHandle);
        IAAgent::connect(aHandle,engineHandle);
    }
  }
  catch (...) {
    cerr << "Adapters setup exception ";
    return false;
  }
  return true;
}
```

```
bool NITAgent::loadConductSet() // LOAD CS TO ENGINE
{
    char *CSname;
    char *selector;

    char *cfgword[3];
    char *word;
    int i,j, w = 0;

    // parse linebuf
    for (i = 0; linebuf[i] != '\n' || linebuf[i] != '\r'; i++) {
        for (j = 0; isspace(linebuf[i]) != 0; j++) {
            word[j] = linebuf[i];
        }
        word[j] = '\0'; // to null terminate word;
        cfgword[w] = word;
        w++;
    }
    if (cfgword[0] == "conductset") {
        // get CS parms from cfg file.
        if (cfgword[1] == "RAISE") {
            CSname = cfgword[2];
            if (cfgword[3])
                selector = cfgword[3];
            else selector = "";
        }
        else {
            cout << "Not conduct set statement." << endl;
            return false;
        }
    }
    try {
        IAAgent::loadConductSet(libHandle, engineHandle, CSname, selector);
        return true;
    }
    catch (IAError &e) {
        cerr << e.errormsg() << endl;
        cerr<< "Error loading conduct set with name = " << CSname <<endl;
        exit(5);
    }
}
```

Annex 3

```
/*************************************************************************
** File:        NTracker.cpp
** Description:  Implementation of the New Interest Tracking adapter
**
*************************************************************************/

#include <stdlib.h>
#include <iostream.h>
#include <fstream.h>
#include <ctype.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <math.h>

#include "NTracker.h"
#include "strlist.h"
#include "stop.h"
#include "stem.h"


typedef struct _StrListStruct {
        short size;     // current length of the list
        short max_size; // room for this many strings
        char **string;  // the string array
        } StrListStruct;


typedef struct _StrListStruct * StrList; // the base string list type

// functions for common string list operations written in ANSI C and
// includded in the project
extern "C" StrListAppendFile( StrList list, char *filename );
extern "C" StrList StrListCreate( void );
extern "C" void QSort( char **string, int lb, int ub );
extern "C" DFA   BuildDFA(StrList words );
extern "C" char *GetTerm( FILE *stream, DFA machine, int size, char *output );

// Define tokens for any sensors and effectors
const int NewInterestAdapter::AgentDecision_Token     = 1;
const int NewInterestAdapter::FindTermOfInterest_Token = 2;
const int NewInterestAdapter::WeightTerms_Token       = 3;
const int NewInterestAdapter::CheckInUM_Token         = 4;

bool resp;

NewInterestAdapter::NewInterestAdapter():IAAdapter() {}

//^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
// identify - register all effectors and sensors with agent
//^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

IABool NewInterestAdapter::identify() {

  // Register Effectors

   registerProcedure("WeightTerms", IAEffectorProc,
```

```
        "String", WeightTerms_Token);
    registerProcedure("FindTermOfInterest", IAEffectorProc,
        "String", FindTermOfInterest_Token);
    registerProcedure("AgentDecision", IAEffectorProc,
        "String", AgentDecision_Token);

    // Register Boolean Sensors

    registerProcedure("CheckInUM", IABooleanSensorProc,
        "String", CheckInUM_Token);

  return true;  // Success
}


//^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
// performAction - Execute Effector procedure
//^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
void NewInterestAdapter::performAction(IAProcToken token,
  const IAEventHeader &hdr, const char *argument)
{
  // Build the atom from the argument string (a KIF string). argument is
  // what the engine passes to NewInterestAdapter (as per the rule).
  int count;
  VectorAtom **foundSet;
  IAAtom atom;
  atom.setKIF(argument);

  switch(token) {
     case WeightTerms_Token: {
          weightTerms(atom);
          break;
     }
     case FindTermOfInterest_Token: {
          foundSet = findTermsOfInterest(atom, count);
          if (foundSet) {        // send event to engine
               IAAtom foundAtom; // create an atom
               foundAtom.setPredicate("TermName");
               IAFactSet foundFactSet; // create a fact set
               foundAtom.setTerms(""); // clear any current atom terms
                                       // for a single term
               foundAtom.addTerm(foundSet[i]->term);
               foundAtom.addTerm(foundSet[i]->weight);
               foundFactSet.add(foundAtom); // add atom the current fact set
               IATriggerEvent termFoundEvent;
               termFoundEvent.setType("TermFoundEvent");
               termFoundEvent.setFactSet(foundFactSet);
               notify(termFoundEvent);
          }
          else
               cerr << "No term found.." << endl;
               break;
     }
     case AgentDecision_Token : {
               agentDecision();
               break;
     }
  }
}
```

