Information Filtering

with Collaborative Interface Agents

Report by Tomas Olsson
Department of Computer and Systems Sciences
The Royal Institute of Technology
March 1998

This report describes a distributed approach to social filtering based on the agent metaphor. Firstly, previous approaches are described, such as cognitive filtering and social filtering. Then a couple of previously implemented systems are presented and then a new system design is proposed. The main goal is to give the requirements and design of an agent-based system that recommends web-documents. The presented approach combines cognitive and social filtering to get the advantages from both techniques. Finally, a prototype implementation called WebCondor is described and results of testing the system are reported and discussed.

This thesis corresponds to the effort of 20 full-time working weeks.

Preface

The work described in this report was accomplished at Ellemtel Utvecklings AB in Stockholm, Sweden. Ellemtel Utvecklings AB is owned by equal parts of LM Ericsson AB and Telia AB. LM Ericsson AB is a world-leading supplier of equipment for telecommunication systems and related terminals, and Telia AB is Sweden's largest telecommunication operator.

Ellemtel Utvecklings AB is a development company in the area of end-user services and the corresponding improvements in the telecommunications and data communications networks.

The original idea for the work in this report was my own extensions to an earlier project description at Ellemtel Utvecklings AB for a single intelligent agent.

I first would like to thank my supervisor Bertil Yvling at Ellemtel Utvecklings AB for the courage to let me form the project in my own way and then, I would like to thank my supervisor Dr. Henrik Boström at the Department of Computer and Systems Sciences, a joint department of the Royal Institute of Technology and Stockholm University.

I also would like to thank Dr. Fredrik Kilander at the Department of Computer and Systems Sciences for his help to find a way to form the user models.

Table of contents

1 Introduction

1.1 The Problem Statement

Today when a user wants to find interesting documents on the WWW, the user has to actively search the web. There are some tools to help the user to find relevant documents and most of today's tools for searching the WWW use query-based techniques. Because of the properties of such techniques the results of a query often include many irrelevant documents (low precision) and exclude many relevant documents (low recall).

The usual way to get around this problem is to use different retrieval and filtering techniques that are able to adapt to the user automatically or manually.

A program that actively tries to help the user is called a User Agent. A User Agent that automatically adapts to the user is called an Interface Agent. If the Interface Agent tries to find relevant documents for the user, it is called an Interface Agent for Information Filtering — an agent that actively finds and filters information for the user.

Existing agents for information filtering do not usually benefit from the work of other agents. In particular, they do not use information already accumulated in other agents. In this thesis, it is studied if the performance of an agent can be improved by co-operation, that is, if Information Filtering with Collaborative Interface Agents is a successful approach to the problem of low precision and recall.

1.2 Contributions of this thesis

The contribution of this thesis is a distributed collaborative system that combines the techniques of content-based and social filtering. A prototype implementation of the collaborative part of a system that recommends web documents is described, and results of testing are reported.

The report has the following structure: Section 2 describes previous approaches and it describes briefly the proposed system in comparison to the previous ones. Section 3 contains an analysis of the problem area and criterions for the design. Section 4 describes the proposed system. Section 5 describes the implemented prototype WebCondor. Section 6 describes the simulated user tests and their results. Section 7 describes major problems with systems such as the proposed one. Section 8 contains a short summary of what actually was achieved in this thesis and a short discussion on future work.

2 Information Filtering with Collaborative Interface Agents

2.1 Definition of important concepts

Software Agent

A software agent (or agent) is a software program that actively or autonomously performs its work. The term agent is used in different ways in different contexts and it has been a subject of discussion for a long time among scientists. A good introductory paper about software agents is Hyacinth S. Nwana's *Software Agents: An Overview* [Nwana 1996].

Interface Agent

An interface agent is a software agent that adapts to and actively helps a user. For example, an interface agent could monitor the user's actions and give advice how to perform them in a "better" way. A good example of interface agents is given by Pattie Maes et al. in their paper *Collaborative Interface Agents* [Lashkari, Metral & Maes 1994]. They describe an interface agent that helps a user to sort e-mail.

Multi-Agent System (MAS)

A multi-agent system is a group of interacting agents. The agents can be situated in the same computer or distributed at different places using a network to communicate.

Collaborative or Co-operative Agents

Collaborative or co-operative agents are agents in a multi-agent system, where the agents are not just existing together but they are also actively helping each other to achieve their goals, where the goals can be similar or completely different.

2.2 Previous Work

In the area of information filtering and information retrieval, there is a lot of previous work related to the proposed system, and in the area of collaborative agents, there is an even more diverse collection of related work.

The two concepts information filtering and information retrieval are often hard to distinguish from each other and the two concepts are usually intermixed. Both concepts address the problem of getting wanted information, but in information retrieval, one tries to find all relevant documents in a collection, while in information filtering, one tries to remove all irrelevant documents from a collection [Ejdeberg 1997]. In information filtering, the collection can be seen as a stream of documents trying to reach the user and unwanted documents are removed from the stream. In information retrieval, one could say that the user tries to pick wanted documents from the collection. Section 2.2.1 and 2.2.2 describes adaptive information retrieval respectively adaptive information filtering.

In the area of collaborative agents, there are many different sorts of applications, not only for searching and for finding information. In section 2.2.3, some collaborative agent systems are described with the focus on sharing and handling information.

2.2.1 Adaptive Information Retrieval

An adaptive information retrieval system is usually based on a query and relevance feedback from the user [Sheth 1994]. The user queries the system and then the user in-

dicates if the retrieved documents match the wanted information. The system forms a new query from the old query and the matching documents, and new documents are retrieved. This is repeated until the user is satisfied.

2.2.2 Adaptive Information Filtering

There are mainly two ways to filter information adaptively: cognitive filtering and social filtering [Sheth 1994; Maltz 1994]. They are presented in section 2.2.2.1 and in section 2.2.2.2 respectively. There are also approaches that try to combine the cognitive and the social filter techniques and thereby gain advantages from both of them. Combined filtering approaches are presented in section 2.2.2.3.

2.2.2.1 Cognitive Filtering

The cognitive or content-based approach analyses the content of the information and compares it to a model of the user (a user model). The closer match, the more likely it will suit the user and at some threshold value for the "closeness", a document is proposed. The relevance feedback from the user, for a proposed document, is used to change the user model. This is very close to how adaptive information retrieval works.

A content-based system can discover new interesting information if it is similar to previously encountered documents, but it removes interesting documents if they did not have the same sorts of content. The problem to find interesting documents of a sort previously not encountered is called the problem of *serendipity* [Firefly 1997]. Cognitive filtering is often performed by stand-alone applications and therefore, it also has the problem of *cold-start* [Lashkari, Metral & Maes 1994], which means that it takes a while before the system has learned to filter in a good way. There are also problems with *the static view of the content providers* and *the amount of downloading*. The static view of the providers means that the providers are not actively delivering documents. The usual way to solve this problem is by polling the providers for updates. The problem of downloading means that agents have to download all information by themselves to find what is interesting. If all users have such agents, it will be many accesses to the providers.

An example of cognitive filtering is Beerud Seth's *A Learning Approach to Personalised Information Filtering* [Sheth 1994]. He presents an artificial evolution approach to build the user model. Artificial evolution is a machine learning method that tries to imitate biological evolution. In artificial evolution, one tries to evolve a good solution from a large number of possible solutions by mutation, reproduction and competition. As genotype, he uses the user models, but he also lets the models learn during their lifetime to make the adaptation go faster. It is called the "Baldwin effect". Each model is used to search for documents and recommend them to the user. The user's response is used to change the fitness of the model and to let it adapt during its lifetime. This approach, because of the evolutionary algorithm, tries to benefit from an effective parallel search to find the best models.

Another good example is *The Info Agent* [D'Aloisi & Giannini 1995]. The Info Agent consists of three different co-operating agents, the Interface Agent, the Internal Services Agent and the External Retrieval Agent. The Interface Agent builds a user model and it uses the model to guide the two other agents' search for documents. The system is designed to be flexible to extensions and changes.

2.2.2.2 Social Filtering

Social filtering is solely based on what different users are recommending. If the user likes certain documents, the user will probably like some other documents because other users with the same preferences did. Cognitive filtering works reasonable well with text based documents, but social filtering is well suited for other domains too, such as pictures, movies, music, etc. The system relies in other users' opinions and not in the content of the documents, therefore, the user can get documents of a sort previously not encountered and the system does not have the problem of serendipity. A problem is however that some users must have liked a document before it can be proposed to a new user. It might require quite a few users in the system before it starts to work well, but when there is a sufficient amount of users, new users can benefit from the previous users' ratings. This means that the problem of cold-start is reduced when the system does not have to start from scratch every time a new user begins to use it.

A problem with systems based on social filtering is that they are mainly centralised [Foner 1997]. This is a bottleneck for scalability and availability, and a risk for the integrity of the users. The scalability is a problem because the straightforward solution is to compare all users' models to match them, which is a search of quadratic-order. The integrity of the users is at risk because of the sensitive information the system has at the same place. What happens if somebody unauthorised gets hold of it? The availability is a problem because there is a single point where a failure can happen. If a failure occur, it might lead to consequences such as letting some unauthorised person get hold of the user models.

A social filtering system for net news is presented in *Distributed Information for Collaborative Filtering on Usenet Net News* [Maltz 1994]. In the system, each user can read an article and vote for or against it. The votes are then sent to a vote server where they are grouped together and shared with other vote servers. The servers aggregate all the different readers' opinions into one collective opinion. This aggregated opinion is then used by newsreaders to filter shown articles.

Firefly[1] is one of the most known companies using social filtering [Firefly 1997]. They call their technique Feature-Guided Automated Collaborative Filtering. This filtering technique builds a model of each user with their opinions for different documents. The documents are divided in different groups (classes of documents) and for each group, the users are clustered in a nearest "neighbour" stile. With a nearest neighbour technique means that the most similar examples to a given example are used to compute a prediction about the given example. For each group of documents, the users' opinions in the same cluster are compared. To find documents to recommend it matches all users in a cluster (this is of course a simplified explanation). If two users have approximately the same opinions for most of the documents in a group, but they have not read and rated all of them, then it is likely that the users have the same opinion for the unread documents of the group too.

2.2.2.3 Combined filtering

In a master thesis at the Telia Research AB, they have designed a system that uses both cognitive filtering and social filtering [Ejdeberg 1997]. They are using a user agent that does cognitive filtering according to a user model and all the user agents let the same

[1] **Firefly** is a trademark of Firefly Network, Inc.

broker search the web for them. The broker is in addition building a social filter to effectively recommend new documents.

Fab is also a system using both social and content-based filtering [Balabanovic 1996; Balabanovic & Shoham 1997]. Their system consists of two kinds of agents, selection agents (one for each user) and collection agents (common to the users). The collection agents collect information and deliver documents to a central repository from which the selection agents filter interesting documents according to user models. Responses from the users are used to modify both the selection agents and the collection agents. If a document matched a user well, the collection agent that found this document is "rewarded" and the agent is thus able to specialise to a certain kind of documents.

2.2.3 Collaborative Agents

The most well known example of collaborative interface agents is the paper of Pattie Maes et al. called *Collaborative Interface Agents* [Lashkari, Metral & Maes 1994]. They describe a system with interface agents that help a user to handle e-mail. They compare the performance between a single interface agent and two collaborating interface agents. The interface agent builds a model of how the user sorts incoming e-mail and it tries to sort the mail automatically. To improve the performance and avoid the problem of cold-start the agent can learn which other agents to ask for advice. Given an e-mail — How much can the agent trust the other agents' advice? To learn whom to trust the agent asks the other agents to give an advice for an already known mail. If the other agents give the advice to handle it the same way the user already has done, the agent increases the trust for the other agents' ability to handle this kind of mails (or it decreases the trust if it is a wrong advice). A problem not addressed is how the agents are supposed to find each other. If there are many agents, this might be hard. However, this problem is addressed by the other systems described in the rest of this section.

Another interesting example with collaborative agents is Leonard Foner's *Yenta* [Foner 1997]. In Yenta, the agents try to build models of the users' interests and use the models to find other agents with the same interests. An agent builds the model of its user from any available texts describing the user's interests, for example, the user's e-mail or produced papers (in other words it is a content-based approach). Foner mainly addresses the problem for agents to find each other without any central control. His approach is to let the agents self-organise into clusters with other agents with similar interests. He achieves this self-organising in the same manner as we humans find other people with the same interests, that is by referrals; through knowledge about other people and the other people's knowledge about further other people, etc. The referrals consist of lists with other agents or more precisely, models of the other agents' user models. A problem in Yenta is that the agents compare their interests directly by comparing their user models, which means that the agents have to use standardised data representations or comparison methods. This makes it hard to add to the system new agents working in a different way.

Henry Kautz et al. take a similar approach to Foner's *Yenta* in their paper *Agent Amplified Communication* [Kautz, Milewski & Selman 1996]. They are also using agents to find people, but they are trying to find experts in some domain through referrals. They let the agent build a user model of the user's e-mail and it uses the user's e-mail communication pattern to generate referrals to other users. A referral is built as a simpler user model. If the user needs an expert in a certain domain, the user submits a text description of the domain to its agent. The agent uses the description to ask other agents.

If the other agents are not able to answer, they might give referrals to additional other agents.

ACORN is another example [Marsh & Masrour 1997]. In ACORN, a user can create smart documents to tell other users about documents and create smart queries to search for documents. A smart document is a mobile agent with the document represented as manually or automatically formed metadata. It starts travelling around between users chosen from a list of possibly interested user. Each user is represented by a client that can filter interesting agents based on a user model. The smart document travels as long as it has new users to visit. During its lifetime, it can learn about new users from; its own user, special meeting places for agents called cafés and the clients of the other users. A smart query is just a smart document with the goal to find the described information, and now and then, it reports found documents to its user. It finds documents by meeting smart documents and by using search engines. A weakness of the ACORN system is that it is dependent on metadata and methods to create it, which means that there must be standardised protocol to describe documents. Because the metadata of a document does not hold all of the information the original document contained, some information is lost that another agent could have used. The loss of information makes it, as in Yenta, harder to add to the system new agents working in a different way. However, a new agent can handle the metadata in another way, which makes the system somewhat better than Yenta. The system also has a problem of limited centralisation. It uses special places for the agents to meet and to get to know more agents.

2.3 The Proposed Approach

The proposed approach for information filtering is based on *Collaborative Interface Agents*. There are three types of agents, *the Interface Agent*, *the Interest Agent* and *the Retrieval Agent.*

An Interface Agent is an interface between the system and a user. Through the Interface Agent, the user will have access to the whole WWW via a web-browser and the user will get recommendations of documents from other users. The Interface Agent forms its user model by letting the user create categories and sort interesting documents into them (explicit feedback). For each category, the Interface Agent creates an Interest Agent.

An Interest Agent has mainly three tasks: Firstly, it finds users with similar interests represented by other Interest Agents and secondly, it recommends or routes documents to other agents. This can be seen as if the Interest Agent tells the other agents what sorts of documents it wants, which also means that for it to be recommended documents it has to recommend documents. Thirdly, it proposes documents to the Interface Agent based on the content of the document (cognitive filtering) or the trust in the recommending Interest Agent (social filtering).

