Amalthaea: Information Filtering and Discovery Using A Multiagent Evolving System

Alexandros G. Moukas

Bachelor of Science, Business Administration
American College of Greece, Athens, Greece, 1994

Master of Science, Artificial Intelligence
University of Edinburgh, Edinburgh, Scotland, 1995

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning,
in partial fulfillment of the requirements for the degree of
Master of Science in Media Technology
at the Massachusetts Institute of Technology

June 1997

Author [illegible]
Alexandros G. Moukas
Program in Media Arts and Sciences,
May 9, 1997

Certified by [illegible]
Pattie Maes
Associate Professor, Program in Media Arts and Sciences,
Thesis Advisor

Accepted by [illegible]
Stephen A. Benton
Chair, Departmental Committee on Graduate Students
Program in Media Arts and Sciences

Amalthaea: Information Filtering and Discovery Using A Multiagent Evolving System

Alexandros G. Moukas

Submitted to the program in Media Arts and Sciences
School of Architecture and Planning
on May 9, 1997
in partial fulfillment of the requirements for the degree of
Master of Science in Media Technology
at the Massachusetts Institute of Technology

Abstract

Agents are semi-intelligent programs that assist the user in performing repetitive and time-consuming tasks. Information discovery and information filtering are suitable domains for applying agent technology. Amalthaea is an evolving, multiagent ecosystem for personalized filtering, discovery, and monitoring of information sites. Amalthaea's primary application domain is the World-Wide-Web and its main purpose is to assist users in finding interesting information.

Ideas drawn from the field of autonomous agents and artificial life are combined in the creation of an evolving ecosystem composed of competing and cooperating agents. A co-evolution model of information filtering agents that adapt to the various user interests and information discovery agents that monitor and adapt to the various on-line information sources is analyzed. A market-like ecosystem where the agents evolve, compete, and collaborate is presented: agents that are useful to the user or to other agents reproduce while low-performing agents are destroyed.

Thesis Advisor:
Pattie Maes
Associate Professor,
Program in Media Arts and Sciences

Amalthaea: Information Filtering and Discovery Using A Multagent Evolving System

Alexandros G. Moukas

Thesis Reader

Richard Belew
Associate Professor of Computer Science and Engineering
University of California, San Diego

Thesis Reader

Chrysanthos Dellarocas
Assistant Professor of Information Technology
MIT Sloan School of Management

Thesis Reader

Katia Sycara
Associate Professor of Robotics
The Robotics Institute, Carnegie Mellon University

Acknowledgments

This thesis is dedicated to my gradparents, Αλέκο Μούκα και Αθανάσιο Παναγιωτόπουλο, to which I am grateful for getting me interested in science and, more important, for teaching me how to think.

Thanks to family, Γιώργος Μούκας, Ιωάννα Παναγιωτοπούλου-Μούκα και Κατερίνα Μούκα. They always supported me and my decisions and helped me throughout my life.

Thanks to my academic advisor and research supervisor (two in one!) Pattie Maes for her assistance and guidance. Pattie is a great motivator: she always steered me towards feasible and interesting research questions and supported the idea behind this work from the very beggining.

Thanks to my three thesis readers, Rik Belew, Katia Sycara and Chris Dellarocas for providing valuable feedback and for being able to cope up with my somewhat strict deadlines (a direct result of my procrastination.)

Thanks to my hard-working UROPs Giorgos Zacharia (C++), Ronald Demon (Java), Asli Aker (Graphics/Interface Design) and Ozlem Ouzuner (Server scripts) who made this projects possible.

Thanks to all the members, current and past, of Agents group for (usually!) being such fun persons to work with: Bruce Blumberg, Anthony Chavez, Christie Davidson, Lenny Foner, Rob Guttman, Mike Johnson, Tomoko Koda, Akira Kotani, Henry Lieberman, David Maulsby, Agnieszka Meyro, Nelson Minar, Karen Modrak, Brad Rhodes, Juan Velasquez and Alan Wexelblat.

Thanks to all my friends, members of the KB and Αδράνεια underground campus organizations, for making sure that it would take me one more semester than necessary to complete this research.

Last but not least, thanks to Νατάσα Σταγιάννου for always being there when I needed her and for coping up with me during the write-up of this thesis.