```
//^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
// testCondition - Invoke a BOOLEAN sensor procedure.
//^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

IABool NewInterestAdapter::testCondition(IAProcToken token, const IAEventHeader &hdr,
                                         const char *argument)
{

  // Build the atom from the argument string
  IAAtom atom;
  atom.setKIF(argument);

  switch(token) {
          case CheckInUM_Token: {
      CheckInUM(atom);
      break;
    }
  }
}


void NewInterestAdapter::weightTerms(IAAtom iatom)
{
          short TotalWords, VecSize;
          unsigned int totDocTerms;
          resp = false;
          char * filename = iatom.getStringTerm(1);

          /* open/create files. To assist the testing operation all
            intermediate results are stored in files */

          fstream tempFile("templist.dat", ios::in | ios::out ); // vector file
          if (!tempFile) {
                    cout << "templist.dat can not be opened. " << endl;
                    exit(1);
          }
          fstream listFile("termlist.dat", ios::in | ios::out);
          if (!listFile) {
                    cout << "termlist.dat can not be opened. " << endl;
                    exit(1);
          }

          fstream docVecFile("docvect.dat", ios::out | ios::in);
          if (!docVecFile) {
                    cout << "docvect.dat can not be opened. " << endl;
                    exit(1);
          }

          fstream tempaccumFile("tempacum.dat", ios::out | ios::app | ios::in);
          if (!tempaccumFile) {
                    cout << "tempacum.dat can not be opened. " << endl;
                    exit(1);
          }

          // process the new document
          RemoveStop(filename, tempFile);
          TotalWords = StemTerms(tempFile, listFile);
          VecSize = CalculateDocWeights(listFile, docVecFile, TotalWords, totDocTerms);
          resp = UpdateTempAccum(docVecFile, tempaccumFile, VecSize, totDocTerms, wFile);
          if (resp) {
```

```
                // send event to engine
                IAAtom respAtom; // create an atom
                respAtom.setPredicate("EventName");  // should have been
                                                     //    "WeightingFinishedEvent"
                IAFactSet respFactSet;  // create a fact set
                respAtom.setTerms(""); // clear any current atom terms
                respAtom.addTerm("WeightingFinishedEvent");
                respFactSet.add(respAtom);  // add atom to the current fact set
                IATriggerEvent weightingFinishedEvent;
                weightingFinishedEvent.setType("WeightingFinishedEvent");
                weightingFinishedEvent.setFactSet(respFactSet);
                notify(weightingFinishedEvent);
        }
        else
                cerr << "Updating of TermAccum can not be done !" << endl;
        listFile.close();
}

void NewInterestAdapter::RemoveStop(char *fn, fstream &tlist)
{
        char term[128];
        StrList words;
        DFA stopfilter;
        int c = 0;

        FILE *stream;

        words = StrListCreate ();  // initialize the structure
        StrListAppendFile (words, "stop.wrd");
        stopfilter = BuildDFA (words);

        if (!(stream = fopen(fn, "r")) ) exit (1);
        do
        {
                tlist << term << endl;  // write term to temp vector file
        } while (GetTerm(stream, stopfilter, 128, term) != NULL);
        tlist.seekg(0); // positions file pointer to start
        (void)fclose (stream);   // close the filtered document
}


short NewInterestAdapter::StemTerms(fstream &tlist, fstream &listf)
{
        char term[64];
        short NoOfTerms = 0;

        while (GetNextTerm(tlist,64,term)) {
                if (Stem (term)) {
                        listf << term << endl;  // write term to listf file
                        ++NoOfTerms;
                }
        }
        tlist.close(); // close the temp vector file
        listf.seekg(0); // positions file pointer to start
        return NoOfTerms;
}

bool NewInterestAdapter::GetNextTerm(fstream &Lstream,
                                                int size, char *term)
{
```