To manage to perform these tasks, the Interest Agent builds models of some other Interest Agents. The model of an Interest Agent consists of a confidence value for the Interest Agent and an interest model of its interest. The confidence value states how much it can be trusted for recommendations and the value is based on previously sent good or bad recommendations (a social aspect). The interest model is used to choose where to recommend or route documents and the model is based on the content of all of its previously sent recommendations (a cognitive aspect).

Attached to the Interest Agent, there is a Retrieval Agent. The Retrieval Agent does actively search for documents for the Interest Agent. It uses the context of the links to

build a model of what links on a page it should follow. It uses feedback from the Interest Agent to model the context of the links. The connections of the agents are shown in figure 1.

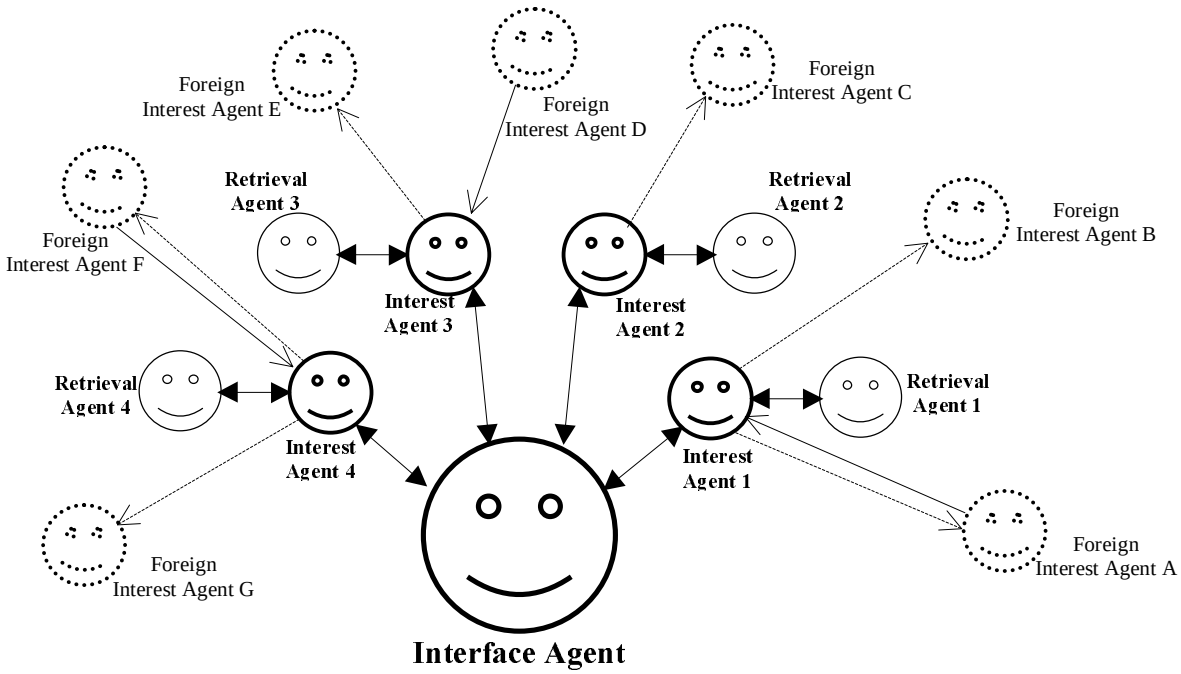

Figure 1: An Interface Agent and its Interest Agents with Retrieval Agents and other users' Interest Agents.

In this way, one is able to get the same advantages as social filtering has and still have the advantages cognitive filtering has. One can get documents recommended by other users, which means, that the system is able to find documents with properties not earlier encountered and thereby, it solves the problem of serendipity. The problem of cold-start is reduced because a new user benefits from the other users already in the system. Although, it can take a while before the agent of the user has learned what other agents to trust. The advantage from the cognitive technique is that the agent (via the Retrieval Agent) is able to find documents of a similar sort that no other user has seen before

The proposed system could be seen as an information retrieval system where the query and feedback loop is a continuous process and the collection of documents is the WWW. The different Interest Agents correspond to the user's queries. On the other hand, one could say that the documents sent between the agents are a stream from which the agents remove unwanted documents and therefore the system should be regarded as an information filtering system. Information retrieval is more of a short-term process where the user's goal is to find some specific information, while information filtering is more of a long-term process where the user's goal is to find interesting information. It seems more reasonable to call the system an information filtering system.

The difference between the proposed approach and previous suggested approaches is foremost that it tries to combine the techniques of content-based and social filtering in a distributed solution with the expectation to gain advantages of both techniques. The proposed solution is completely distributed and thereby, it has no centralised parts and therefore it does not suffer from the corresponding disadvantages of scalability, robustness, etc.

Some differences compared to other distributed systems:

- The clustering of similar agents in the proposed system is not based on the similarities between the content of their interest models as in Yenta, ACORN and the system presented by [Kautz, Milewski & Selman 1996]. Instead, the similarities between the Interest Agents are measured by the social connections (the trust).
- By sending documents between the agents, the proposed system reduces the amount of downloads and makes it possible for a content provider to send information directly to interested users. In ACORN, both the user that wants to find information and the user that wants to deliver information are supported. In the proposed approach, this is not explicitly addressed, but a user that wants to deliver information only has to create the document and sort it into a category (interest) and thereby send it to presumably interested users. This might be a problem, however, if there are owners of the information that want the users to download documents directly from their servers (maybe they are charging the user). The proposed system would work like a dynamic distributed cache, where a user is not forced to download documents directly from the source.
- The proposed system recommends or routes information, which means that the wrapper of the information sent to another agent must be standardised, but the agents can represent interests of users or models of other agents in different ways. This makes it easy to add to the system new agents working in a different way, which was hard in both Yenta and ACORN.

3 The System Design Criterions

Based on Höök's *A framework for adaptive systems* [Höök 1996] I have analysed the area of information filtering and information retrieval with web browsers on the World Wide Web to design the multi-agent system.

3.1 The Analysis of the Problem Area

The massive amount of information on the World Wide Web (WWW) makes it hard for users to navigate. The users are overwhelmed by the huge number of documents and they are cognitively overloaded by the task of filtering all found documents to get the interesting ones. This makes the problem an area well suited for adaptive techniques and because of the users' different opinions on what documents they find interesting, a solution that adapts to each user is very suitable.

The main problem is how a user finds what the user wants on the World Wide Web. I have classified the use of the web in three categories (users' different goals).

- Surfing: The users are discovering new interesting documents and their purpose is to entertain themselves.
- Searching specifically: The users are searching for specific information, for example the home page of an author.
- Searching broadly: The users are searching for information in some different areas, for example, horses, mathematics, etc.

Marko Balabanovic et al. classify the access to the web in two categories, searching and browsing [Balabanovic & Shoham 1995]. The first category corresponds to my two last search categories and the second is the same as my surfing category. I have chosen to make a distinction between searching specifically and broadly because I think there is a significant difference between the two ways of searching. The specific one needs a different solution than the broadly one.

It is known that users are not normally strictly following a plan, they usually adapt to the different circumstances [Waern 1996]. This means that they would usually mix the three different categories. If you are surfing you might find interesting documents in a certain area and you want more of these, which means that you start to search broadly or may be specifically. In the same manner, you might be searching and suddenly you find some interesting documents for which you were not searching. Now you start surfing around to see if they there are some more interesting documents linked to these documents. This means that one can not completely separate the solutions to the three problem categories.

To assist the users to find what they think is interesting there are mainly search engines. Such engines use mostly keyword based document retrieval techniques, that is, the users have to formulate queries to find interesting documents. Other solutions that try to make the surfing and searching easier are personal agents which based on the users' preferences recommend documents, collaborative filtering that recommend documents based on other users opinions and programs based on information retrieval techniques that try to focus the users queries with feedback from the user.

To see if my analysis was relevant to the user population I e-mailed a group of 31 students at KTH and asked them how they accessed the WWW through a web browser. I got 13 answers and the answers indicate that the users are not surfing very much, but

they are searching both broadly and specifically, see Appendix B. When they are surfing, they tend to start from known documents with many references and if they are searching broadly, they tend to use search engines like Yahoo![2] and AltaVista[3]. If they are searching specifically, they usually also use a search engine, foremost AltaVista. None of them used an adaptive system with relevance feedback. What they think is burdensome is mainly the waiting time for downloading documents and formulating search queries. The interview did not reveal any other area of use of web browsers than my three categories. However, it did not confirm my statement that the users mix the different categories but it is supported by other surveys [Waern 1996]. The interview does support the intended goal for this thesis to build a tool for automatically searching and filtering information for the user; thus, they do not have to formulate the queries by themselves. The interview does also support the need for a tool that helps the user to formulate queries (information retrieval), but this is not the intention of this thesis. Notice that this interview does just indicate what students at KTH, mainly computer science students use their web browsers for and it is possible that other groups of users are surfing at a greater extent. I assume that this is how most users use their browsers today.

3.2 System Functionality from a User's View

The system's function is to recommend documents for the users to help the users surfing and searching broadly on the World Wide Web. It does it in such a way that the users are not forced to formulate queries and read all documents by themselves to find the relevant documents.

To make the system usable it should

- *Adapt to the user*. This is according to statements in section 3.1. The system shall build a model of the user through monitoring the user and use this model to find and present interesting documents. It shall let the user give implicit and explicit feedback to get advantages from both of them [Sakagami & Kamba 1997]. There shall also be settings with which the user is able to change the system's working, for example, when it should work and how it presents recommended documents.
- *Act as unnoticed by the user as possible*. The user should not be disturbed if it is not necessary [Oppermann 1994]. The system shall not interrupt the user if it has some suggestions to make. It should be integrated to the existing software environment as much as possible; that is, it should work as much as possible like the normal software environment of the user. The system shall also adapt its working habit to times when it is not loading the computer too much and become an obstacle for the user.
- *Let the user modify its internal state*. The system should be transparent for the user and let the user be in control [Höök 1997]. The user shall be able to modify the system's user model and it should be possible to change the system's rating of documents.
- *Let the user decide when to read recommended documents*. The system must be predictable [Höök 1997]. The presentation of the recommended documents should be in such a way that the user is able to see them but not forced to read them. The recommended documents should not pop-up in the middle of the users' regular

[2] **Yahoo!** is a trademark of Yahoo! inc.

[3] **Alta Vista** is a trademark of Digital Equipment Corporation.

work. See also the item *Act as unnoticed by the user as possible*.

- *Protect the integrity of the user*. The user has to be able to trust the system and have privacy [Höök 1997]. To protect the user, the system should use sufficient security techniques, like cryptographic, etc., to protect the communication and prevent unauthorised access to user models.
- *Use back channels to signal that it is working*. The users can get frustrated if it is not shown that something is going on [Fischer & Christoph 1997]. The system should indicate in some area of the user interface what it is doing and that it is working.

4 The Proposed Multi-Agent Architecture Design

To get a distributed solution based on both cognitive and social filtering, the proposed system is designed as a multi-agent system (MAS). Clearly, a domain of many distributed users is well suited for a distributed solution. Some inherent advantages with multiple agents are according to [Stone & Veloso 1997]

- Speed through parallel computing
- Robustness through agents with redundant functions
- Scalability because of the modular approach.

In the proposed approach, the different users' agents are able to search the WWW in parallel, because they are co-ordinating their behaviours. The system will be robust to malfunctioning agents, because the agents that depended on the malfunctioning agents shared the same interest which means that if an agent disappears it just stops recommending documents and if it starts to recommend in a wrong way the trust for it will decrease and lastly it will be ignored. The scalability of the system is secured by the robustness. It is possible to add a new user without disturbing the old agents, but a problem is to introduce a new agent into the system.

4.1 An Overview of the System

There are three different types of agents in the proposed system, *the Interface Agent*, *the Interest Agent* and *the Retrieval Agent*.

An Interface Agent is an interface between the system and the user. Through the Interface Agent, the user will have access to the WWW and the user will get recommendations of documents. The Interface Agent forms a user model by letting the user create categories and sort interesting documents into them (explicit feedback, but no rating). For each category, the Interface Agent creates an Interest Agent. The goal of an Interest Agent is to find users with similar interests represented by other Interest Agents and the Interest Agent has the task of recommending (actually the user recommends) or routing documents to other Interest Agents. In this way, it is possible to get the same advantages as social filtering has and still have the advantages cognitive filtering has. A user can get documents recommended by other users, but the agents also use the content of a document to decide if it should propose a document to the user.

For the Interest Agent to find Interest Agents it can trust for recommendations and to know to which agents it can send recommendations, it has to model them. This can be done in different ways, for example as in *Yenta* [Foner 1997]. The problem with the approach in Yenta is that it is implementation dependent. The Interest Agents have to know how to compare their interests, and therefore it is not possible to add new Interest Agents with a different inner working. The straightforward solution to this problem is to let them exchange URLs or the unchanged documents in some way. The proposed solution's way to find and model other Interest Agents is inspired by (not based on) Ruud Schoonderwoerd et al. and their paper *Ants for Load Balancing in Telecommunications Networks*[4] [Schoonderwoerd, Holland, Bruten & Rothkrantz 1996]. The proposed sys-

[4] In *Ants for Load Balancing in Telecommunications Networks*, they connect telephone calls using ants to mark which nodes to use. Every ant walks between two arbitrary nodes, leaving a trail of simulated pheromones at each node it is passing. This trail makes it more likely for the next ant to choose the same path backwards. There are many such ants and all leave trails. The compounded trails form a base for choosing routes for calls.

tem uses Information Objects to represent documents recommended by the Interest Agents. Such an object consists mainly of the document content and the identifications of the Interest Agents that recommended or routed it. The Information Object is moved around between the Interest Agents, leaving a trail of recommendations behind that the different Interest Agents can use to model the agents that recommended the contained document. The trail makes it easier for the next similar Information Object to be sent the same path backwards, because an Interest Agent uses its models to choose where to send an Information Object. All trails compounded makes the flow of recommended documents to go to the right Interest Agents, in another way one could also look at it as a way of routing Information Objects between the agents. Observe that there is no clearly stated destination for the Information Objects which means that ordinary routers are not good to use! If an Interest Agent decides that a document in an Information Object is worthy to be evaluated, it sends it to its Interface Agent.