Table of Contents

List of Figures

Chapter 1 Thesis Overview

In the first section of the chapter the problem domain and the techniques used are analyzed. Following the research contribution section, an outline of the whole thesis is presented.

1.1 Introduction

1.1.1 Problem Domain

The domain we focus on is that of personalized information filtering and information discovery. We introduce Amalthaea, a personalized system that pro-actively tries to discover information from various distributed sources that may interest its user and presents it to her in the form of a digest. Amalthaea learns about the user's interests by examining the hotlist and browsing history and by getting feedback in the form of ratings of documents (usually Universal Resource Location strings - URLs). The system autonomously collects related documents and URLs. It considers three domains in parallel:

- World-Wide-Web documents and **data discovery**. Amalthaea does not search the WWW itself but instead launches multiple agents that utilize existing indexing engines and perform a "meta-search'" in order to discover information that is broadly of interest to the user. The system then further analyzes the retrieved documents using weighted keyword vectors techniques in order to select those closer to the user's preferences.
- Continuous flow of information environments and **information filtering.** Again, multiple agents are analyzing the articles and select only the proper ones. In this case, instead of pro-actively searching the web, Amalthaea filters a stream of incoming documents, like the Clarinet news or the Reuters newsfeed.
- **Monitoring** of frequently changing information resources. Sometimes the user wants to monitor certain URLs that are updated in fixed (or random intervals). For instance he wants to know about the new Formula 1 motor

races results that are updated every other Sunday. Or the user wants to track new articles on an on-line journal, or new CDs by an artist. Such URLs will be monitored in regular intervals for changes. If such changes do exist and are significant then the user will be notified.

Amalthaea's operation does not require the presence or the attention of the user. The information is presented to the user in the form of a digest: new URLs that might be interesting, news that is personalized, notification of new material in certain sites. The user browses the digest, is able to follow the links, gives feedback on how good or bad an item is, rates the relevance of an item or keyword etc. The idea is that the system adapts to the user and follows his interests as they evolve over time.

1.1.2 Techniques Used

We implemented Amalthaea by creating an artificial ecosystem of evolving agents that cooperate and compete in a limited resources environment. Two general species of agents exist: the *Information Filtering (IF) Agents* and *Information Discovery (ID) Agents.* Information filtering agents are responsible for the personalization of the system and for keeping track of (and adapting to) the interests of the user. The information discovery agents are responsible for handling information resources, adapting to those information sources, finding and fetching the actual information that the user is interested in. When trained, the discovery agents become better at in specifying which search engines are good in returning what kind of documents.

Evolving[1] a multiagent solution is particularly suited to this domain since it provides a way of utilizing the best possible existing solution to the problem (in our case the best possible match to the user's interests) with the ability to quickly adapt to new situations (for instance following a new user interest or adapting to changes in the domain). At the same time the system continues to explore the search space for better solutions using evolution techniques such as mutation and crossover for refreshing and specializing the agents population.

Significant work has been done on applying artificial intelligence techniques to information filtering. In contrast, our work is inspired by the artificial life approach resulting in fully distributed learning and representation mechanisms. In order to understand the system's global behavior in filtering one has to think of each information filtering agent as a very specialized filter that is applied only in a narrow sector of the domain. When a user changes interests, the filters assigned to the old interests are eventually destroyed and new ones are created that are "directed" towards the new interests by evolution and natural selection. The intriguing issue in such multiagent systems is to find ways of

1. Evolution is obviously not the only method of adaptation for a multiagent system. One could also use techniques where the agents learn during their life time. For example techniques like induction and case-based reasoning (to name a few) also yield promising results.

allowing the system to reach self-imposed equilibria while continuously adapting to new user interests. In order to evaluate the performance of each agent we have established the notion of an ecosystem which operates on the basis a simple economic model: the agents that are useful to the user and to other agents get positive credit, while the bad performers get negative credit. The more credit an agent has the better it stands and the less chances it has to be destroyed.

Amalthaea is uses an agent-based evolutionary architecture. The system is adapting by evolving a number of agents and trying to bring them closer to the user interests and not by individual learning in the single agent level.