```
        if (!Lstream.eof()) {
                Lstream.getline(term, size); // read line
                return true;
        }
        else
            return false;
}


short NewInterestAdapter::CalculateDocWeights(fstream &listf,
                                        fstream &vecFile, short TotWords, unsigned int &tptr)
{
        unsigned int i, wFreq, NoOfVecElements = 0;
        char word[MAX_WORD_SIZE];

        char **TListArray;
        tptr = 0;

        TListArray = new char*[TotWords];  // array of pointers
        for (int j = 0; j <= TotWords; j++)  // allocate mem pointed to by TListArray[j]
                TListArray[j] = new char[MAX_WORD_SIZE];

        VectorAtom *docVector = new VectorAtom;
        // VectorAtom docVector;

        for (i = 0; i <= TotWords - 3; i++) {
                if (!listf.eof()) {
                        listf.getline(word, MAX_WORD_SIZE); // read terms into the vector
                        strcpy(TListArray[i], word);
                }
                else
                        cout << "Error! - Terms are" << i << endl <<
                                "       While TotWords is" << TotWords << endl;
        }

        wFreq = 1;
        QSort(TListArray, 0, TotWords);

        // calculate weight of each term
        for (i=0; i <= TotWords - 4; i++) {
                if (!strcmp(TListArray[i], TListArray[i+1])) {
                        ++wFreq;
                }
                else {
                        strcpy(docVector->term, TListArray[i]);
                        docVector->doctf = wFreq;
                        docVector->weight1 = 0.0;
                        docVector->weight2 = 0.0;
                        docVector->weight3 = 0.0;
                        docVector->weight4 = 0.0;
                        docVector->weight5 = 0.0;
                        docVector->weight6 = 0.0;
                        docVector->df = 0;
                        docVector->cumwt = 0.0;
                        WriteVecElement(vecFile, docVector);
                        ++NoOfVecElements;
                        tptr += wFreq;
                        wFreq = 1;
                }
        }
```

```
        // process the last word
        strcpy(docVector->term, TListArray[i]);
        docVector->doctf = wFreq;
        docVector->weight1 = 0.0;
        docVector->weight2 = 0.0;
        docVector->weight3 = 0.0;
        docVector->weight4 = 0.0;
        docVector->weight5 = 0.0;
        docVector->weight6 = 0.0;
        docVector->df = 0;
        docVector->cumwt = 0.0;

        WriteVecElement(vecFile, docVector);
        ++NoOfVecElements;
        tptr += wFreq;

        for (i = 0; i <= TotWords; i++)
                delete [] TListArray[i];
        delete [] TListArray;
        delete docVector;
        cout << "WEIGHTED !" << endl;
        return NoOfVecElements;
}

void NewInterestAdapter::UpdateTempAccum(fstream &vecFile, fstream &accumFile,
                                         short NoOfVecElements, unsigned int &tptr, fstream &weFile)
{
        char Tstream[80];
        char totstream[10];
        unsigned int docterms = 0;
        unsigned int tot;

        fstream accumCopy("tempcopy.dat", ios:: trunc | ios::in | ios::out);
        if (!accumCopy) {
                cout << "tempcopy.dat can not be opened. " << endl;
                exit(1);
        }
        accumFile.seekg(0); // positions file pointer to start
        if (!accumFile.eof()) {
                accumFile.getline(totstream, 10); // first line is tot words
                tot = atoi(totstream);
        }

        // copies tempacum.dat to temporary file
        while (!accumFile.eof()) {
                accumFile.getline(Tstream, 80);
                accumCopy << Tstream << endl;
        }
        accumFile.close();
        // deletes content of tempacum.dat
        accumFile.open("tempacum.dat", ios::trunc | ios::in |ios::out);
        if (!accumFile) {
                cout << "tempacum.dat can not be opened. " << endl;
                exit(1);
        }
        MergeVecs(vecFile, accumCopy, accumFile, tptr, weFile); // update tempacum.dat
}

bool NewInterestAdapter::MergeVecs(fstream &dFile,fstream &cFile,
                                   fstream &aFile, unsigned int &tptr, fstream &wFile)
```