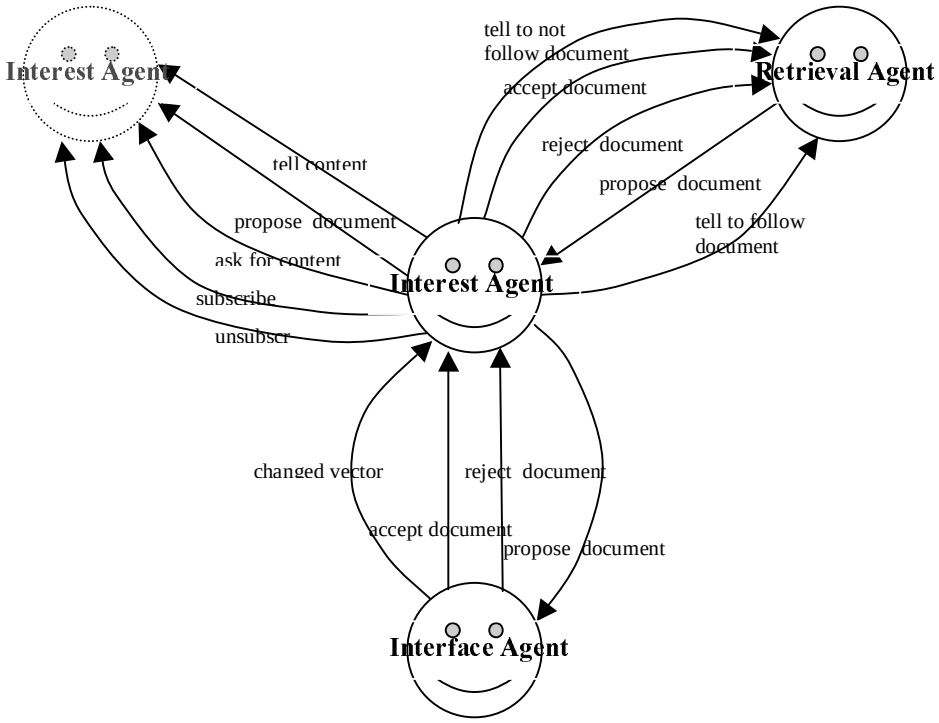

Figure 2: The conversation between the agents.

Another source of recommended documents for an Interest Agent is a Retrieval Agent that is searching the WWW for new documents in a similar way as a WWW robot. A WWW robot is a program that traverses the WWW links [Koster 1995]. Each Retrieval Agent corresponds to an Interest Agent and the Retrieval Agent learns from the Interest Agent what documents to retrieve. One could say that it models the profile of the Interest Agent as the Interface Agent models the profile of the user. Instead of learning the profile based on the documents, it uses the context of the links in a document [Edwards, Bayer, Green & Payne 1996]. If a link is sufficiently interesting, the link is chosen to be followed. This form of adaptive search is more effective than an ordinary WWW robot, because an ordinary WWW robot does not usually take advantage of the built-in relation between documents linked to other documents [Menczer 1997]. When a Retrieval Agent recommends a document, it bases the recommendation on the link context. The Retrieval Agent sends the link to the Interest Agent that evaluates it against its interest model. If the document is not good enough the Interest Agent sends negative feedback

to the Retrieval Agent, otherwise the Interest Agent sends it to the Interface Agent for evaluation.

When an Interface Agent has evaluated a document by asking the user, it sends feedback to the recommending Interest Agent about the evaluation. The feedback consists of an indication of the belonging of the document — if it belonged to the Interest Agent or not. If the Retrieval Agent recommended the document, the Interest Agent will send it feedback, but if the document was recommended by another Interest Agent, the trust for the agent is changed according to the feedback. When the trust for the recommending Interest Agent gets too low it will be "forgotten" and ignored. If the trust is high for the Interest Agent that recommended the document, the Interest Agent will tell the Retrieval Agent to not follow the links from the document. This means that the Interest Agent trusts the other agents to traverse those documents and it makes the use of the different Retrieval Agents more effective; we get a parallel search.

The documents that the user finds, which have not been recommended by another Interest Agent, the Interest Agent sends to the Retrieval Agent as a starting point for information retrievals.

All good links the Interest Agent gets from the Interface Agent will it send to all subscribing Interest Agents unless the category the Interest Agent represents is marked with "don't share". The flag "don't share" for an Interest Agent means that it should not communicate with other Interest Agents.

To make the system communicate with other agents not belonging to this system, the agents should communicate in KQML, which is a popular language for agents. It was designed for knowledge sharing [UMBC KQML Web]: "KQML or the Knowledge Query and Manipulation Language is a language and protocol for exchanging information and knowledge".

4.2 The Interface Agent

An Interface Agent is an interface between the system and the user. Through an Interface Agent, the user will have access to the WWW and get recommendations of documents. An Interface Agent consists of two parts, the browser and the actual agent. The browser makes the agent familiar to the user, and the actual agent is the part that makes an Interface Agent an agent. This is a similar approach as *Letizia* [Lieberman 1995], *WebMate* [Chen] and *WBI* [IBM 1996] has.

4.2.1 The Graphical User Interface

An Interface Agent's graphical user interface consists of the browser and a separate control panel.

The control panel shows recommended documents and gives the user access to the agent part of the Interface Agent. It has buttons called quit, stop, start, preferences and categories.

The quit button ends the agent, stop makes the agent stop working and start restarts the agent. The preferences button opens a window to let the user make changes to the agents acting, such as the threshold for recommended documents, and the categories button opens a window for handling the user model. The panel also has an area where the agent tells the user what it is currently doing and an area that shows in decreasing order of importance recommended documents for each user-defined category.

The user accesses the user model in form of a tree of categorised document in a way similar to a file manager where there are folders (categories), sub-folders (sub-categories) and files (documents). The user can create a new category and delete empty categories. The categories can also be marked with “share” and “don’t share” indications. The indication for “don’t share” means of course that the content should not be shared to other users. To indicate an uninterest in a document the user just deletes a document or puts it in the uninteresting category. The user can change the model simply by dragging a file to another category. Another way to change it is to click on the document and then the document’s belonging to different categories is shown in an area above the tree as a couple of coloured bars, one bar for each category. The user can change the length of the bars and at the same time its assumed belonging. The tallest bar is the category that is said to contain the document, which means that the document is listed in the category in the tree. If the user clicks on a category, the area shows the category’s statistic and a button for marking it as sharable. The statistic is the most common words and number of documents.

In the area for recommended documents, each document is presented with its title if there is one, otherwise the address, and its size, domain, type, the category it belongs to and a rating bar indicating how much the belonging is. If the user clicks on the category the same window as for changing the user model appears. Each document has also a button for more details. If it is pressed, the user gets more information about the document in a new window. Details could be title, address, size, type, headlines, keywords (used to distinguish this document), number of links, recommended documents that are linked from this document, etc. If the user clicks on the title or address (in any of the mentioned widows), the document is downloaded and shown in a browser. Next to the category, there is an explanation button, and if the user presses it, a window is opened that explains the chosen category. If the agent has predicted a category, it should be explained how it happened, that is, if the document matched some other documents or that a user with similar interest liked it. In addition, if the user categorised the document it should be stated.

In the browser, there is the same information and buttons attached to each shown document and, in addition, an emphasising button. All previously presented buttons works as above. The user emphasises what is interesting in a document by pressing the emphasise button. When the user presses the button, the agent assumes that the user has highlighted an area of the current document to indicate an interesting part of the document and the areas content affects the learning of the document. This also means that there should be a way for the user to control the highlighted text areas associated to the document. In the window with the explanation, an area shows why the user liked the document according to the emphasised text and there is also a possibility to change the chosen texts.

4.2.2 The User Modelling

The Interface Agent uses the feedback from the user to form the user model. For the agent to be able to assign Interest Agents to different parts of the model the agent has to divide the model into distinct parts. The agent also has to be able to predict if the user likes a document, therefore the model must be organised to do that.

One could have used a clustering algorithm to form these parts based on the content of the documents. However, in the proposed system, the agent lets the user create categories with sub-categories and sort the documents into them as described in previous sec-

tion. A document has a certain belonging to each category, and the document is contained in the category with the largest belonging. Because the agent lets the user build the model in this way, the load on the computer is reduced, but the extra task loads the user. An advantage, however, is that the categories might contain documents with no similarities based on words, but there are similarities in the mind of the user which other users might benefit from when the documents are shared to them. Documents with no similarities based on words could for example be documents containing images, music, etc.

To be able to predict the users liking of a document, each category must be generalised to represent the interest of the user. The system represents an interest with a weighted term vector computed from the documents' belongings to the category. The weighted term vector is defined in *The Weighted Term Vector*, section 4.5. Shortly, it is a vector of terms, where every term has a weight of importance. A term corresponds approximately to a word in the document. Another representation one could have used is simply the group of documents in a category and to classify a new document, one could have used a nearest neighbour method, but it would require more computing at the classification time than the chosen method.

$$\mathbf{interest}(c) = \sum_{d \in c} belonging(d, c) * \mathbf{termVector}(d)$$

$$belonging(d, c) = \begin{cases} d \notin evaluatedDocuments \Rightarrow similarity(\mathbf{termVector}(d), \mathbf{interest}(c)) \\ d \in evaluatedDocuments \Rightarrow definedBelonging(d, c) \end{cases}$$

Where d is a document and c is a category containing a group of documents. A document's belonging to a category is defined either by the user or, if the document is not yet evaluated by the user, the agent. The agent computes the similarity between the document and the category to predict the assumed belonging. The similarity function is continuos ranging from zero to one, and higher value means more similarity and belonging.

A user that often uses a web browser has also often a tree of links in form of bookmarks or favourites. To fit into the user's normal software environment, the proposed system takes advantage of this fact. At start, an Interface Agent can use the user's bookmarks or favourites as input to make the initial user model. The bookmarks are usually ordered in a hierarchy of categories. However, because the user usually sorts them in an arbitrary way, it tries to arrange the documents in a more suitable way, asking the user for permission to do so. The user model can then function as the user's new tree of bookmarks instead of the old one or as a complement to the old tree.

The algorithm to sort the bookmarks is

1. Let *temporaryDocuments* be all documents in the hierarchy of categories.

2. Let *evaluatedDocuments* be an empty set.

3. Repeat while *temporaryDocuments* is not empty:

    3.1. Let *smallestDocument* be the *document* in *temporaryDocuments* and in the category with the smallest *belonging(document, category)*.

    3.2. Let *userChosenCategory* be the *category* in which *smallestDocument* is contained.

    3.3. Remove *smallestDocument* from *temporaryDocuments* and from *userChosenCategory*.

3.4. For each *category* in the hierarchy of categories, compute the *belonging(smallestDocument, category)*.

3.5. If the *category* for the largest *belonging(smallestDocument, category)* is not equal to *userChosenCategory* then:

3.5.1. If the user want to sort, ask the user to chose a category that the document should belong to and to what extent it belongs (the user might create a new category): *definedBelonging(smallestDocument, chosenCategory) = the chosen belonging*.

Otherwise chose the *category* with the largest belonging and *definedBelonging(smallestDocument, chosenCategory) = the largest belonging(smallestDocument, chosenCategory)*.

3.5.2. Let *userChosenCategory* be the *chosenCategory*.

3.6. Add the document to the *userChosenCategory* and to *evaluatedDocuments*.

4. Let *temporaryDocuments* be *evaluatedDocuments*.

5. If the user does not sort, and all categories contain the same documents as before step 3, then end the algorithm.

6. Go to step 2 if the user does not end the algorithm.

The user can let the algorithm sort by itself, by letting all documents be added to the category where it belongs mostly, see step 3.5.1. If there is no initial collection of links it just means that the user has to read and sort some documents before there will be any categorisation.

4.2.3 The Interface Agent's Working

To each category an Interface Agent assigns an Interest Agent that will use the generalisation of the category, that is, the weighted term vector **interest***(category)*, to find other with similar interest and to start getting recommendations.

When the Interface Agent gets a recommendation from an Interest Agent it computes the document's belonging and presents it for the user. The user categorises the document and the user's feedback is sent back to the Interest Agent. The feedback is used to modify the user model as described above.

When the user is browsing, and finds a nice document, the user evaluates it by adding it to a category, which means that it is entirely an explicit evaluation. The proposed system does not use implicit feedback since it is hard to know how to do a categorisation without asking the user directly. The reason for the user to not categorise a document could be anything; the user categorises only very interesting documents or the user has not for the moment time to categorise all interesting documents.

4.3 The Interest Agent

An Interest Agent is an agent that represents the user's interest in contact with other users' Interest Agents. It acts like a contact mediator and looks for other Interest Agents that represent similar areas of interest.

An Interest Agent represents the interest of a user as a weighted term vector, actually, with the same vector, **interest***(category)* that the Interface Agent computed for the user model.

An Interest Agent has three lists of models of other Interest Agents. An agent model consists of an interest model and a confidence value ranging from zero to one. An agent for which there is a high confidence in, corresponds to an agent model with a high confidence value. There is one list with models of very trusted Interest Agents, with high confidence values, and one list with fairly trusted agents, with low to medium high confidence values, and one list with not trusted agents, with zero to low confidence values. However, the very trusted list and the fairly trusted list can be seen as one list, where the agents in the very trusted list is treated in a slightly different way. The size of the lists is of course limited to hinder them from containing all other agents.

There is also a list with agents from which the Interest Agent is subscribing and there is a corresponding list with agents subscribing from the Interest Agent. An Interest Agent always subscribes from the $n_{subscribe}$ agents that has the largest confidence values.

An Interest Agent uses all these lists to decide where to recommend or route documents (that is, to help other agents to find information), and to know which agents to trust for recommendations. The Interest Agents sends the documents as Information Objects between each other. Such an object consists of the URL, the document content, and the agents that recommended or routed it.

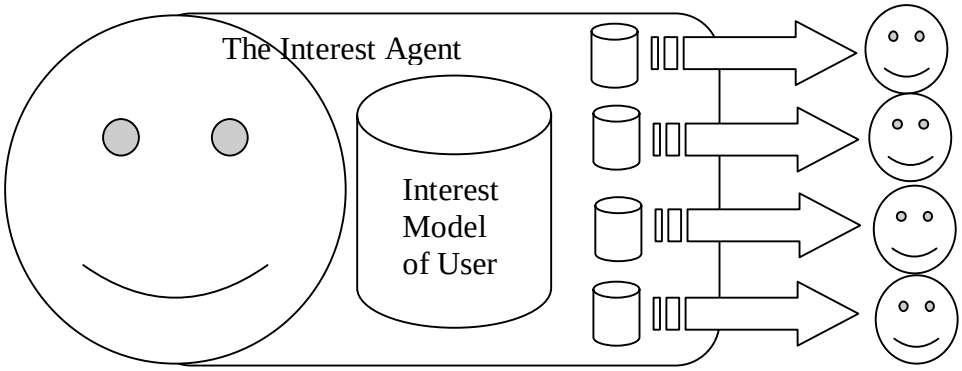

Figure 3: The Interest Agent with models of other Interest Agents.

The list of not trusted agents helps other agents to find similar agents, because an Interest Agent keeps contact with not trusted agent with different interests to be able to route to them documents not necessarily belonging to its interest. The fairly trusted list is needed to ensure the discovering of new trustworthy agents. The very trusted list is needed to present recommended documents to the user in a greater extent than otherwise possible, that is, to hinder an Interest Agent discarding good documents.

When an Interest Agent gets an Information Object from an other Interest Agent, it either based on the content of the Information Object or based on the trust for the recommending agents decides to propose the document to the Interface Agent for evaluation. If a Retrieval Agent proposes a document to the Interest Agent, the Interest Agent fetches the document and based on the content of the document, it decides if it should propose the document to the Interface Agent.

4.3.1 The Model of other Interest Agents

A model of another Interest Agent consists of a weighted term vector and a confidence value. The term vector is a representation of the agent's interest and the confidence value is a representation on how much it can be trusted.

The term vector, the agent builds from the content of the recommended documents in a

similar way as an Interface Agent builds a user model. An agent uses new recommended documents to modify the model.

$$\mathbf{model}(agent) = \sum_{document \in recommendations_{agent}} \mathbf{termVector}(document)$$

The confidence value states how much an agent trusts another agent for recommendations. An Interest Agent uses the confidence value to compute a probability for a recommendation to be sent to the Interface Agent for the user's evaluation. The confidence increases if the user liked the recommended document otherwise the confidence decreases. If the document is not presented to the user or the user does not answer, neither happens.