The major components of the system include:

- The user interface which is browser and Java-based and enables the communication between the user and the application
- The information filtering agents which request broad categories of documents that are similar to the user profile
- The information discovery agents that given the above-mentioned requests, query various search engine to get relevant sites and then fetch the contents of those sites.
- The evolution mechanisms that are applied to the above two populations
- The spiders that retrieve the actual documents at the sites returned by the search engines
- The text processing and vectorization mechanisms that given a retrieved document produce a keyword vector
- The credit allocation mechanisms that convert the rating that the user give to the system's recommendations to credit. Consequently, those mechanisms distribute the credit to the filtering and discovery agents that were responsible for presenting that digest item
- The database where the URLs of all the retrieved articles are stored in order not to make duplicate recommendations to the user.

The terms "filtering" and "discovery" might seem very difficult to coexist in a single system, since the literature tends to see them as opposites. However, the documents that are discovered (retrieved) in Amalthaea are presented to a population of agents that perform the filtering. The system is functioning along both edges of the scale at the same time.

Along the "agent pro-activeness vs. user querying/user initiating the action" axis, Amalthaea is positioned very close to the pro-activeness side: after the system is bootstrapped by the user it autonomously collects interesting articles, always running in the background. The system presents information without the user explicitly asking it to do so or formulating a query.

1.2 User Experience

Amalthaea is a tool that pin-points interesting sites and presents the users with a digest of ten or twenty documents that will be of interest to them. Amalthaea acquires the user's interests in four different ways. Users can:

- submit a bookmark list with favorite sites/documents to provide a starting point for the system.
- submit the browser history files so that the system can identify patterns in their behavior (like visiting a web site at regular time intervals)
- select pre-trained "packages" of agents (each package focuses on a particular topic; for instance "European soccer", "Greece", "Agents research" etc.) to speed-up Amalthaea's learning.
- specify a specific interests by training the system based on some relevant documents.

Based on that information, the system will use search engines to find other similar documents. Amalthaea will recommend those documents to the user and will keep improving by receiving two forms of feedback: direct, where the user rates a specific document and indirect, where the system sees how much time a user spends inside a web page, and computes a "likability" factor for that page. Users can monitor Amalthaea's operation (through the visualization of the system state and of the visited URLs) and adjust its behavior accordingly.

1.3 Research Contributions and Results

The research goals of this work are:

- To investigate artificial life techniques and their application to information discovery and filtering; introduce co-evolution of agents in an ecosystem and explore the relationships between the two populations.
- To explore learning mechanisms not in individual agents but as an emerging property of a system.
- To apply evolutionary techniques to information filtering and discovery.
- To provide ways of expressing to the user the state of multi-agent systems through visualization

The project also produced a prototype (called Amalthaea). In order to validate the effectiveness of Amalthaea we had to show that the system's evolutionary algorithms were converging, that the ecosystem was properly designed and was able to reach equilibria points and that users found that the system provided them with useful suggestions.

The initial set of experiments we performed was focused on setting up the initial set of primitives and parameters for the evolution and the economic model.

After a stable platform for experimentations was established, we focused on fine-tuning the properties of the evolutionary algorithms (mutation, crossover, cloning) and the credit distribution and flow between the different components of the system, the Information Filtering Agents and the Information Discovery Agents.

Consequently we performed real-time iterative experiments by building an external system that extracted the profiles of different users and was giving automatic feedback to Amalthaea. Those experiments allowed us to understand how the system was dealing with real user profiles and deduce the optimal ratio of Information Filtering Agents to Discovery Agents.

Our final experiments involved actual user interaction with the system and we measured the effectiveness of the system as perceived by its users in terms of measuring the error in the system's recommendations, the rate at which the error was decreased as the system adapted to the user and the precision of the system in recommending relevant documents. Significant parts of this work were published and presented at international conferences and journals (Moukas, 1996, Moukas and Zacharia, 1997, Moukas, 1997).

1.4 Thesis Outline

Chapter one contains an overview of the thesis.

Chapter two begins with a discussion of the motivation for this work and introduces key research issues, terms and definitions. It also describes the main research topics of Amalthaea, namely information filtering and discovery, user modeling and agent-based evolutionary architectures. The focus of this chapter is to present the state of the art of technology used in this project, and discuss similar agent systems.