```
            // process the last word
            strcpy(docVector->term, TListArray[i]);
            docVector->doctf = wFreq;
            docVector->weight1 = 0.0;
            docVector->weight2 = 0.0;
            docVector->weight3 = 0.0;
            docVector->weight4 = 0.0;
            docVector->weight5 = 0.0;
            docVector->weight6 = 0.0;
            docVector->df = 0;
            docVector->cumwt = 0.0;

            WriteVecElement(vecFile, docVector);
            ++NoOfVecElements;
            tptr += wFreq;

            for (i = 0; i <= TotWords; i++)
                    delete [] TListArray[i];
            delete [] TListArray;
            delete docVector;
            cout << "WEIGHTED !" << endl;
            return NoOfVecElements;
}

void NewInterestAdapter::UpdateTempAccum(fstream &vecFile, fstream &accumFile,
                                         short NoOfVecElements, unsigned int &tptr, fstream &weFile)
{
        char Tstream[80];
        char totstream[10];
        unsigned int docterms = 0;
        unsigned int tot;

        fstream accumCopy("tempcopy.dat", ios:: trunc | ios::in | ios::out);
        if (!accumCopy) {
                cout << "tempcopy.dat can not be opened. " << endl;
                exit(1);
        }
        accumFile.seekg(0); // positions file pointer to start
        if (!accumFile.eof()) {
                accumFile.getline(totstream, 10); // first line is tot words
                tot = atoi(totstream);
        }

        // copies tempacum.dat to temporary file
        while (!accumFile.eof()) {
                accumFile.getline(Tstream, 80);
                accumCopy << Tstream << endl;
        }
        accumFile.close();
        // deletes content of tempacum.dat
        accumFile.open("tempacum.dat", ios::trunc | ios::in |ios::out);
        if (!accumFile) {
                cout << "tempacum.dat can not be opened. " << endl;
                exit(1);
        }
        MergeVecs(vecFile, accumCopy, accumFile, tptr, weFile); // update tempacum.dat
}

bool NewInterestAdapter::MergeVecs(fstream &dFile,fstream &cFile,
                                   fstream &aFile, unsigned int &tptr, fstream &wFile)
```

```
{
        // Implementation of cosequential algorithm for merging two vectors
        // namely, the new doc vector and temp. accumulator vector

        float weight;
        unsigned int docno;
        VectorAtom *docVector = new VectorAtom;
        VectorAtom *accumVector = new VectorAtom;
        bool MORE_VECS = true;

        dFile.seekg(0);
        cFile.seekg(0);
        aFile << tptr << endl;
        cout << "Enter Doc No.  ";
        cin >> docno;

        MORE_VECS = ReadVecElement(dFile, docVector);
        MORE_VECS = ReadVecElement(cFile, accumVector);
        cout << "Here" << endl;
        while (MORE_VECS) {
                if (strcmp(docVector->term, accumVector->term) < 0) {
                        docVector->df += 1;
                        WriteAcumVec(docno,aFile, docVector, tptr);
                        MORE_VECS = ReadVecElement(dFile, docVector);
                }
                else if (strcmp(docVector->term, accumVector->term) > 0) {
                        WriteVecElement(aFile, accumVector);
                        MORE_VECS = ReadVecElement(cFile, accumVector);
                }
                else {
                        // add frequency,recalculate weight and write to accumFile
                        accumVector->doctf += docVector->doctf;
                        accumVector->df += 1;
                        WriteAcumVec(docno, aFile, accumVector, tptr);
                        MORE_VECS = ReadVecElement(dFile, docVector);
                        MORE_VECS = ReadVecElement(cFile, accumVector);
                }
        }
        delete docVector;
        delete accumVector;
        return true;
}

bool NewInterestAdapter::WriteAcumVec(int docn, fstream &file, VectorAtom *vec, unsigned int &tptr)
{
 // writes to tempacum.dat file
 float tempw;
 switch (docn) {
        case 1: {
                cout << vec->weight1 << endl;
                cout << tptr << endl;
                tempw = vec->doctf;
                cout << vec->df << endl;
                cout << vec->doctf << endl;
                vec->weight1 = tempw/tptr * vec->df;
                cout << vec->weight1 << endl;
                break;
                }
```