An intuitively rather good (but ad hoc chosen) model of the confidence value is the following function:

$cv(nop, non) = sigmoidT(nop)*(nop/(nop+non)),\ 0 <= cv(nop, non) < 1$

nop is the number of positive examples (documents liked by the user).
non is the number of negative examples (documents disliked by the user).

$sigmoidT(v) = 1/(1+exp(-a*(v-T/2)),\ a = 2*ln(B)/T$

T is a threshold, $sigmoidT(0) \sim 0$ and $sigmoidT(T) \sim 1$
B is a big number that specifies how close to 0 and 1 *sigmoidT* gets.

The curve *cv(nop, non)* is proportional to the per cent liked recommendation at $nop > T$, but at $nop < T$ the curve is suppressed. This makes the agent favour agents from which it has got more good recommendations. That is, if $T=10$ it trusts an agent with eight good and two bad recommendations more than an agent with four good and one bad recommendation. The threshold can also be said to state when to start to trust two agents the same way. For example, two agents are approximately equally trusted if one agent has sent 90 good and 10 bad recommendations, and one agent has sent 900 good and 100 bad recommendations. Another way to compute the trust is to base the confidence value on, for example, the last 25 recommendations instead of all recommendations. This would let the confidence value vary faster in response to the changed focus of the agents recommendations.

4.3.2 Routing of the Information Objects

At start, an Interest Agent advertises its existence to a close neighbourhood with a multi-cast message, this can be done with time-to-live constrained messages. When an Interest Agent gets an existence-advertising message, it recommends the advertising agent a document close to its interest, that is, it sends an Information Object.

When an Interest Agent gets an Information Object, it checks the list of recommending agents. If any recommending agent is in any of the Interest Agent's lists with agent models, it modifies the agents' models. Then, it adds the remaining recommending agents to the list that they belong to, the not trusted list or the fairly trusted list. If there are any routing agents, that is, agents that routed the Information Object, they are added to the not trusted list with the confidence value set to zero.

An Interest Agent decides to propose a document to its Interface Agent

- If the recommending agent is very trusted that is if the confidence value of an agent is larger than cv_{very}.

- If the recommended document is sufficient similar to the agent's interest, that is if: *similarity(***termVector***(document),* **interest***(agent's category))* > $similarity_t$
- If *RandomValue* < *max(cv(nop, non)* + $cv_{propose}$, *1.0)*

Where *0* <= *RandomValue* < *1,* $cv_{propose}$ <= *min(cv(nop, non)* + $cv_{propose}$, *1.0)* <= *1.*

This means that a document is recommended at least at every $1/cv_{propose}$ time (in average). For example, if $cv_{propose} = 0.2$ then at least every fifth document is going to be recommended. In the beginning it is good to choose a document to present at random, before the Interest Agents have learned what the user likes. However, it might be better to decrease the $cv_{propose}$ when an agent has a better prediction of what the user likes and thereby decrease the number of presented non-relevant documents.

If the agent does not propose the document to the user or the user did not like the document, the Interest Agent routes it, but if the user liked the document, the Interest Agent recommends it.

The difference between recommending and just routing a document is that when an Interest Agent recommends a document, it adds itself to the recommending list of the Information Object and sends it to all subscribing Interest Agents, but when an Interest Agent routes a document, it only adds itself to the routing list of the Information Object. In both cases, when recommending or routing a document, the Interest Agent sends the Information Object, firstly, to the Interest Agent with the interest model most similar to the document. Secondly, it sends the Information Object to every Interest Agent with a probability proportional to the similarity between the document and the other Interest Agents' interest models. In addition, there is some *noise*, which means that there is a certain probability that the agent sends the Information Object to a total randomly chosen known Interest Agent.

The reason for the agents to recommend or route an Information Object dependent on the model similarities and at random is to spread the information about the agents in a wider area. They tell other agents what their interests are and they send documents to, in their view, interested agents. In this way, the subscribing lists (based on the fairly and very trusted lists) connect similar Interest Agents into clusters of mutually trusting agents, but the clusters are not isolated, they also have contact with each other through the list of not trusted agents.

When the confidence for an agent is sufficiently high (larger than cv_{ask}), the Interest Agent asks this agent for ALL its recommendations. Then the Interest Agent gets a call for ALL its recommendation it sends a list with all its URLs as answer. When an Interest Agent gets all of another agent's recommendations, it downloads the content for each URL it has not already seen and it creates an Information Object of it. This means that the agent will handle each URL as an Information Object recommended by the asked agent.

All lists with agents are limited in size. This means that moving the agents between different lists or adding new agents may lead to the deletion of other agents.

When the lists are full:

- For the not trusted list, a new agent replaces another agent if its confidence value is sufficiently low and it extends the information about different interests. That is, if the agent has replaced another agent the new list has a more diverse collection of interest models in such away that it is possible to route Information Objects be-

longing to a more diverse collection of interested agents.

- For the fairly trusted list and the very trusted list, a new agent replaces another agent if it is more trusted than the previous one.
- The replaced agents are moved to the other lists if they fulfil the stated requirements.

The very trusted list consists of all agents with a confidence value larger than cv_{very}. The not trusted list contains all agents for which the confidence value is zero (no positive recommendation) and in addition, as stated above, the agents interest model must make the list more diverse. The maximum number of agent models in the not trusted list is n_{not}.

The algorithm that adds an agent (actually the agent model) to the not trusted list and makes it more diverse is:

1. If the not trusted list is not full, add the new agent model and stop
2. Otherwise: Find the most similar agent model in the not trusted list (based on the interest).
3. Compute the sum of the similarities between the similar agent model and the remaining agents on the list.
4. Compute the sum the similarities between the new agent model and the remaining agents on the list.
5. If the difference between the sums indicates that the sum for the new agent model is smaller than for the similar agent, then it should replace the similar agent model.

A more accurate way would be to compute the sum of similarities for every agent model and not only for the most similar to the new agent model, but it requires more computing.

The fairly trusted list contains the remaining agent models, but no more than $n_{fairly\&very}$ agent models minus the number of agent models in the very trusted list.

4.3.3 Problems

A problem with the proposed approach is how the Interest Agents should be able to know of previous documents contained in the other agents' interests. The described system will only model other agents from newly read documents unless the confidence of the other agents is sufficiently high then they will be asked for all of their URLs. This could be solved by letting the Interest Agents now and then recommend an old document to a randomly chosen agent from the lists. The other agents would eventually get the message and learn the sender agent's old interest. On the other hand, when an Interest Agent asks for the content of another agent, then all old documents are once more routed in the system. The problem might be less important when it is most interesting to find other active users. The proposed system is dependent on active users that find, categorise and thereby recommend new documents.

A much more important problem is how to limit the flow of Information Objects. The described system stops this in two ways:

- An agent does not send an Information Object to an already encountered agent, that is, the agent is in the recommending or routing list of the Information Object.

- Every Information Object has a time to live called t_{steps} that limits the number of allowed recommendations and routings of it. This is probably the most effective way to limit the flow.

There are of course more possible ways to limit the traffic. An agent could for example be hindered to send a recommendation of a certain URL more than once (unless the content has changed). This means that there should be a list for each modelled Interest Agent with all of its recommended URLs and a way to check if the content of any URL has changed since last time. This check could be done, for example, by comparing the new term vector with the old one.

4.4 The Retrieval Agent

The Retrieval Agent is the agent that searches for new documents matching the user model on the WWW. The Retrieval Agent does this through monitoring the Interest Agents opinions and learning the Interest Agent's profile. One could say that the agent builds an agent model of the Interest Agent and instead of learning the profile based on the documents, it uses the context of the links in a document. The context is defined in a similar way as in figure 4.

Each feature in figure 4 is the context of a link. A link is either an area or an anchor. The agent builds a weighted term vector of the link context to learn what links are interesting to follow. Actually, each slot in the link context, that is the title text, a heading text, etc., is a weighted term vector in the model of the Interest Agent's profile and together they represent the link. If a Retrieval Agent finds a very interesting link, it sends it to the Interest Agent for evaluation, and if it is a "bad" link the Retrieval Agent modifies the vectors and decreases the influence of the bad link's words. However, if it is a good link it increases the influence.

For each slot a:

$$\mathbf{model}_{a,\,t+1} = \mathbf{model}_{a,t} + fb*delta*(\mathbf{termVector}(slot_a) - \mathbf{model}_{a,t})$$

Where *delta* is a scalar defining the learning rate, $0 < delta < 1$, and *fb* is the feedback, $fb > 0$ for positive feedback and $fb < 0$ for negative feedback.

A Retrieval Agent has a list with interesting documents from which it starts its search. The agent takes the most interesting document on the list and parses it for interesting links. Links that is sufficiently interesting (for all slots a in link context, $similarity(\mathbf{model}_{a,b}\ \mathbf{termVector}(slot_a)) > similarity_a$), the agent sends to the Interest Agent, and if that agent says the links are good, the Retrieval Agent adds them to the list and modifies its models for all slots as described above.

At the start, the Retrieval Agent gets the weighted term vector of the Interest Agent's interest that it uses as the initial weighted term vector of the context of the links. If there is no interest at start, the agent has to wait to get some pages from which it can start its search.

If the list is empty and there is a weighted term vector, the agent uses the vector to query a search engine. It adds the returned document (the result of the query) to the list and it starts to parse the returned document for interesting links. The Interest Agent can also send documents that should be added to the list and it can also tell the Retrieval Agent to change its weighted term vector of the context to undo earlier feedback.

Web Document

<HTML>
<TITLE> Page about Agents </TITLE>
<H1> Interface Agents: A Heading </H1>
A bit of text
<A HREF = “http://www.abdn.ac.uk/”>
A link somewhere
</A>
<H2> Another Heading </H2>
Text around link
<A HREF = “http://www.csd.abdn.ac.uk/”>
This is a link about LAW
</A>
even more text
</HTML>

Features

Link 1

Title Text = {page about agents}
Heading Text = {interface agents heading}
Surrounding text = {bit text}
Link Text = {link somewhere}

Link 2

Title Text = {page about agents}
Heading Text = {another heading}
Surrounding Text = {text around link even more}
Link Text = {link about law}

Figure 4: Extracting Features from an HTML Document [Edwards, Bayer, Green & Payne 1996].

The Retrieval Agent has also a list of already encountered documents to avoid recommend old documents or traverse them. To this list, the agent adds those document that it gets from the Interest Agents recommended by other Interest Agents. This means that it trusts the other Interest Agent to check these documents.

For the Retrieval Agent to not disturb the WWW-community it should follow the agreed on rules for WWW robots [Koster 1993].

4.5 The Weighted Term Vector

The weighted term vector is used for:

- Extracting information from a document
- Generalising bunches of documents
- Measuring the similarity between documents and the generalised bunches.

Term vectors are easy to implement and the technique has been used in many applications. Together with the cosine similarity, it has been proven to work quite well

[Kilander, Fåhræus & Palme 1997].

A weighted term vector **w** is defined as

$$\mathbf{w} = \{w_i, w_{i+1}, \ldots, w_n\}$$

Where w_t is the weight for term t and a term is the stem of a word. It is good to keep only the stem of a word because the stem often contains the meaning of the word and it has the same meaning regardless of the inflection. For example, the stem for both *index* and *indexing* is *index*. In addition, the size of the term vector is reduced by removing words that do not add any information to the meaning of a text, for example, words like *but*, *as*, *and*, etc. Such words are often called stopwords.

4.5.1 The Creation of a Weighted Term Vector

The weighted term vector is created either from a plain text document or from an HTML document. The straightforward way to create a weighted term vector is to count the occurrence of each term in a document (the Term Frequency, which is often called tf), but to extract the most important information from a document, it might be better to take advantage of the terms' context too. The context might indicate the importance of the term, for example, a term in the title of a document might be more important than a term in the body text. Therefore, in the proposed system, the context effects the term weights in a vector when it is created. Each occurrence of a term is multiplied with a scalar and then added together to form the weight of the term. For plain text, where it is hard to distinguish the context, the scalar for each term is 1. In HTML documents, where the term context is indicated with tags, it is easy to vary the scalars and in the proposed system, the scalars are defined as in figure 5. In addition, the scalar is 4 for all words in attribute values of the tag META and for the words in the attribute value ALT of a tag. ALT gives an alternative text shown instead of, for example, an image. The scalars are all *ad hoc* and there is no special reason for the chosen values other than that the weights correspond to the supposed importance of the tags (or attribute values).

TAG	**A**	**I**	**B**	**H4**	**H3**	**H2**	**H1**	**TITLE**
Scalar	2	2	2	2	3	6	12	16

Figure 5: HTML tags with corresponding scalar. For other tags, the scalar is 1.

The algorithm for creating a term vector for a document, **termVector***(document)*:

1. Create a stopword table.
2. Create an empty term vector **w** = {}.
3. *word* is next word in document.
4. If *word* is in the stopword table, go to step 3.
5. Term *t* is the stem of *word*.
6. If w_t is already defined, add the scalar of *t* to w_t, otherwise let w_t be the scalar of *t* and add w_t to **w**.
7. Repeat steps 3-6 until there are no more words in document.
8. Return term vector.

The described algorithm is similar to the algorithm Salton has described [Gloor 1997]. It differs in how the weights are computed. He computes the weight for term *a* as

$w_a = termFrequency_a * log(numOfDocuments/CollectionFrequency_a)$

Where $termFrequency_a$ is the number of occurrence of the term in a document and $CollectionFrequency_a$ is the number of documents in which the term occurs in the collection of documents. The variable *numOfDocuments* is of course the total number of documents in the collection. This is often called the tfidf measure, an acronym for Term Frequency Inverse Document Frequency. In the proposed approach the weights corresponds more to the term frequency if the scalar is 1, that is, in the proposed system

$w_a = K*termFrequency_a$

Where K is the average sum of the scalars of the term a. For a comparison between different ways to compute weights, see *PEFNA The Private Filtering News Agent* [Kilander, Fåhræus & Palme 1997].

4.5.2 The Similarity Measure

The similarity measure the proposed system uses is Salton's cosine measure [Kilander, Fåhræus & Palme 1997]:

$similarity(\mathbf{w}, \mathbf{v}) = \mathbf{w}*\mathbf{v}/(|\mathbf{w}|*|\mathbf{v}|)$

Where **w** and **v** is weighted term vectors. If a term is absent in one of the vectors, the weight of it is zero.

This measure is the cosine of the angle between the two term vectors. This means that what really matters in a vector is the relative size between the weights in a vector and not how large a weight for a word in one vector is compared with same word in the other vector. If one vector is multiplied with a constant, it does not change the similarity between the two vectors.

Instead of the cosine similarity, one can computed the number of common terms or any other imaginable way of comparing vectors of numbers.