Chapter three discusses the functionality of Amalthaea, and its user interface: how the user builds his profile, configures, fine-tunes and uses the system. Different aspects of the user interface are presented, along with details of its various subcomponents, such as digest presentation, new filtering agents generation, monitoring of sites and browser control. A description of the bootstrapping procedure is followed by discussions on ways of visualizing multi-agent systems and secure client-server communication schemes.

Chapter four extensively describes the system and the relationships between its sub-components: the document discovery engine, information filtering agents, the information discovery agents, the interactions between them and the economic model methods used. The chapter concludes with a synopsis of credit assignment and information flow in Amalthaea.

Chapter five explains in detail the implementation techniques used. It introduces the different components of Amalthaea and their input and output mechanisms from a software engineering point of view. Furthermore, it discusses certain security and privacy considerations on the design of the system. We try to reveal possible privacy attack methods, and offer suggestions about different encryption-based and architecture-based solutions.

Chapter six is devoted to the testing and evaluation. We introduce two main categories of evaluation techniques, virtual users and real users. Virtual users utilize the profiles of real users and their interests evolve over time. They allow us to perform many iterations on the same data set. Real user experiments are based on a set of seven people that used the system over a certain time period.

Chapter seven contains the concluding remarks of this work. It introduces several limitations and drawbacks of Amalthaea and identifies possible areas of future work. The thesis wraps up with conclusions drawn during the progress of the research and a summary.

Chapter 2 Motivation, Issues and Related Work

This chapter begins with a discussion of the motivation for this work and introduces key research issues, terms and definitions. It also describes the main research topics of Amalthaea, namely information filtering and discovery, user modeling and agent based architectures. The goal of this chapter is to present the state of the art of the technology used is in this project and discuss similar agent systems.

2.1 Motivation

The exponential increase of computer systems that are interconnected in on-line networks has resulted in a corresponding exponential increase in the amount of information available on-line. This information is distributed among heterogeneous sources, is often unstructured and continuously changing (as in the case of the World Wide Web.) As it is becoming more and more difficult for users to cope with such amounts of information, new tools like software agents need to be devised to assist in dealing with information overload. Agents, semi-intelligent computer programs, will increasingly be used to assist in handling repetitive and time-consuming tasks. In order for agents to be of real help to the user they have to **learn** the user's interests and habits using machine learning techniques, maintain their competence by **adapting** to the changing interests of the user while at the same time **exploring** new domains that may be of interest to the user. We are witnessing a paradigm shift in human-computer interaction from "direct manipulation" of computer systems to "indirect management" in which agents play a key role (Maes, 1994).

Search engines and sites indices are available to people interested in finding more information about a particular topic. In order to formulate a query to a search engine one needs to know what one is searching for. Furthermore, search engines usually return tons of information unless the query is formulated very carefully. Indices like Yahoo contain links only to interesting pages (as rated by humans) but one usually has to be familiar with the categorization ontology of the index in order to easily find information she needs. On top of that, hundreds of sites are getting added each day and the average user has neither the time nor the inclination to find out if they contain something of interest to her.

We see the need for a pro-active (i.e. not query-driven) system that would operate on the background and present to the user a digest of sites that the agent thinks are of interest to the user. We also want our system to be usable by a majority of users so it cannot require a permanent internet connection. In response to this we have biult a system called Amalthaea. The system's user interface is based on HTML and Java. The system is pro-active and non-intrusive. It runs continuously on a remote server, even when the user is not logged-on, and presents its results at the next possible time.

2.2 Key Issues, Terms and Definitions

This work spans several different research fields, such as information filtering, information discovery and retrieval, user modeling, evolution and multi-agent systems. Researchers in those fields sometimes use different terms to refer to the same concepts. In this chapter we will try to identify the key issues and define the key terms that will be used:

- Information filtering and retrieval systems appear similar in that they both try to return useful and relevant information to the users.
- However, filtering systems try to fulfill long-term user goals, while retrieval systems usually focus on one-time queries.
- User Modeling describes the effort to create profiles define the interests of individual users and use those profiles in a variety of tasks.
- Agents are semi-intelligent, proactive computer programs that help users cope with repetitive tasks, and deal with information overload.