```
            case 2: {
                    vec->weight2 = (float) vec->doctf/tptr * vec->df;
                    break;
                    }
            case 3: {
                    vec->weight3 = (float) vec->doctf/tptr * vec->df;
                    break;
                    }
            case 4: {
                    vec->weight4 = (float) vec->doctf/tptr * vec->df;
                    break;
                    }
            case 5: {
                    vec->weight5 = (float) vec->doctf/tptr * vec->df;
                    break;
                    }
            case 6: {
                    vec->weight6 = (float) vec->doctf/tptr * vec->df;
                    break;
                    }
            }

            WriteVecElement(file, vec);
            return true;
}

bool NewInterestAdapter::WriteVecElement(fstream &vFile, VectorAtom *Vatom)
{
            // write formatted vector atom to file
            vFile << "(\"" << Vatom->term
                    << "\"" << " \"" << Vatom->weight1
                    << "\"" << " \"" << Vatom->weight2
                    << "\"" << " \"" << Vatom->weight3
                    << "\"" << " \"" << Vatom->weight4
                    << "\"" << " \"" << Vatom->weight5
                    << "\"" << " \"" << Vatom->weight6
                    << "\"" << " \"" << Vatom->cumwt
                    << "\"" << " \"" << Vatom->df
                    << "\"" << " \"" << Vatom->doctf
                    << "\")  " << endl;
            return true;
}

bool NewInterestAdapter::ReadVecElement(fstream &vFile, VectorAtom *Vatom)
   // reads a vector element from file, parses it and stores in Vatom
{
            char *temp = new char[];
            char *vline = new char[300];
            bool TNOTREAD, WNOTREAD, DFNOTREAD, W1NOTREAD, W2NOTREAD,
                W3NOTREAD,  W4NOTREAD, W5NOTREAD, W6NOTREAD, TFNOTREAD;
            int i, j = 0;
            *temp = *vline = '\0';

            if (!vFile.eof()) {
                    vFile.getline(vline, 300);
                    TNOTREAD = W1NOTREAD = W2NOTREAD = W3NOTREAD = W4NOTREAD =
                            W5NOTREAD = W6NOTREAD = DFNOTREAD = WNOTREAD =
                            TFNOTREAD = true;
```

```
for (i = 0; i < MAX_ATOM_SIZE && (TNOTREAD == true || W1NOTREAD == true
        || W2NOTREAD == true || W3NOTREAD == true || W4NOTREAD == true ||
        W5NOTREAD == true || W6NOTREAD == true || DFNOTREAD == true ||
        TFNOTREAD == true || WNOTREAD == true); i++) {
        if (TNOTREAD == true) {
                if (vline[i+2] != '\"')
                        Vatom->term[i] = vline[i+2];
                else {
                        Vatom->term[i] = '\0';
                        cout << Vatom->term << ",";
                        TNOTREAD = false;
                }
        }
        else if (W1NOTREAD == true) {
                if (vline[i+4] != '\"') {
                        temp[j] = vline[i+4];
                        j++;
                }
                else {
                        temp[j] = '\0';
                        Vatom->weight1 = atof(temp);
                        j = 0;
                        temp = "";
                        W1NOTREAD = false;
                }
        }
        else if (W2NOTREAD == true) {
                if (vline[i+6] != '\"') {
                        temp[j] = vline[i+6];
                        j++;
                }
                else {
                        temp[j] = '\0';
                        Vatom->weight2 = atof(temp);
                        j = 0;
                        temp = "";
                        W2NOTREAD = false;
                }
        }
        else if (W3NOTREAD == true) {
                if (vline[i+8] != '\"') {
                        temp[j] = vline[i+8];
                        j++;
                }
                else {
                        temp[j] = '\0';
                        Vatom->weight3 = atof(temp);
                        j = 0;
                        temp = "";
                        W3NOTREAD = false;
                }
        }
        else if (W4NOTREAD == true) {
                if (vline[i+10] != '\"') {
                        temp[j] = vline[i+10];
                        j++;
                }
                else {
                        temp[j] = '\0';
                        Vatom->weight4 = atof(temp);
```