4.6 Protection of the User-Integrity

The most important problem in a distributed system where users share their profiles is how to protect the user's integrity. This problem is not in the focus of this thesis but there are a couple of different techniques to hinder unauthorised users to read other users' messages and to let a user be anonymous. One can use cryptographic to code and decode messages and one can distribute the encoding of the agent addresses so that no agent has the full address to any other and thereby make the agents (and the users) anonymous. To hinder somebody to watch the communication and thereby, get some information of the involved users one can send messages via some other agents to spread the communication in the net. There are more on this subject in Leonard Foner's paper *A Security Architecture for Multi-Agent Matchmaking* [Foner 1996]. There are also some simpler methods one can use, such as separating the user model in two parts, one public and one private part [Höök 1997]. This is the way it works in the proposed system then the user might sign a category with "share" or "don't share", but this loads the user more than if the user could be anonymous.

4.7 Evaluation of design criterions

The proposed design fills, more or less, most of the criterions in section 3.2. The system should:

- *Adapt to the user*. The proposed design adapts to the user. It builds the user model based on explicit feedback (but not based on implicit feedback) and it uses the model to find interesting document, which is done by the Retrieval and Interest Agents. It also uses the user model to present the interesting documents to the user, which is done by the Interface Agent. There is also a way to explicitly change the system's working through the preferences button.
- *Act as unnoticed by the user as possible*. The system presents interesting document in a special window. The user decides if the document should be shown. However, to have a window showing all recommendations could be disturbing. A small notice of new recommended documents might be better. The user interface is incorporated with the web-browser to look familiar and it is easy to make it to work at non-working time by letting an agent work as a screen saver telling the other agents to start working.
- *Let the user modify its internal state*. The tree-structure of the user model makes it easy to modify it with a graphical interface looking like an ordinary file manager.
- *Let the user decide when to read recommended documents*. The user clicks on a recommended document when the user wants to read it.
- *Protect the user's integrity*. The current system does not protect the user in any sophisticated way. The user can only indicate if a category is allowed or disallowed to be shared to other users.
- *Use back channels to signal that it is working*. The Interface Agent has an area where it shows what it is doing all the time, such as connecting to a site or proposing a document to other agents.

5 WebCondor — The Implementation

To test the designed system, a version of it called WebCondor was implemented, an acronym for Web-based Collaborating Document Recommender. The goal for the implemented system was to get a working prototype to demonstrate the power of collaboration between agents.

Because of lack of time and because the focus of the thesis has been on the collaborative part, only the Interface Agent and the Interest Agent were implemented.

The prototype implementation was done in Java[5] and JAFMAS version 0.1 [Chauhan & Baker 1997] was used as multi-agent framework. The reason to choose JAFMAS was that it uses multi-cast messages to advertise the agent's presence instead of using a centralised agent name server and thereby the system gets more distributed. Another advantage is the provided design process for developing agent applications focused on the conversation part. In addition, there are only 16 classes in the JAFMAS package, which means that it is easy to grasp and understand the functionality of them.

In the implemented prototype, all agents are running on the local machine and the Interest Agents communicate with RMI [Sun 1996]. RMI is an acronym for Remote Method Invocation and it is used for calling methods in processes running on remote hosts. In the proposed design in section 4.1, it was stated that the communication should be in KQML, but it is not used in the implementation, although, the agents use the form of speech act communication supported by JAFMAS. One could easily make it work like KQML, but KQML seems more suited for expert systems and the impression was that one had to "squeeze" the proposed system into KQML. There was not sufficient time to figure out how this should be done.

There are some differences compared to the proposed system design and some important implementation details that the following sections describe.

5.1 The Weighted Term Vector

The code for the stemming algorithm is an implementation of the Porter algorithm [Porter 1980; Frakes & Baeza-Yates 1992]. It is written in C and compiled as a library that is loaded by the Java application. The list of stopwords is the combination of the lists provided by [Frakes & Baeza-Yates 1992] and SMART [Salton & Buckley 1994]. SMART is an old, well-known, information retrieval package. The implemented classes for the weighted term vector does only work for English texts. Other languages must have their unique implementations.

To limit the computing, it is good to limit the size of the vectors in some way, therefore, the system is able to create vectors with a certain size and just keep the terms with the largest weights.

5.2 The Interface Agent

The Interface Agent consists of a Java application and two Java applets in a Netscape Communicator browser. The application communicate with the browser in two ways, either the application works as a proxy server [Janes] or the applets communicate with the application directly via UDP sockets. One starts the application in a dos-window in

[5] **Java** is a trademark of Sun Microsystems, Inc.

which it prints what it is doing.

5.2.1 The User Model

The agent builds the user model by letting the user sort the documents into user defined categories. The graphical interface (the graphical user interface is described in Appendix A) lists the categories and the user is able to add new categories but not remove a category or change the content of a category. Neither the tree structure for handling the user model nor the algorithm to build the initial model from the user's bookmarks is implemented. Each interest is handled by the Interest Agents and not separately by the Interface Agent as in the proposed design.

To limit the amount of used memory and computing, the Interface Agent limits the size of the term vectors. The maximum number of words in the term vector of a document when the Interface Agent gets it is $n_{document}$. For the Interface Agent it is

$$\mathbf{termVector}_{InterfaceAgent}(document) = \mathbf{trim}(\mathbf{termVector}(document),\ n_{document})$$

$\mathbf{trim}(...)$ is a function that returns a term vector with the $n_{document}$ terms with the largest weights.

To simplify the formation of the interest model of the user, the interest model for each document and category is defined with a *belonging(document, category) = 1*, which means that (compared to section 4.2.2)

$$\mathbf{interest}(c) = \sum_{d \in c} \mathbf{termVector}_{InterfaceAgent}(d)$$

Where d is a document and c is a category containing a group of documents.

5.3 The Interest Agent

Remember that an Interest Agent manages all communication and collaboration between the Interface Agents. It has three lists with models of other Interest Agents and a list with subscribers and a list with subscriptions.

When the Interest Agent is created, it advertises its presence to the other agents with a multi-cast message. The message reaches all other agents and is not limited by the time-to-live stamp described in the design in section 4.3.3.

The model of the interest of the other Interest Agents was defined in section 4.3.1. The previous definition was

$$\mathbf{model}(agent) = \sum_{document \in recommendations_{agent}} \mathbf{termVector}(document)$$

To reduce the computations, the vector of the models of the other Interest Agents contains max n_{model} words with the largest weights. This modifies the model of the agent to

$$\mathbf{model}(agent,\ t+1) = \mathbf{trim}(\mathbf{model}(agent,\ t) + \mathbf{termVector}(document),\ n_{model})$$

Where t indicates the t^{th} computed model and $\mathbf{trim}(...)$ is a function returning a term vector with the n_{model} terms with the largest weights.

5.4 The communication between the Interface Agent and Interest Agents

An Interest Agent can:

- Subscribe from another Interest Agent, that is, it is going to get all recommendations

from the other Interest Agent.

- Unsubscribe from another Interest Agent, that is, it should stop sending recommendations.
- Ask for the content of another Interest Agent, which means that it gets all the URLs that the other agent has, see next item.
- Tell an asking Interest Agent its content as a list of all of its URLs.
- Propose a document to another Interest Agent, which means that it sends an Information Object as a recommendation to the other agent.
- Propose a document to the Interface Agent.

The Interface Agent can communicate to the Interest Agent in some different ways. It can send:

- Tell document, which means that the document belongs to this agent and it should recommend it to other Interest Agents.
- Accept document, which means that the proposed document belongs to this agent and that the document should be recommended to other Interest Agents.
- Reject document, which means that the proposed document does not belong to this agent.

To change the interest of an Interest Agent without routing or recommending a document, an Interface Agent can send to an Interest Agent:

- Add document, which means that this document should be added to the interest, that is, the document is added to the URL list and to the interest term vector.
- Remove document, which means that this document should be removed from the interest, that is, the document is removed from the URL list and subtracted from the interest term vector.

6 Results from Simulated User Tests

6.1 The Simulation Set Up

To test the system, four simulations of six users with five interests each were performed. The users were divided into two groups with similar interests (called A-users respectively B-users), that is, there are three users with the same interests in each group. The interests were taken from [Yahoo! 1998]. The agents' interests are presented in Appendix C.

The simulations were run on two PCs, a Pentium 200 MHz MMX with 32 MB RAM and a Pentium Pro 200 MHz with 64 MB RAM. There were three WebCondor applications running on each computer and the retrieved documents were all downloaded locally.

In the simulations, each Interface Agent has a list of all documents belonging to its interests and a list of documents that the simulated user finds during the session. No user finds the same documents. In average, a user finds one document per every 1.5 minute and the maximum rate of handled recommendations is one per 5 seconds. The probability distribution is constant.

There are many variables in the system and they are all mainly chosen ad hoc:

- The smallest probability for an Interest Agent to propose a recommended document to its Interface Agent:

  $cv_{propose} = 0.2$

- The similarity threshold, between a document and an interest, for a document to be proposed to an Interface Agent:

  $similarity_t = 0.2$

- When the confidence value of an agent is larger than this threshold, it is very trusted:

  $cv_{very} = 0.8$

- When the confidence value of an agent is larger than this threshold, the Interest Agent asks the agent for its content:

  $cv_{ask} = 0.5$

- The maximum number of words in the term vector of an agent model:

  $n_{model} = 200$

- The maximum number of words in the term vector of a document when the Interface Agent gets it:

  $n_{document} = 128$

- The maximum number of agent models in the not trusted list:

  $n_{not} = 3$

- The maximum number of agent models in the fairly trusted list and the very trusted list together:

  $n_{fairly\&very} = 2$

- An Interest Agent subscribes from the maximum number of agents:

  $n_{subscribe} = 2$

- The number of times an Information Object is allowed to be routed or recommended (time-to-live):

  $t_{steps} = 4$

- The probability for an Interest Agent to send an Information Object to an agent regardless of the confidence value and content of the information:

  $noise = 0.02$

- The suppressing function:

  $sigmoidT(v) = 1/(1+exp(-a(v-T/2)),\ a = 2ln(B)/T$

  $T = 4$ and $B=999$, which means that $sigmoidT(0) = 0.001$ and $sigmoidT(4) = 0.999$

From section 4.3.2, an Interest Agent proposes a document

- If the recommending agent is very trusted, that is, if the confidence value of an agent is larger than cv_{very}.
- If the recommended document is sufficient similar to the agent's interest, that is if: $similarity($**termVector***(document),* **interest***(agent's category))* $> similarity_t$
- If $RandomValue < max(cv(nop, non) + cv_{propose}, 1.0)$

Where $0 <= RandomValue < 1$, $cv_{propose} <= min(cv(nop, non) + cv_{propose}, 1.0) <= 1$ and $cv(nop, non)$ is defined as

$cv(nop, non) = sigmoidT(nop)*(nop/(nop+non))$

nop is the number of positive examples (documents liked by the user).

non is the number of negative examples (documents disliked by the user).

This means that a document is recommended at least at every $1/cv_{propose}$ time (in average). For example, if $cv_{propose} = 0.2$ then at least every fifth document is going to be recommended. In the current implementation, the $cv_{propose}$ is not decreased when an Interest Agent learns to make better prediction.

6.2 Measuring the Performance of the System

A common way to measure the performance of an information retrieval system is to measure the *precision* and the *recall* [van Rijsbergen 1979].

To measure the *precision* for an Interface Agent, the total number of proposed documents belonging to any of the user's interests divided by the total number of all proposed documents was computed. The *recall*, was measured by computing the total number of proposed documents belonging to any of the user's interests divided by the total number of all documents belonging to the interests. The goal is to have high values for both precision and recall. For example, the maximum value for recall is one, which is simple to get. One only has to retrieve all documents, but that would give a low value for the precision.

One could have measured the precision and recall for each interest and not for all interests together, but in reality, the Interest Agents help each other. A document belonging to another interest than the proposing agent has also been retrieved for the benefit of the

user. This should therefore also be the case in the simulations.

Another way to measure the performance is to see how the Interest Agents are connected to each other. For which other agents do they trust? The wanted result is of course that they have maximum trust (confidence value is one) for every agent with the same interest. In the simulation, each agent can at most trust two other agents. To measure the connectivity for an Interface Agent to the others, a value between zero and one was computed by the function

$$connectivity(agent) = \sum_{i \in interestAgents_{agent}} sumCV(i) / \sum_{j \in interestAgents_{agent}} maxSumCV(j)$$

Where in the simulations

- *sumCV(i*) is the sum of all confidence values for other agents the Interest Agent *i* of Interface Agent *agent* has
- *maxSumCV(j) = 2* for all Interest Agent *j* of every Interface Agent
- the number of interests for every user is five (the second sum is equal to 2+2+2+2+2=10).

A high value for connectivity means that the agent is going to get more documents from trusted agents than an agent with less value for the connectivity.

It might be hard to know if the performance of the system is good or bad without anything to compare with, therefore two pairs of different simulations were run. The first pair is a comparison between the proposed system's way of making proposals (the ordinary way) and a pure random way of doing the same. One test has Interest Agents that make proposals to the Interface Agent as described in previous sections (hereafter called the ordinary-test) and one test has Interest Agents that make proposals at a 50 per cent probability (hereafter called the 50%-test). The second pair is a comparison of the affect of the cognitive and the social filtering. In the first test, only the cognitive filtering technique is used to propose documents and in the second test, only the social filtering technique is used.

6.3 The Analysis of the Performance Result

The first pair of simulations was running for three hours. Where the one with the ordinary way to make proposals had 161 documents in the system, with 71 documents belonging to the A-users and 90 to the B-users. The one with 50 % probability to make a proposal had 155 documents in the system. There were 65 documents belonging to the A-users and 90 to the B-users. The total number of possible available documents was 162. There were 72 documents for the A-users and 90 documents for the B-users, see Appendix C. This means that not all documents were "found" by the simulated users during the two sessions, but the numbers are comparable between the two tests although the A-users have fewer documents. This means that it is harder for the A-users to find relevant documents than for the B-users and therefore, the A-users have smaller values in the results.

In figure 6 is the result of the first pair of simulations presented. It seems like if the precision is some what better for the random based proposal than for the proposed system, but there is only a small difference for the over all performance (Average for A & B). The reason for the good values of the 50%-test is probably that there is a probability of about 50 per cent that a completely random chosen document from the collection belongs to the proposing agent. The recall and connectivity for the proposed system seems

	Precision Ordinary	Precision 50 % prob.	Recall Ordinary	Recall 50 % prob.	Connectivity Ordinary	Connectivity 50 % prob.
User A1	0.188	0.340	0.184	0.395	0.118	0.499
User A2	0.365	0.346	0.660	0.220	0.700	0.115
User A3	0.329	0.300	0.609	0.255	0.700	0.300
Average for A	0.294	0.329	0.484	0.290	0.506	0.305
User B1	0.515	0.429	0.576	0.356	0.653	0.553
User B2	0.387	0.500	0.492	0.322	0.800	0.453
User B3	0.542	0.569	0.516	0.532	0.697	0.700
Average for B	0.481	0.499	0.528	0.403	0.717	0.589
Average for A&B	0.388	0.414	0.506	0.347	0.611	0.437

Figure 6: The result of the first pair of simulations.

however to be much better for the ordinary-test. The reason for this is probably that an Interest Agent, that uses the content of a document to decide if it should propose a document, helps its user's other agents with similar interests. If two Interest Agents of the same user have similar but not the same interests, then a document received by one of the agents might get proposed and then delivered to the other agent, based on the document content.