The terminology we use is influenced by the bottom-up, Artificial Life approach we are using to build Amalthaea. Some members of the scientific community might disagree with this terminology. Let's take for instance the relationships between the Information Filtering and Discovery Agents: some may disagree with the use of the terms"cooperation" or "collaboration" because this implies a more active desire to cooperate maybe through formal exchange of information, through KQML (Labrou and Finin, 1994) for instance. Again, based on a bottom-up, biologically inspired approach we use the term "collaboration" to refer to a form of symbiosis. In general, interpreting the terms depends a lot from what field one approaches agents from.

2.3 Information Filtering

Information filtering systems usually share the following characteristics (Belkin and Croft, 1992)

- They involve large incoming streams of data
- They primarily deal with unstructured or semi-structured textual data
- They are based on some sort of predefined filter

- Their objective is to prune data that does not match the filter rather than locate data

Information filtering systems can be categorized along several different axes based on the technology/architecture they use for filtering the data, the location of their operation relative to the information source and the user, the way they find information sources and the way they represent different user profiles.

Several different architectures have been proposed for building effective and efficient filtering systems. They can all, however, be classified under two broad categories:

- Content-based filtering, where the system actually processes a document and tries to extract useful information about its content. The techniques used in content-based filtering can vary greatly in complexity. Keyword-based search is one of the simplest techniques that involves matching different combinations of keywords (sometimes in boolean form). The Newsweeder system (Lang, 1995) was designed for filtering in USENET newsgroups. The Fishwrap system (Chesnais et al., 1995) developed at the Media Lab at the early 90s is another such system. Statistical keyword analysis represents a more advanced form of filtering, where the stop-words are removed from the document, the rest of the words are stemmed, vectorized and given a weight based on their significance. Introduced in the SMART system in the late 60s, early 70s (Salton, 1971), this form of representation is one of the most popular. A more advanced form of filtering is the one based on extracting semantic information of the documents' contents. This can be achieved by using techniques like associative networks of keywords in a sentence (Riordan and Sorensen, 1995), directed graphs of words that form sentences, or exploration of the semantic meaning of words by using tools like WordNet, a lexical reference system (Miller, 1985).

- Social (or collaborative) and Economic-based filtering, where the system utilizes feedback and ratings from different users to filter out irrelevant information. These systems do not attempt to analyze or "understand" the contents of the documents; instead they are using the impressions that the users had when they were reading them to create a "likability" index for each document. This index is not global, but is computed for each user on the fly by using other users with similar interests: documents that are liked by many people will have a priority over documents that are disliked. Economic-based filtering augments this idea with a cost-benefit analysis on behalf of the user. It takes into consideration parameters like the price of the document and its cost of transmission from the source to the user (in the case of company intranets) when making decisions on whether to filter it out or not. Systems that use these approaches include Lens (Malone et al., 1987), GroupLens (Resnik et al., 1994) Webhound, (Lashkari, 1995), and Ringo (Shardanand and Maes, 1995).

Based on the location where they operate relative to the user and the data source, information filtering systems can be divided into two broad categories: systems located at the source of information and systems located in the user's machine (we assume that the systems that are filtering information at some point inbetween the user and the source belong to the former category.) Both approaches, have their advantages and disadvantages. The server-side filtering approaches main advantage is minimization of network traffic, since all non-relevant articles are filtered before reaching the end user. On the other hand, a server based system requires that users send their profiles to a centralized place, with all the update limitations and privacy considerations that this design decision carries.

Finally, filtering systems can described as "active" or "passive" based on the methods they employ for finding the information. Passive are the systems that filter out a given information stream, while active are the ones that besides this stream try to find relevant information at an external database or even the WWW. As mentioned before, the user profile representation methods used by different systems vary. Smart-like keyword vector representations, keyword search based rules, neural networks and genetic algorithms are among the most popular.