```
                j = 0;
                temp = "";
                W4NOTREAD = false;
        }
}
else if (W5NOTREAD == true) {
        if (vline[i+12] != '\"') {
                temp[j] = vline[i+12];
                j++;
        }
        else {
                temp[j] = '\0';
                Vatom->weight5 = atof(temp);
                j = 0;
                temp = "";
                W5NOTREAD = false;
        }
}
else if (W6NOTREAD == true) {
        if (vline[i+14] != '\"') {
                temp[j] = vline[i+14];
                j++;
        }
        else {
                temp[j] = '\0';
                Vatom->weight6 = atof(temp);
                j = 0;
                temp = "";
                W6NOTREAD = false;
        }
}
else if (WNOTREAD == true){
        if (vline[i+16] != '\"') {
                temp[j] = vline[i+16];
                j++;
        }
        else {
                temp[j] = '\0';
                Vatom->cumwt = atof(temp);
                WNOTREAD = false;
        }
}
else if (DFNOTREAD == true) {
        if (vline[i+18] != '\"') {
                temp[j] = vline[i+18];
                j++;
        }
        else {
                temp[j] = '\0';
                Vatom->df = atoi(temp);
                j = 0;
                temp = "";
                DFNOTREAD = false;
        }
}
else if (TFNOTREAD == true) {
        if (vline[i+20] != '\"') {
                temp[j] = vline[i+20];
                j++;
        }
```

```
                                        else {
                                                temp[j] = '\0';
                                                Vatom->doctf = atoi(temp);
                                                j = 0;
                                                temp = "";
                                                TFNOTREAD = false;
                                        }
                                }

                        }
                        return true;
                }
                else
                        return false;
                delete [] tempwt;
}

bool NewInterestAdapter::CheckInUM(fstream &umFile, char *givenTerm)
{
 // checks whether a term is already in user model
        VectorAtom *umVector = new VectorAtom;

        umFile.seekg(0);
        ReadVecElement(umFile, umVector);

        while (!umFile.eof()) {
                if (strcmp(umVector->term, givenTerm) < 0) {
                        ReadVecElement(umFile, umVector);
                }
                else if (strcmp(umVector->term, givenTerm) > 0) {
                        delete umVector;
                        return false; // term is not available
                }
                else {
                        delete umVector;
                        return true; // term is available
                }
        }
        delete umVector;
        return false;
}

bool NewInterestAdapter::AddToUM(fstream &umFile, VectorAtom *vElement)
{
        // adds a term to the user model
        int i = 0, j = 0, NoOfElems, t=0;
        VectorAtom **tempVec1;
        VectorAtom **tempVec2;

        tempVec1 = new VectorAtom *[1000];
        for (i = 0; i < 1000; i++)
                tempVec1[i] = new VectorAtom[sizeof(VectorAtom)];

        tempVec2 = new VectorAtom *[1000];
        for (i = 0; i < 1000; i++)
                tempVec2[i] = new VectorAtom[sizeof(VectorAtom)];
        VectorAtom *UMvec = new VectorAtom[sizeof(VectorAtom)];
        umFile.seekg(0);
        i=0;
```

```
                                        else {
                                                temp[j] = '\0';
                                                Vatom->doctf = atoi(temp);
                                                j = 0;
                                                temp = "";
                                                TFNOTREAD = false;
                                        }
                                }

                        }
                        return true;
                }
                else
                        return false;
                delete [] tempwt;
}

bool NewInterestAdapter::CheckInUM(fstream &umFile, char *givenTerm)
{
 // checks whether a term is already in user model
        VectorAtom *umVector = new VectorAtom;

        umFile.seekg(0);
        ReadVecElement(umFile, umVector);

        while (!umFile.eof()) {
                if (strcmp(umVector->term, givenTerm) < 0) {
                        ReadVecElement(umFile, umVector);
                }
                else if (strcmp(umVector->term, givenTerm) > 0) {
                        delete umVector;
                        return false; // term is not available
                }
                else {
                        delete umVector;
                        return true; // term is available
                }
        }
        delete umVector;
        return false;
}

bool NewInterestAdapter::AddToUM(fstream &umFile, VectorAtom *vElement)
{
        // adds a term to the user model
        int i = 0, j = 0, NoOfElems, t=0;
        VectorAtom **tempVec1;
        VectorAtom **tempVec2;

        tempVec1 = new VectorAtom *[1000];
        for (i = 0; i < 1000; i++)
                tempVec1[i] = new VectorAtom[sizeof(VectorAtom)];

        tempVec2 = new VectorAtom *[1000];
        for (i = 0; i < 1000; i++)
                tempVec2[i] = new VectorAtom[sizeof(VectorAtom)];
        VectorAtom *UMvec = new VectorAtom[sizeof(VectorAtom)];
        umFile.seekg(0);
        i=0;
```