The second pair of simulations was also running for three hours. The simulation with only the cognitive part of the decision to propose a document had 162 documents in the system. There were 72 for the A-users and 90 for the B-users. The simulation with the social part of the decision had 152 documents in the system, with 62 documents for the A-users and 90 for the B-users. The difference for the simulations might be somewhat too large, but from earlier simulations with even fewer documents for the A-users, the values for the B-users seemed to counteract the lower values for the A-users.

In figure 7, the result of the second pair of simulations is presented. As one can see the precision is rather good, especially for the cognitively based decision, but the other values are not good at all. It does not seem to be a good idea to use only one and not the other way of deciding if a document should be proposed. The social aspect of the decision might work better when there are more documents found by each user and the agents receive more documents recommended from each user. The reason for the social based system to work at all, is because there is a threshold added to the confidence value. The threshold makes the system work like the 50%-test but with only 20 per cent probability to propose a document if it is recommended by an agent with confidence value of zero. When the two methods are used together, the probability threshold helps the system to start to propose some documents that form the initial interest model, which makes the cognitively based part perform better. Thereby does the agent find more documents of interest, which increases its trust for the recommending agents and thereby, the agent gets more recommendations of interesting documents, and so on.

Another aspect of the system is that the routing of documents is also based on the con-

	Precision Cognitive	Precision Social	Recall Cognitive	Recall Social	Connectivity Cognitive	Connectivity Social
User A1	0	0	0	0	0	0
User A2	0.857	0.475	0.128	0.514	0.106	0.497
User A3	0.800	0.463	0.170	0.514	0.200	0.403
Average for A	0.552	0.312	0.185	0.342	0.102	0.300
User B1	1.000	0.875	0.085	0.119	0.106	0.153
User B2	0.714	0.051	0.085	0.034	0.103	0.050
User B3	1.000	0.800	0.306	0.129	0.356	0.112
Average for B	0.905	0.575	0.159	0.094	0.188	0.090
Average for A&B	0.729	0.444	0.172	0.218	0.145	0.095

Figure 7: The result of the second pair of simulations.

tent of the documents. This might also add some positive effect on the performance of the system. The routing might mean more for the performance of the system than the different ways of deciding if a document should be proposed. The significance of the content-based routing should also be investigated in some way.

Some shortcomings of the simulations are

- There are a limited number of runs, which makes it hard to make any strong conclusions. Ideally one should performed many more runs and analyse the average of all results
- The multi-cast messages reach all agents at start. It makes it theoretically possible for the agents to find all other agents with similar interests at once. However, this does not happen. The reason for this is probably that even when one tries to synchronise the start of all agents, they do not start at the same time. Before an agent is started, it gets a document to send as a recommendation in answer to the multi-cast message. Not all agents get the document at the same time. They can receive a multi-cast message before they get the first document, which means that they do not have any document to send in answer. To really solve this problem, one could let the Interest Agents answer only, for example, every third multi-cast message
- There are no overlapping interests. All interests are distinct and no interest for any user shares any documents with any other interest. This means that if a recommendation from an agent is interesting for a user, all recommendations from that agent are interesting and the trust can only increase for the agent. In reality, one would get uninteresting recommendations too. In addition, no A-user shares any interest of any B-user, which means that a user's Interest Agents which have much in common help each other to find interesting documents. If an Interest Agent gets a recommendation from any Interest Agent in the same group of users (A or B), this recommendation does always belong to another Interest Agent of its user.
- The simulations only test some different ways to propose documents to the user. There many more parameters and algorithms that can be tested, such as number of

agents (the scalability), how to route Information Objects, etc.

- The two PCs are not entirely equal and therefore executes the agents at different speeds. The simulations should be performed on a single computer, in a single environment, to give better control of the variables.
- The simulations do not show the benefits of the social filtering, that is, if the users get documents of a kind that the content-based analyse can not recommend. To do this, one has to somehow measure the number of proposed documents with a content of a different sort than text, for example images.
- The WWW is much larger than 162 documents and there are many more possible interests! The simulations indicate only that it works for small groups of users, but then, it would be much easier to have a centralised system.

7 The Major Problems

The designing, implementation and testing of the proposed system brought up seven major areas of problems for such systems:

- *How does the system represent the information in documents?* The proposed system represents the information of a document in a weighted term vector, which is one of the most commonly used representations. The weights, however, could be computed in different ways, and instead of terms, one could use the ratings from the other users in some way. Another related problem is how to measure the similarity between the documents. One could for example count the number of words common in both documents or as in this thesis, use the cosine similarity. Yet another question is how one handles synonyms or inflections. Synonyms and inflections could be handled by some sort of dictionary, or a stemming algorithm, as in this thesis, that takes care of some inflections.
- *How does the system model interest of the users and of other agents?* The proposed system models interests as the computed sum of the weighted term vectors of the documents belonging to the interest. Instead, one could have used a neural network trained with the term vectors of the interest's documents. One could also have trained a naive bayesian network, or just represented the models by the instances of the documents and then used a nearest neighbour algorithm. In *Syskill & Webert: Identifying interesting web sites* they present and compare different ways to learn user models [Pazzani, Muramatsu & Billsus 1996]. Another related problem is how to model the trust in other agents. The proposed system bases it on the number of positive examples and suppresses the trust for low numbers. For large number of positive examples, the trust is proportional to the number of positive examples divided by the total number of examples. This could have been done in a different way by, for example, a time constrained trust dependent on the last 100 examples. A good paper to read if one wants to study trust is *Formalising Trust as a Computational Concept* [Marsh 1994].
- *How do the Interest Agents find other agents with similar interests?* One can imagine different ways to do this. In the proposed system the agents sends information to other agents and expects that the other agents will send back similar information and in addition, information about other agents. Another way could be to send a document and ask another agent to evaluate it and then to route it to some other known agents. The answers should then come back to the original asking agent. Yet another way to compare the interests is to do it directly between the agents as in Yenta [Foner 1997]. In Yenta, each agent has knowledge and a model of some other agents, which the agent uses to find more agents with similar interests.
- *How does the user communicate its interests to the system?* This is the problem of user modelling. How does the system know what the user's interests are? There are different ways, as usual, to do this. The obvious way is to ask the user, which is a very common way to do it. The user could for example input the key words for a category, which helps the agent to search. Another way is to let the user select its interest from a predefined list of documents. In the proposed system design, the agent uses the user's bookmarks to build the initial user model and then the user sorts interesting documents. To let the user tell what is interesting is called explicit feedback, but there is a possibility for implicit feedback too. Implicit feedback means that the system records the user's actions and based on the actions, guesses

what the user thinks. This can be dangerous, as the assumptions could be completely wrong. The Letizia [Lieberman 1995] uses implicit feedback to learn what the user likes. For example, if the user follows many links from a page it is assumed that the user likes the page containing the links. The proposed system uses implicit feedback to decide from which agents to subscribe. The Interest Agent assumes that it should subscribe from the agents for which it has the highest confidence values. Which sort of feedback is best? One might say neither of them. Explicit feedback gives a more precise information about the user, but it loads the user with more decisions. The implicit feedback lets the user ignore it but it gives often not very precise information about the user. The best might be to use both ways to gain the advantages from both of them. This has been shown in *Learning Personal Preferences On Online Newspaper Articles From User Behaviours* [Sakagami & Kamba 1997].

- *How is the integrity of the users protected?* The most important problem is probably to make a system as the proposed one acceptable to the users. The users might not want to share their interests in any great extent [Höök 1997], therefore one could argue that the system must be so good that the user benefits more from it than it costs to use it. In the proposed system, the user can mark an Interest Agent with ”don’t share”, which means that it does not communicate with other Interest Agents, and thereby, it does not share its interest to other users. More on this in section 4.6.
- *How does the agents find each other when they are new in the system?* In the proposed system, the agents multi-cast and advertise their presence, but this could be handled by asking a broker that has a list of some known agents or let the user tell the agent of some known agents. The shortcoming of this technique is that it is more centralised than the proposed system.
- *How is the system tested?* A problem that needs to be well thought through is how the system is tested. The system should be scalable which means there should be many users and to test it in reality could be very hard (and very expensive), therefore the system must be simulated. Hence one has to know how to simulate a user; what interests the user has, what documents the user finds, how recommendations are handled, what probability distribution should be used, etc. One also has to choose parameters to test. In the simulations of the implemented system, the performance was only compared to the random choice of proposals, the social filtering and the cognitive filtering. These tests were rather limited.

8 Conclusions and Future Work

A multi-agent system for information filtering with collaborative agents is quite a complex system. It is hard to know how things are going to work and it might be very hard to explain how the system is working to others, especially then there is an emergent behaviour such as information routing involved. It is not possible to see what is really going to happen if one look at only one agent. One must always have the whole system in consideration.

This is achieved in this report:

- An analysis of the problem area and a list of design criterions for a system that recommends web-documents are presented. The need for a user-adaptive system is clear and it must work in a way the users can accept.
- A system that meets most of the criterions is presented. It adapts to the users, protects the integrity of the users (although in a quite simple way) and the users can control the system's user models.
- An implemented prototype called WebCondor is described and the results of the testing are reported. The implementation consists of the collaborative parts and the tests show that the system might work for a small group of users, although the result might not be completely convincing. In the future, it might be good to simulate the system in a special simulation environment on a single computer and not as distributed concurrently running entities, in order to reduce the number of uncontrolled variables.
- A list of major problems for the proposed sort of systems is presented. The main problems are to choose methods to use in the system, to choose the way the users communicate their interests to the system and to protect the integrity of the users in the system.
- The conclusion is reached that the proposed system is certainly worth future research and hopefully, this report is a good starting place for such a research.

The described prototype has the most important parts implemented, but to gain all advantages, the Retrieval Agent should be implemented too. In addition, the user interface should also be fully implemented to make the system easy to use.

The proposed system is of ad hoc nature. The chosen values for parameters and the chosen methods are all chosen based on what was thought reasonable. What is needed foremost are simulations with varying values for the parameters and simulations where the system uses different techniques and algorithms for the routing, models, term vector, similarity measure, etc. One could

- Use another similarity function such as counting the common words in the two vectors
- Represent documents in vectors of weighted term classes. For example, if the term *donkey* and the term *cat* are in a document and the terms belong to the class *animal*, both terms give weight to the element *animal* in the vector of the document
- Let the Interest Agents use different techniques in the same system. The system might work better with a more diverse structure
- Test the scalability of the system by increasing the number of agents. Scalability is

supposed to be an important advantage for a multi-agent system compared to centralised systems.

Presumably, to make the users to start to use the system, it must securely handle the integrity of the users, which the proposed system handles poorly. However, the advantages of using the system might outweigh the disadvantages. This could be investigated, but it probably requires tests with real users.

References

Balabanovic, M & Shoham, Y. 1995. *Learning Information Retrieval Agents: Experiments with Automated Web Browsing*. AAAI-95 Spring Symposium on Information Gathering from Heterogenous, Distributed Environments. [http://robotics.stanford.edu/people/marko/papers/lira.ps]

Balabanovic, M. 1996. *An Adaptive Web Page Recommendation Service*. Stanford University Digital Libraries Project Working Paper SIDL-WP-1996-0041. [Also revised version in: Proceedings of the First International Conference on Autonomous Agents (Agents'97). ACM Press. pp 378-385] [http://www-diglib.stanford.edu/cgi-bin/WP/get/SIDL-WP-1996-0041]

Balabanovic, M & Shoham, Y. 1997. *Fab: Content-based, Collaborative Recommendation*. In: Communication of the ACM, Vol 40, No 3. [http://www.acm.org/pubs/articles/journals/cacm/1997-40-3/p66-balabanovic/p66-balabanovic.pdf]

Chen, L. *WebMate: A Personal Agent for World-Wide Web Browsing and Searching*. [Web page with software]. [http://www.cs.cmu.edu/~softagents/webmate/]. [Accessed 05/01/98]

Chauhan, D & Baker, B. 1997. *JAFMAS, A Java-based Agent Framework for MultiAgent Systems*. Department of Electrical and Computer Engineering and Computer Science, University of Cincinnati, Ohio. [Web page with Java source code and documentation]. [http://www.ececs.uc.edu/~abaker/JAFMAS]. [Accessed 05/01/98]

D'Aloisi, D & Giannini, V. 1995. *The Info Agent: an Interface for Supporting Users in Intelligent Retrieval*. In: Proceedings of "ERCIM Workshop on Towards User Interfaces for All: Current Efforts and Future Trends". Heraklion. pp 145-158. [http://www.cilea.it/GARR-NIR/nir-it-95/atti/giannini/giannini/giannini-nir-95.html]

Edwards, P & Bayer, D & Green, C L & Payne, T R. 1996. *Experience with Learning Agents which Manage Internet-Based Information*. AAAI 1996 Stanford Spring Symposium on Machine Learning in Information Access. SS-96-05. AAAI Press. pp 31-40 [ftp://ftp.csd.abdn.ac.uk/pub/pedwards/PUBS/mlia96.ps.Z]

Ejderberg, D. 1997. *Creating a User Agent in a Distributed Information Retrieval Environment*. Department of Computer and Systems Sciences, Royal Institute of Technology. Sweden. M. Sc. Thesis

Firefly Network, Inc. 1997. *Collaborative Filtering Technology: An Overview*. [Web page] [http://www.firefly.net/company/CollaborativeFiltering.fly]. [Accessed 27/08/97]

Fischer, G & Christoph, T G. 1997. *Using Agents to Personalize the Web*. In: Proceedings of the International Conference on Intelligent User Interfaces (IUI'97). ACM Press. pp 53-60. [http://www.gmd.de/fit/publications/ebk-publications/IUI97.pdf]

Foner, L N. 1996. *A Security Architecture for Multi-Agent Matchmakin*g. The Second International Conference on Multi-Agent Systems (ICMAS'96). Keihanna Plaza, Kansai Science City, Japan. [http://foner.www.media.mit.edu/people/foner/Yenta/]

Foner, L N. 1997. *Yenta: A Multi-Agent, Referral-Based Matchmaking System*. In: Proceedings of The First International Conference on Autonomous Agents (Agents '97). pp 301-307. ACM Press. [http://foner.www.media.mit.edu/people/foner/Yenta/]

Frakes, W B & Baeza-Yates, R. 1992. *Information Retrieval: Data Structures and Algorithms*. Prentice-Hall. [The implementation: **Frakes, W B**. 1992**.** *Information Retrieval Code; stemmer*. Software Guild. [C source code]. [http://www.asset.com/WSRD/abstracts/ABSTRACT_330.html]. [Accessed 16/10/97]]

Höök, K. 1996. *A Glass Box Approach to Adaptive Hypermedia*. Chapter 2; *A framework for adaptive systems*. Department of Computer and Systems Sciences, Stockholm University. Sweden. Ph. D. thesis.