2.4 Information Retrieval

Although they appear similar to filtering systems, Information Retrieval (IR) systems have inherently different characteristics (Belkin and Croft, 1992):

- Information retrieval usually deals with static databases of information as opposed to the dynamic streams used in filtering.
- Information retrieval systems are usually query-based as opposed to non-query driven filtering systems.
- Information retrieval focuses on single interactions of the user with the system and assumes that by the end of each query the system has reached its goal (short-term goals). Information filtering assumes a different approach to the interaction of the user with the system and tries to identify long-term goals, that span over multiple interactions.
- Information retrieval mainly focuses on static, organized, and structured databases

Information Retrieval systems are divided into three major categories: Boolean-based systems, Vector-space based systems and probabilistic systems (Salton and Buckley, 1987, Salton, 1983).

- Boolean systems are based on keywords or phrases that are combined with boolean operators (like AND, OR, NOT) to form queries. These systems are also called exact-match systems because an exact match is needed between the textual elements of the query and the contents of the database elements that will be retrieved.
- The vector-space model uses multi-dimensional vectors composed of keywords and weights to represent queries and retrieved text. One of the advan-

tages of vector space models, is that unlike boolean systems, the retrieved documents can be ranked based on their relevance.

- Probabilistic models try to analyze the statistical distribution of terms in the database and identify relevant and non-relevant items using inference network models like Bayesian networks.

2.5 User Modeling

User modeling can be defined as the effort to create a profile of the user's interersts and habits and employ the profile in order to improve human-computer interaction. User Modeling systems differ in the ways they acquire, use and represent a user profile. Profiles can be acquired or generated in a variety of ways:

- By direct user interviews
- By "knowledge engineers" using user stereotypes (that is, a collection of interests that are shared by users that belong to a specific group.) For instance the stereotype of "Computer Science" users would include a subcategory "programming" into their profile.
- Machine learning techniques like inference, induction, where the modeler tries to identify certain patterns in the user's behavior.
- Profile building by example, where the user provides examples of his/her behavior and the modeling software records them.
- Rule-based profiles, where the user specify their own rules in the profile, rules that control the behavior of the model under certain trigger conditions.

The above-mentioned methods have their advantages and disadvantages, but in general the most successful are those that try to analyze the information not just at a keyword level, but sometimes at a contextual and semantic level. User profiles can be represented using a wide range of techniques, from simple keyword-based files, to artificial-intelligence based representations. The representational formats include keyword-based profiles, rule-based profiles, vector representations of collections of keywords and weights and Neural network-based representations (usually associative and backpropagation networks).

It is useful to note that a typical user has multiple and sometimes overlapping interests. The categories of interests in stereotype user profiles must be very fine-grained and the user has to select those categories on her own in order to build her profile. User interviews are very time-consuming, sometimes the users fail to properly identify categories of their interests and the cost of profile maintenance is very high. The use of machine learning techniques for generating and maintaining user profiles in information filtering applications is therefor very compelling.

2.6 Software Agents

Presently, in the Computer Science and Artificial Intelligence literature, the use of the term "agent" is overwhelming. Almost any system that performs a function and is described as a black box is labeled an agent. It is getting increasingly difficult to draw a line amongst different systems and clearly define what is and what it is not an agent. Agent systems should best be viewed as a direction towards which software should be headed. Nonetheless, in our opinion agent systems must have a set of different characteristics:

- They must be pro-active, act without a direct command, on behalf of the user
- They must be personalized, that is they acquire the user's interests and adapt as they evolve over time
- They must be persistent, either run continuously or save their state, so the user can see the agent as a stable entity and develop a trust relationship with the agent.

In general, agent and multi-agent systems like Amalthaea are not trying to solve the old AI problem, they are just tools to assist human capabilities and lead to a world of augmented intelligence (Maes, 1997).

2.7 Evolutionary Techniques

Adaptation in computer systems via evolution was first proposed by John Holland as a limited framework (Holland, 1962) and later in a full-blown form (Holland, 1975). Evolutionary techniques have been used to evolve program code in search for an optimal solution to a programming problem (Koza, 1992). It has also been used to evolve collections of parameters rather than the code that uses them (for instance neural network evolution, general optimization problems, etc.)

Evolutionary systems draw paradigms from biology and apply them to computer science and usually refer to populations of computational entities that compete. Optimization or adaptation is a result of the overall/macroscopic behavior of the system, and not of individual learning by the members of the population. However evolution can coexist with systems where the individual entities exhibit some sort of learning (Belew and Mitchell, 1996). Amalthaea falls in the former category.