```
        ReadVecElement(umFile, tempVec1[i]);
        while (!umFile.eof()) { // read terms into tempVec1
                ReadVecElement(umFile, tempVec1[i]);
                i++;
                NoOfElems++;
        }
        for (i = 0; i < NoOfElems; i++) {
                if (strcmp(tempVec1[i]->term, vElement->term) < 0) {
                        continue;
                }
                else if (strcmp(tempVec1[i]->term, vElement->term) > 0) {
                        t = i;
                        for (j = 0; i < NoOfElems; j++) { // copy terms below to tempVec2
                                tempVec2[j] = tempVec1[i];
                                i++;
                        }
                        tempVec1[t] = vElement;
                        j = 0;
                        for (i = t+1; i < NoOfElems + 1 ; i++) {
                                tempVec1[i] = tempVec2[j];
                                j++;
                        }
                        umFile.close();
                        umFile.open("usermodl.dat", ios::trunc | ios::out);
                        for (i = 0; i < NoOfElems + 1 ; i++) {
                                WriteVecElement(umFile, tempVec1[i]);
                        }
                        delete UMvec;
                        delete [] tempVec1;
                        delete [] tempVec2;
                        return true; // term is not available
                }
                else {
                        delete UMvec;
                        cerr << "addition to User Model faiure !" << endl <<
                                "element " << vElement << "is already available" << endl;
                        return false; // term is available
                }
        }
        tempVec1[i+1] = vElement; // i.e all vecs are less than vElement
        umFile.close();
        umFile.open("usermodl.dat", ios::trunc | ios::out);
        for (i = 0; i < NoOfElems+1 ; i++) {
                WriteVecElement(umFile, tempVec1[i]);
        }
        delete UMvec;
        delete [] tempVec1;
        delete [] tempVec2;
        return true;
}

VectorAtom** NewInterestAdapter::FindTermsOfInterest(fstream &accumFile, float th, int &cptr)
{
        // identifies a term that meets the new interest threshold
        int i = 0;
        VectorAtom **acumVec;

        VectorAtom *tempVec = new VectorAtom[sizeof(VectorAtom)];

        // file to store results
```

```
        fstream resultFile("result.dat", ios::out | ios::in |ios::trunc);
        if (!resultFile) {
                cout << "result.dat can not be opened. " << endl;
                exit(1);
        }

        acumVec = new VectorAtom *[2000];
        for (i = 0; i < 2000; i++)
                acumVec[i] = new VectorAtom[sizeof(VectorAtom)];

        accumFile.seekg(0);
        ReadVecElement(accumFile, tempVec);
                                cout << "here" << endl;
        i=0;
        cptr = 0;
        while (!accumFile.eof()) { // read terms into tempVec
                ReadVecElement(accumFile, tempVec);
        }
                if (tempVec->weight < th) {
                        continue;
                }
                else {
                        strcpy(acumVec[i]->term, tempVec->term);
                        acumVec[i]->weight = tempVec->weight;
                        resultFile << setiosflags(ios::left) << setw(25)
                                << acumVec[i]->term << acumVec[i]->weight << endl;
                        cout << setiosflags(ios::left) << setw(25)
                                << acumVec[i]->term << acumVec[i]->weight << endl;
                        i++;
                        cptr++;
                }
        }

        resultFile << cptr << endl;
        return acumVec;
}

IAAdapter *newAdapter(const char *domain, void *parms)
{
 // function for creating new instances of New Interest Adapters.

  IAAdapter *adapter = new NewInterestAdapter();

  if (!adapter) {
    ITHROW(IException("newAdapter() is unable to create a new"
      " NewInterestAdapter object."));
  }
  return adapter;
}
```

DECLARATION

This thesis is my original work and has not been submitted for a degree in any other university.

Dawit Yimam Seid

May, 1998

The thesis has been submitted for examination with our approval as university advisors.

Tesfaye Biru

May, 1998