Höök, K. 1997. *Steps to take before IUI becomes real*. In: Proceedings of the 'The reality of intelligent interface technology' workshop. Edinburgh. [http://www.sics.se/~kia/papers/reality_of_II.html]

IBM Corporation. 1996. *WBI (Web Browser Intelligence) Personal Web Agent*. [Web page with software]. [http://www.networking.ibm.com/iag/iaghome.html#software]. [Accessed 29/11/97]

Janes, Sherman. *HTTP proxy server in Java*. [Java source code] [http//rabbit.cs.utsa.edu/~ggonzale/java/research.html]. [Accessed 20/10/97]

Kautz, K & Milewski, A & Selman, B. 1996. *Agent Amplified Communication*. In: Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI'96), Vol 1. AAAI Press/The MIT Press. pp 3-9. [http://portal.research.bell-labs.com/orgs/ssr/people/kautz/papers-ftp/ampl.ps]

Kilander, F & Fåhræus, E & Palme, J. 1997. *PEFNA The Private Filtering News Agent*. Technical report 97-004. Department of Computer and Systems Sciences, Stockholm University. [http://www.dsv.su.se/~fk/if_Doc/Pefna/pefna.ps.Z]

Koster, M. 1993. *Guidelines for Robot Writers*. [Web page]. [http://info.webcrawler.com/mak/projects/robots/guidelines.html]. [Accessed 05/01/98]

Koster, M. 1995. *Robots in the Web: threat or treat?*. In: ConneXions, Vol 9, No 4. Interop Company. Foster City, California. [http://info.webcrawler.com/mak/projects/robots/threat-or-treat.html]

Lashkari, Y & Metral, M & Maes, P. 1994. *Collaborative Interface Agents*. In: Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI'94), Vol 1. AAAI Press/The MIT Press. pp 444-450 [ftp://ftp.media.mit.edu/pub/agents/interface-agents/coll-agents.ps]

Lieberman, H. 1995. *Letizia: An Agent That Assists Web Browsing*. In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'95). Morgan Kaufmann Publishers. [http://lieber.www.media.mit.edu/people/lieber/Lieberary/Letizia/Letizia-Intro.html]

Maltz, A D. 1994. *Distributed Information for Collaborative Filtering on Usenet Net News*. Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology. M. Sc. thesis. [http://www.cs.cmu.edu/afs/cs.cmu.edu/user/dmaltz/Doc/mit-thesis.ps.gz]

Marsh, S. 1994. *Formalising Trust as a Computational Concept*. Department of Computing Science and Mathematics, University of Stirling. Ph. D. thesis. [http://ai.iit.nrc.ca/~steve/Publications.html]

Marsh, S & Masrour, Y. 1997. *Agent Augmented Community Information — The ACORN Architecture*. In: Proceedings CASCON'97, Meeting of Minds. [http://ai.iit.nrc.ca/~steve/Publications.html]

Menczer, F. 1997. *ARACHNID: Adaptive Retrieval Agents Choosing Heuristic Neighborhoods for Information Discovery*. In: Proceedings of the 14th International Conference on Machine Learning (ICML'97). Morgan Kaufmann Publishers. [http://www.cs.ucsd.edu/users/fil/Papers/ICML.ps]

Nwana, H S. 1996. *Software Agents: An Overview.* In: Knowledge Engineering Review, Vol. 11, No 3. Cambridge University Press. pp 1-40. [http://www.cs.umbc.edu/agents/introduction/ao].

Oppermann, R. 1994. *Adaptively supported adaptability*. Int. J. of Human-Computer Studies 40. Academic Press Limited. pp 455-472

Pazzani, M & Muramatsu, J & Billsus, D. 1996. *Syskill & Webert: Identifying interesting web sites*. AAAI Spring Symposium. Stanford, California. [http://www.ics.uci.edu/~pazzani/RTF/AAAI.html]

Porter, M F. 1980. *An Algorithm For Suffix Stripping*. Program 14 (3). pp 130-137

Sakagami, H & Kamba, T. 1997. *Learning Personal Preferences On Online Newspaper Articles From User Behaviours*. In: Hyperproceedings of the Sixth International World Wide Web Conference. Santa Clara, California. [http://www6.nttlabs.com/HyperNews/get/PAPER142.html]

Salton, G & Buckley, C. 1994. *SMART Information Retrieval System: common_words.* Department of Computer Science, Cornell University. [Text file]. [ftp://ftp.cs.cornell.edu/pub/smart]. [Accessed 08/10/97]

Sheth, B D. 1994. *A Learning Approach to Personalized Information Filtering*. Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology. M. Sc. thesis. [ftp://ftp.media.mit.edu/pub/agents/interface-agents/news-filter.ps]

Schoonderwoerd, R & Holland, O E & Bruten, J L & Rothkrantz, L J M. 1996. *Ant-based load balancing in telecommunications networks*. HP Technical Report HPL-96-76. Hewlett Packard. [http://www.hpl.hp.com/techreports/96/HPL-96-76.html]

Stone, P & Veloso, M. 1997. *Multi-agent Systems: A survey from a Machine Learning Perspective*. CMU CS technical report number CMU-CS-97-193. Carnegie Mellon University. [http://www.cs.cmu.edu/afs/cs/usr/pstone/public/papers/97MAS-survey/revised-survey.html]

Sun Microsystems, Inc. 1996. *RMI – Remote Method Invocation*. [Web page]. [http://java.sun.com/products/jdk/1.1/docs/guide/rmi/]. [Accessed 28/01/98]

UMBC KQML Web. *What is KQML?* [Web page] [http://www.cs.umbc.edu/kqml/whats-kqml.html] [Accessed 28/01/98]

van Rijsbergen, C J. 1979. *Information Retrieval*, 2nd edition. Butterworths. London. [http://www.dcs.gla.ac.uk/Keith/Preface.html]

Waern, A. 1996. *Recognising Human Plans: Issues for Plan Recognition in Human – Computer Interaction*. pp 13-45. Department of Computer and Systems Sciences, Royal Institute of Technology. Sweden. Ph. D. thesis. [http://www.sics.se/~annika/thesispage.html]

Yahoo! Inc. 1998. *Yahoo!* [Web page] [http://www.yahoo.com] [Accessed 01/12/97]

Appendix A (The implemented Graphical User Interface)

The graphical user interface consists of three parts as shown in figure 8a and 8b. At the top, there are two windows, the Recommendation applet and the dos-window. They show recommended documents respectively what the agent is doing. The window below is the browser with an applet that shows information about the chosen document and buttons to change the user model. There is a button to accept the predicted category and to add a new category. The emphasise and show-result buttons does not yet work. There is also a list showing all visited documents known by this applet and a list with categories to choose. A category named *Uninteresting* and its Interest Agent is the only category that is predefined. When the user adds a new category, it is added to all applets' list of categories and an Interest Agent is created for it. The Interest Agent for the category Uninteresting is currently the only agent that is marked "don't share" which means that it does not communicate with any other Interest Agent.

The user chooses a document to categorise by clicking in its frame in the browser that shows it and then chooses a category for it. When the browser downloads a document via the proxy, the Interface Agent predicts a category for it and when the user clicks in the frame, the prediction is shown as a pre-chosen category and the text "The Category is" is changed to "Prediction is" and a red line is drawn between the current title-URL and the rest of the buttons. In addition, the character "*" is added in front of the title-URL to indicate that it is not yet evaluated by the user. This is shown in figure 8a.

If the user wants to see a previously shown document, the user can choose a document from the document list as shown in figure 8b and then it downloads the document in the frame below the applet.

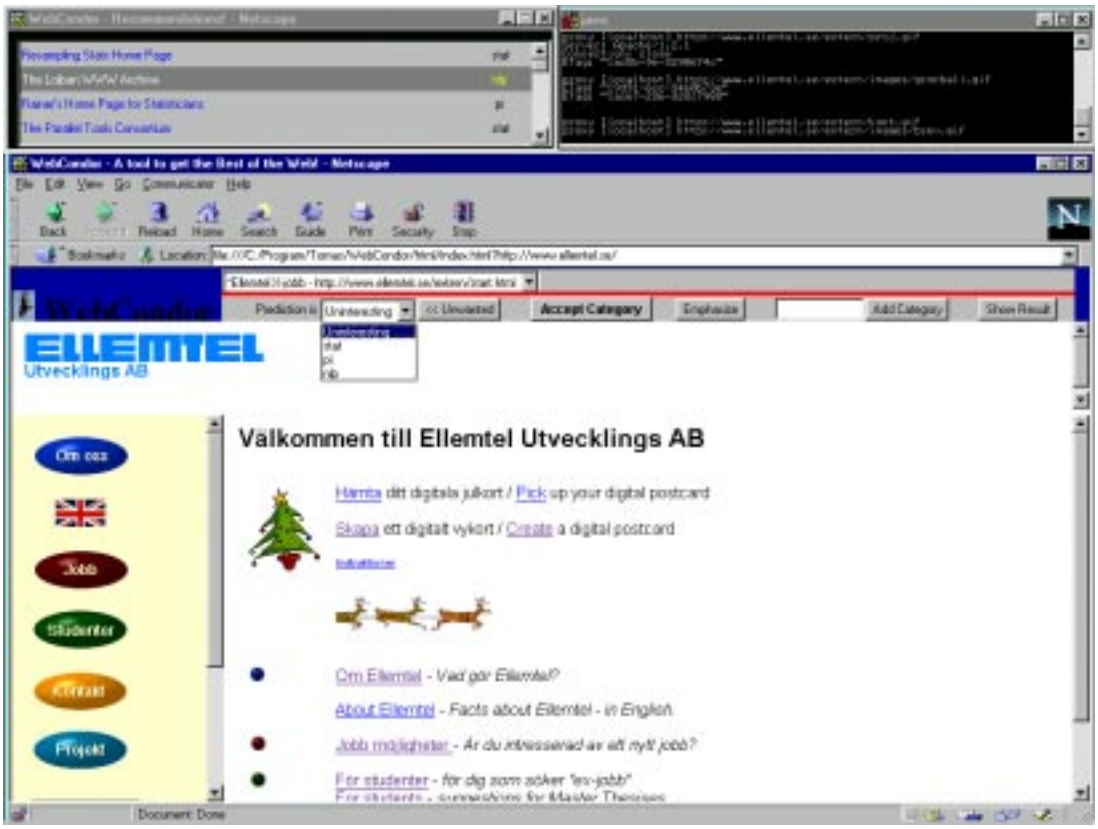

Figure 8a: The Graphical User Interface with a not yet evaluated document.

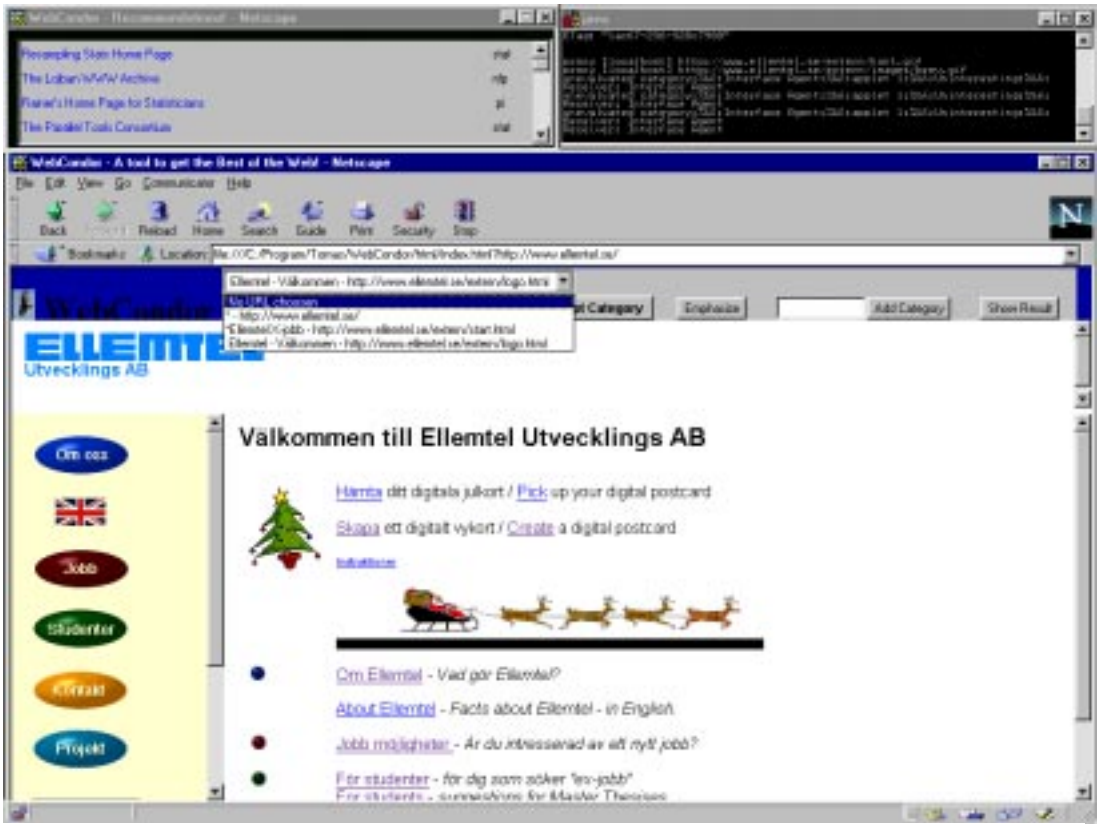

Figure 8b: The Graphical User Interface with an evaluated document.

In the applet with the recommended documents, each document is presented with its title (or URL if no title exists) and the predicted category. If the user clicks on the title-category row, the document is downloaded in a browser. Then the user can categorise it.

Appendix B

The answers from 13 students at KTH (original questions and answers in Swedish): Cursive text is the question, bold text is a predefined answer and normal text is the students own answers.

Do you usually use a web browser (for example, Netscape Navigator or Internet Explorer) to view home pages on the Internet?

Answer: Yes, 13 answers.

If you are using a browser, of what reasons do you do it? (the numbers are number of answers)

To surf around and discover new interesting pages:

Seldom	Rather seldom	Now and then	Rather often	Often
1	6	2	0	0

To find pages belonging to some special categories/interests:

Seldom	Rather seldom	Now and then	Rather often	Often
0	0	3	2	6

To find specific information, for example, an authors home page:

Seldom	Rather seldom	Now and then	Rather often	Often
1	0	2	5	4

Because of another reason and that is;

To be updated about what is going on:

Seldom	Rather seldom	Now and then	Rather often	Often
0	0	0	0	1

To check out things for which I have been given tips and to access web sites for which I have some responsibility:

Seldom	Rather seldom	Now and then	Rather often	Often
0	0	0	0	1

To look at pages I have marked to view:

Seldom	Rather seldom	Now and then	Rather often	Often
0	0	0	0	1

What do you usually do when you are:

1. Surfing?

I start with a good initial page or an index like Yahoo!.

I often have some favourite sites from which I start. I very rarely surf without any goal and then I prefer a search engine.

Eh... I click on links.

I often start from a page for which somebody has given me a tip or I use AltaVista to search in an area of interest.

I go to the C|net or SvD {a Swedish newspaper}, or some similar site with news, to see what is going on and I also follow links that I find interesting.

Via bookmarks or search engines.

I use search engines, such as AltaVista and from there, I follow interesting links. I also use sites with collection of links, such as Torget and Passagen {The square: http//www.torget.se respectively The passage: http://www.passagen.se}...

At random.

I search for a topic and examine the result, and I check the returned links.

I click on different links.

First, I usually query a search engine and then, I follow the returned links or I make a new query.

2. Looking for pages in a specific area?

I use an index site or a search engine.

I use AltaVista and I try to find keywords to get a suitable number of returned links.