2.8 Multiagent Systems

Multi-Agent systems provide a collaborative way of solving complex problems by multiple agents. Multiagent systems can differ along the axes of complexity (ranging from completely reactive, cellular-automata like agents to very complex ones); homogeneity (all agents are identical in terms of programming but they process different data, similar to a SIMD massive parallel architecture programming paradigm or completely different agents); communication (actively communicating with each other or using the changes in their environments as a communication means) and learning (individual agents can have a plastic learning component or can be static). Multiagent systems have been used in telecommunication network switching, load balancing (Chavez et al., 1997), robotic communication and group behavior (Mataric, 1993 and Moukas and Hayes, 1996), industrial process management, electronic marketplaces (Chavez and Maes, 1996.)

2.9 Related Software Agent Systems

Metacrawler (Etzioni, 1995) is an agent that operates at a higher abstraction level by utilizing eight existing WWW index and search engines. Metacrawler is an example of a "parasite" agent that does not index the documents itself, but provides a common interface to a number of search engines. The user posts his/her query once, and metacrawler forwards it to all search engines, collects the results and returns a unified list. It is easy to extend this approach to a higher level of abstraction and have agents that filter information which consult agents that discover information, which in turn consult search engines that index information. By creating several processing levels between the actual information and the user, we allow for greater flexibility in utilizing other novel forms of filtering or other forms of discovery. Etzioni is referring to that as the information food chain and is advocating that Metacrawler is an information carnivore high up the information food source (Etzioni, 1996).

Webcompass is a WWW product by Quarterdeck. Webcompass is directed towards off-line search and indexing. It enables the user to generate queries that will search the WWW off-line and presents the results at a later time. The MACRON multiagent system (Decker and Lesser, 1995) developed at UMass/Amherst, is built on top of the CIG searchbots and uses a centralized planner to generate sub-goals that are pursued by a group of cooperating agents, using KQML (Labrou and Finin, 1994), a standardized language for inter-agent communication and negotiation. A comparable system is RAISE (Grosof, 1995), developed by IBM. RAISE is a rule-based system that provides a framework for knowledge reuse in different domains (like electronic mail, newsgroups e.t.c.) INFOrmer (Riordan and Sorensen, 1995), developed at University of Cork introduces the idea of using associative networks instead of keywords for information retrieval. CMU's RETSINA project (Sycara et al., 1996, Sycara and D., 1996, Decker et al., 1997) defines a framework for distributed intelligent agents. This framework was applied to organizational decision making in the Pleiades system (Sycara, 1995) and to financial investment management in the Warren system. Pleiades introduces task-specific and information-specific agents deployed in different levels that can collaborate with one another to provide the information requested by the user.

Another category of WWW agents includes Carnegie Mellon's University Webwatcher (Armstrong et al., 1995) and MIT Media Laboratory's Letizia

(Lieberman, 1995). These agents are designed to assist the user and provide personalization, while the user browses the WWW. They perform a breadth-first search on the links ahead and provide navigation recommendations. More similar to our work in terms of application domain and representation is the system built at Stanford (Balabanovic and Shoham, 1995). They introduced a system for WWW document filtering which also utilized the weighted keyword vector representation.

In terms of evolutionary filtering systems, NewT (Sheth and Maes, 1993) developed at the Media Lab, is a multiagent system that uses evolution and relevance feedback for information filtering. NewT's application domain is structured newsgroups documents (clarinet) and the system is able to adapt successfully to such a dynamic environment. The main difference between NewT and Amalthaea (apart from the application domain) is that NewT employed only one kind of agents, namely specialized information filters, while the system presented here introduces different types of agents which base their relationships on a simple market-based model. The Infospiders or ARACHNID project (Menczer et al., 1995) at University of California at San Diego combine evolutionary techniques with the idea of endogenous fitness to create a scalable distributed information retrieval system. Finally analysis of text fitness in the internet in an ecological framework was done at the MIT Media Lab using LSI techniques (Best, 1997).

Chapter 3 Functionality of Amalthaea

This chapter discusses the functionality of Amalthaea, and its user interface: how the user builds his profile, configures, fine-tunes and uses the system. Different aspects of the user interface are presented, along with details of its various subcomponenets, such as digest presentation, new filtering agents generation, monitoring of sites and browser control. A description of the bootstrapping procedure is followed by discussions on ways of visualizing multi-agent systems and secure client-server communication schemes.