It depends on how narrow the area is. Is it a wide/common area, then Yahoo! might be good to use. Otherwise, I try a search engine.

I use AltaVista, and I also look up URLs I have found in magazines and newspapers.

I use AltaVista.

I use Yahoo!.

Normally, I use advanced queries and I require keywords to be included with the plus sign.

I query AltaVista or some topical organised collection of links, such as Yahoo! or Sunet's index of Swedish pages {http://www.sunet.se}.

I use search engines, such as AltaVista and I search both for Usenet articles and for web pages.

I use some search engine, for example, AltaVista.

I search for a topic to see if there is something interesting and then I follow links.

I use the search engines Excite {http://www.excite.com} and Yahoo!.

First, I usually query a search engine, and then, either I follow the returned links

or I make a new query.

3. Search for any specific information?

I use a search engine.

I use AltaVista and I try to find keywords to get a suitable number of returned links.

I use a search engine. Most often, AltaVista, but sometimes other search engines.

I use AltaVista and I look up URLs I get from magazines and newspapers.

I usually know of some pages with links in the area from which I start. Otherwise, I query AltaVista.

Normally, I use advanced queries and I require keywords to be included with the plus sign.

I query AltaVista or some topical organised collection of links, such as Yahoo! or Sunet's index of Swedish pages.

I use bookmarks, Yahoo!, or some page with a collection of links, such as Torget and Passagen.

I use some search engine, for example, AltaVista.

I search for a topic to see if there is something interesting and then I follow the returned links.

I use the search engines Excite {http://www.excite.com} and Yahoo!.

4. Perform another task?

I visit other {peoples'} home pages, but then I often has the URL.

What do you find burdensome when you are using a web browser to navigate?

To find pages with collections of good links prepared by other users.

The waiting times. Lot of pictures and junk.

It might be hard to find good keywords to use when you query search engines.

The web is too slow and the browsers are not enough easy to use.

It easy to find information about computers, programming, protocols, etc., but it is hard to find information in other areas, such as history and geography. In that case, it is much easier to go to the local library. It is troublesome to not find the information one wants. There is a lack of alternatives, the queries are hard to formulate for good precision.

The search engines can not find everything and it is burdensome to check out all returned links.

There is a problem to get a general view of the information and links on the WWW.

It is hard to constrain the result of a query to only contain links from the area in which I am interested.

The lack of categorisations similar to Yahoo!.

The web is too slow, sometimes nothing happens.

That the search engines returns too many uninteresting links.

The time to download web pages is too long and you can not find what you want.

When the time to download a page is long and you can not use the next and previous buttons in a page with frames.

What kind of improvements would you suggest?

An improved filtering of the result returned from a query.

Higher network speed. There is also a need for more possibilities for graphics and interactions. Improved possibilities for communication with other people (when you want, most often it is good to be alone). A watchdog that, instead of me, checks if the pages I like have changed (this might already exist in my browser, I have not checked).

Improved possibilities to create my own tags (for example, for formulas) and document structures. Then it is possible to improve the search result. Improved possibilities to control the presentations of web pages (space, fonts, etc.).

Nothing, I let Microsoft do it. However, maybe a better language than HTML. Faster access to the web pages.

I want to know when new information is added to the web. As in Usenet news, but in specially designed groups only for my interests. For example, I could ask for Ron Rivest next article regarding cryptographic analysis or for something even more specific.

Back and forward (which there are in the most well known web browsers) ought to be replaced with some kind of tree structure.

One should be able to query, for example, AltaVista with an URL and get a list with links to similar pages. This would be very helpful and it should deserve a button in the web browser: "Find similar pages" (This Idea, I should sell to Digital and Netscape, if it does not exist already, but I have searched).

The browser should have a 3D-user interface.

Faster Java VM! :o) and more intelligent search engines/agents.

The pages should be categorised (how on earth this should happen...).

Faster Internet, more robust programs, programs that make the interpretation of the HTML-tags in a better way.

Faster transfer of data and a "clean" web browser without too much "junk".

Improve the capacity of the network.

Appendix C

A list of simulated users with corresponding agents and the documents found by the user belonging to each category of interest (agent). User A1, A2 and A3 have the same interests and user B1, B2 and B3 have the same interests. The difference between the users in the same group is that they find different documents in the same category. The total number of documents with URLs is 162.

User A1

Agent: Yahoo! - Science:Mathematics:Statistics

http://www.eecs.umich.edu/~qstout/abs/Seattle97.html
http://www.geocities.com/Colosseum/5585/mprev.html
http://seamonkey.ed.asu.edu/~behrens
http://www.siu.edu/~econ/bartlett.html
http://duke.usask.ca/~rbaker/stats.html
http://www.geocities.com/CollegePark/Quad/2435

Agent: Yahoo! - Science:Mathematics:Numbers:Specific Numbers:Pi

http://www.exploratorium.edu/pi/pi97/pi_one.html
http://www.mathsoft.com/asolve/constant/pi/pi.html
http://ernie.bgsu.edu/~carother/pi/Pi1.html
http://forum.swarthmore.edu/dr.math/tocs/pi.middle.html
http://www.escape.com/~paulg53/math/pi/index.html

Agent: Yahoo! - Computers and Internet:Supercomputing and Parallel Computing

http://www.irisa.fr/pampa/EPEE/epee.html
http://www.cs.virginia.edu/~mentat/homepage.html

Agent: Yahoo! - Science:Computer Science:Artificial Intelligence:Natural Language Processing

http://debra.dgbt.doc.ca/chat/chat.html
http://svr-www.eng.cam.ac.uk/comp.speech/
http://wwwots.let.ruu.nl/Controlled-languages/
http://www.dur.ac.uk/~dcs3mc/aifinance/durhamnlp.html

Agent: Yahoo! - Science:Computer Science:Artificial Intelligence

http://www.cs.cmu.edu/Groups/AI/html/faqs/ai/ai_general/top.html
http://ai.eecs.umich.edu/cogarch0/
http://www.webcom.com/~bsmart/aidef.html
http://www.cs.cmu.edu/Groups/AI/html/repository.html
http://198.110.216.210/

User A2

Agent: Yahoo! - Science:Mathematics:Statistics

http://www.ruf.rice.edu/~lane/hyperstat/contents.html
http://ouvaxa.cats.ohiou.edu/~wallace/class/demon2.html
http://www2.ncsu.edu/ncsu/pams/stat/info/jse/homepage.html
http://ourworld.compuserve.com/homepages/Rainer_Wuerlaender/stathome.htm
http://www.statistics.com/

Agent: Yahoo! - Science:Mathematics:Numbers:Specific Numbers:Pi

http://www.ccsf.caltech.edu/~roy/pi.html
http://www.mcs.csuhayward.edu/~malek/Mathlinks/Pi.html
http://k12.colostate.edu/ais/gunnison/pi/pi.html
http://www-groups.dcs.st-and.ac.uk/~history/HistTopics/Pi_through_the_ages.html
http://cid.com/~eveander/trivia/index.cgi

Agent: Yahoo! - Computers and Internet:Supercomputing and Parallel Computing

http://www.osc.edu/lam.html
http://www.nhse.org/nhsebig5.htm
http://anon.psc.edu/MetaCenter/MetaCenterHome.html
http://www.ptools.org/

Agent: Yahoo! - Science:Computer Science:Artificial Intelligence:Natural Language Processing

http://logos.svenska.gu.se/lbinfoeng.html
http://xiron.pc.helsinki.fi/lojban/
http://sap.mit.edu/projects/mit-lyon/project.html
http://www.dfki.de/lt/registry/index.html

Agent: Yahoo! - Science:Computer Science:Artificial Intelligence

http://www.geocities.com/CapeCanaveral/Lab/7677/index.html
http://www.botspot.com/main.html
http://www.cs.cmu.edu/afs/cs.cmu.edu/project/ai-repository/ai/html/air.html
http://colibri.let.ruu.nl/
http://cdps.umcs.maine.edu/Context/
http://www.geneticprogramming.com/index.html

User A3

Agent: Yahoo! - Science:Mathematics:Statistics

http://wkuweb1.wku.edu/Dept/Academic/Ogden/Math/instruct/statistics/stats.html
http://nilesonline.com/stats
http://lib.stat.cmu.edu/
http://avarice.gsm.uci.edu/~joelwest/SEM/
http://www.stats.gla.ac.uk/directory/

Agent: Yahoo! - Science:Mathematics:Numbers:Specific Numbers:Pi

http://members.aol.com/spoons1000/pi/index.html
http://www.spd.louisville.edu/~dsembr01/rationality-of-pi.html
http://www.ts.umu.se/~olletg/pi/trib.htm
http://www.aquest.com/~paris/pi.html
http://www.go2net.com/internet/useless/useless/pi.html
http://sac.uky.edu/~ksmcke0/Van_Ceulen/

Agent: Yahoo! - Computers and Internet:Supercomputing and Parallel Computing

http://www.geocities.com/SiliconValley/Vista/4015/
http://www.research.ibm.com/parallel_systems/
http://www.lpac.ac.uk/SEL-HPC/Articles/HpcArchive.html
http://pw2.netcom.com/~tandp/services.html

Agent: Yahoo! - Science:Computer Science:Artificial Intelligence:Natural Language Processing

http://itdsrv1.ul.ie/NLP/nlp_directory.html
http://www.signiform.com/
http://www.etl.go.jp/etl/taiwa/~hurst/research/tables/
http://www.cs.cmu.edu/Groups/AI/html/faqs/ai/nlp/top.html

Agent: Yahoo! - Science:Computer Science:Artificial Intelligence

http://www.zdnet.com/iu/archive/issue8/impostor/
http://www.mcs.net/~jorn/html/ai.html
http://www.elet.polimi.it/section/compeng/air/
http://www.lpac.ac.uk/SEL-HPC/Articles/index.html
http://www.cs.bham.ac.uk/~anp/TheDataMine.html
http://stud2.tuwien.ac.at/~e9425704/ai.html

User B1

Agent: Yahoo! - News and Media:Television:Genres:Science Fiction, Fantasy, and Horror

http://www.geocities.com/Hollywood/Hills/7343/
http://www.dwarflander.com/
http://www.geocities.com/Area51/Zone/2168/index.html
http://www.roblang.demon.co.uk/fangrok/
http://www.teleport.com/~keystar/reviews.html

Agent: Yahoo! - Entertainment:Science Fiction, Fantasy, and Horror:Art

http://www.the-net-effect.com/abandon-art/
http://www.info.polymtl.ca/~tranf/fantasy_world.html
http://www.teleport.com/~debdz/photo.html
http://www.geocities.com/SoHo/6301/

Agent: Yahoo! - Business and Economy:Companies:Food:Baked Goods:Cookies:Retail

http://www.mrsfields.com/
http://www.cyberweb1.com/auntg/
http://www.southwind.net/market/helen/
http://www.perfectcookie.com/
http://users.aol.com/bmcookies/cookies.html
http://www.cookiearrangements.com/
http://www.cookieisland.com/
http://www.tradecorridor.com/cookies/

Agent: Yahoo! - Business and Economy:Companies:Food:Desserts and Sweets

http://criweb.com/Atkins/
http://www.thebearfacts.com/
http://www.bindionline.com/index.html
http://www.atbeach.com/shopping/candyk/
http://www.clear-river.com/index.html
http://www.pastry-delites.com/
http://www.davidglass.com/

Agent: Yahoo! - Business and Economy:Companies:Food:Dairy:Ice Cream

http://www.brighams.com/
http://www.vitalo.com/country/
http://www.tiac.net/users/mstodaro/crystal/
http://www.speakeasy.org/dankens/
http://www.eskimopie.com/home1.htm
http://www.friendly.com/

User B2

Agent: Yahoo! - News and Media:Television:Genres:Science Fiction, Fantasy, and Horror

http://home.clara.net/paulj/
http://www.trekman.demon.co.uk/
http://www.surffamily.com/institute/mainContent.html
http://is2.dal.ca/~drclark/fiction.html
http://www.healey.com.au/~comgroup/shadow.html
http://fairfax2.laser.net/~epippi/
http://www.muchmusic.com/space/

Agent: Yahoo! - Entertainment:Science Fiction, Fantasy, and Horror:Art

http://www.hitech.net.au/goblin/
http://sf.www.lysator.liu.se/sf_archive/sf-texts/art/
http://www.lysator.liu.se/lothlorien/
http://members.aol.com/sfiwrite/main.htm

Agent: Yahoo! - Business and Economy:Companies:Food:Baked Goods:Cookies:Retail

http://www.cookiesfromeden.com/index1.html
http://www.cookietime.com/
http://www.corporatecookie.com/
http://www.cmbc.com/products.htm
http://www.cybercookie.com/
http://www.buffalochips.com/
http://www.greatamericancookies.com/home.html

Agent: Yahoo! - Business and Economy:Companies:Food:Desserts and Sweets

http://www.elodiespralines.com/
http://www.frederickspastries.com/
http://www.southwest-foods.com/
http://www.goodiesexpress.com/cookie.html
http://thegoodies.com/goodies2.htm
http://www.netcommunities.com/gourmetsorbet/home.htm
http://www.haitogloubros.com/products/index.htm
http://www.imcworld.com/imperial/

Agent: Yahoo! - Business and Economy:Companies:Food:Dairy:Ice Cream

http://www.garysicecream.com/
http://ourworld.compuserve.com/homepages/geebeecrewe/welcome.htm
http://www.heresthescoop.com/
http://www.haagen-dazs.com/
http://www.redestb.es/bornay/ibenseing/ibenseing.htm
http://www.lapperts.com/

User B3

Agent: Yahoo! - News and Media:Television:Genres:Science Fiction, Fantasy, and Horror

http://www.geocities.com/Area51/Corridor/1170/left.htm
http://www.geocities.com/Area51/3295/index.html
http://www.geocities.com/Area51/Vault/4308/lowframe.html
http://www.geocities.com/Area51/Vault/4308/x.html
http://www.pazsaz.com/scifan.html
http://www.geocities.com/Hollywood/6438/

Agent: Yahoo! - Entertainment:Science Fiction, Fantasy, and Horror:Art

http://www.scifi.com/scifi.con/gallery/
http://members.aol.com/maximania/about.htm

Agent: Yahoo! - Business and Economy:Companies:Food:Baked Goods:Cookies:Retail

http://www.his.com/~vidafina/
http://www.rust.net/martys_cookies/page2.html
http://www.mattscookies.com/home.html
http://Lenell.com/
http://www.maxwellscookies.com/

http://www.galaxymall.com/MTCookies/
http://www.nocookie.com/
http://207.48.113.83/cookies/ckhome.htm

Agent: Yahoo! - Business and Economy:Companies:Food:Desserts and Sweets

http://www.jell-o.com/jell-o100/
http://www.jetpuffed.com/
http://www.bostonian.com/kilvert/
http://www.kozyshack.com/
http://www.rec.co.nz/
http://www.sweetstreet.com/

Agent: Yahoo! - Business and Economy:Companies:Food:Dairy:Ice Cream

http://www.marbleslab.com/
http://www.perrysicecream.com/
http://www.cserveandina.net/alcazar/english.htm
http://www.sorbet.com/
http://www.treadwells.com/
http://www.wellsbluebunny.com/index4ie.html