3.1 Introduction

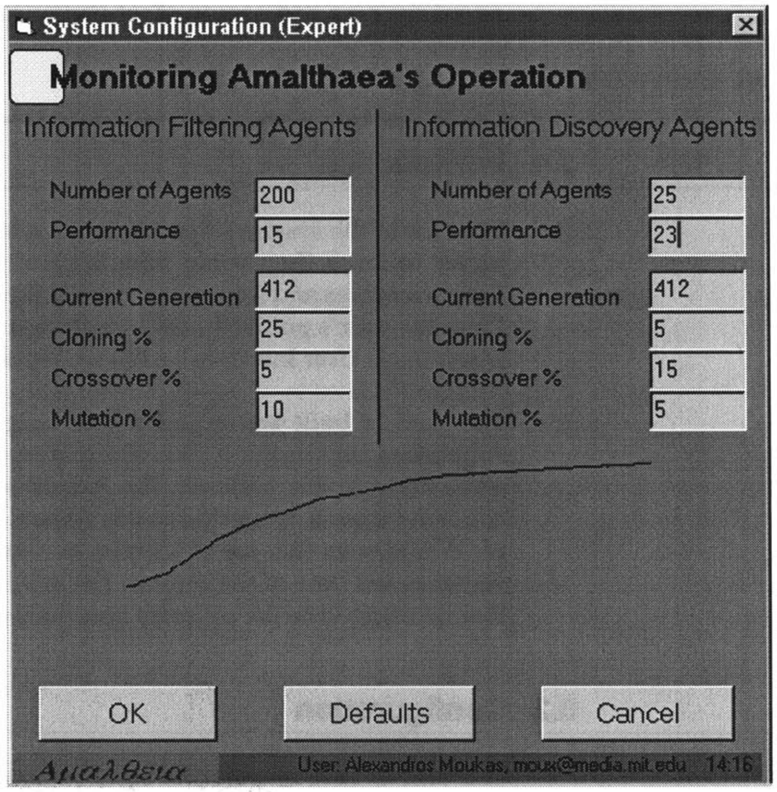
The core of the system's operations runs on a centralized Amalthaea server. The server contains the various Filtering and Discovery Agents for each user, his/her preferences and information about the sites that the user has already visited. From the user's point of view, Amalthaea is controlled via a graphical interface (Amalthaea User Interface - AUI) that runs on the his/her computer.

The AUI is built using Sun's Java language. When a user connects to the Amalthaea server, the AUI is brought up on his/her screen as a Java window separate from the browser. The Amalthaea interface is continuously running while the user is connected to the system. Via the AUI the user can receive lists of WWW sites that are of interest to him or her, give feedback, configure and visualize the state of the system. The AUI is composed of several different parts that correspond to the different functions of the system.

3.2 Configuration

The user is able to alter the basic parameters of Amalthaea and visualize its state. The basic parameters can be configured in two modes. The low level mode (Expert) allows manipulation of variables like evolution, mutation, and crossover rates as well as the number of filtering and discovery agents in the system and the way they collaborate. The higher level mode is more suitable to end users: it provides higher-level abstractions for the above mentioned low-level parameters. Those abstractions include configuration settings such as "Quick Learning" vs. "Slow Learning", "Fast Adaptation", vs. "Stable Interests", etc., that tune the low-level variables in such a way that enable the system to either learn faster but with less accuracy and breadth or slower but more accurate. For instance the evolution rate of the system is lower in the "Slow Learning" setting than it is in the "Quick Learning" setting. The default values in the above-mentioned configuration settings were assigned to the system after performing a set of small-scale experiments on the evolution parameters.

In the Configuration part of the AUI the user is also able to view the current state of the system in terms of fitness of Agents and their evolution over time (in the form of a plot). Each user can further tailor Amalthaea to his/her need by modifying parameters like the number of documents the system should retrieve and present to the IFAs for filtering and the amount of documents the IFA should present to the user. A snapshot of the AUI Configuration Window can be found in Figure 3 - 1.


System Configuration (Expert)

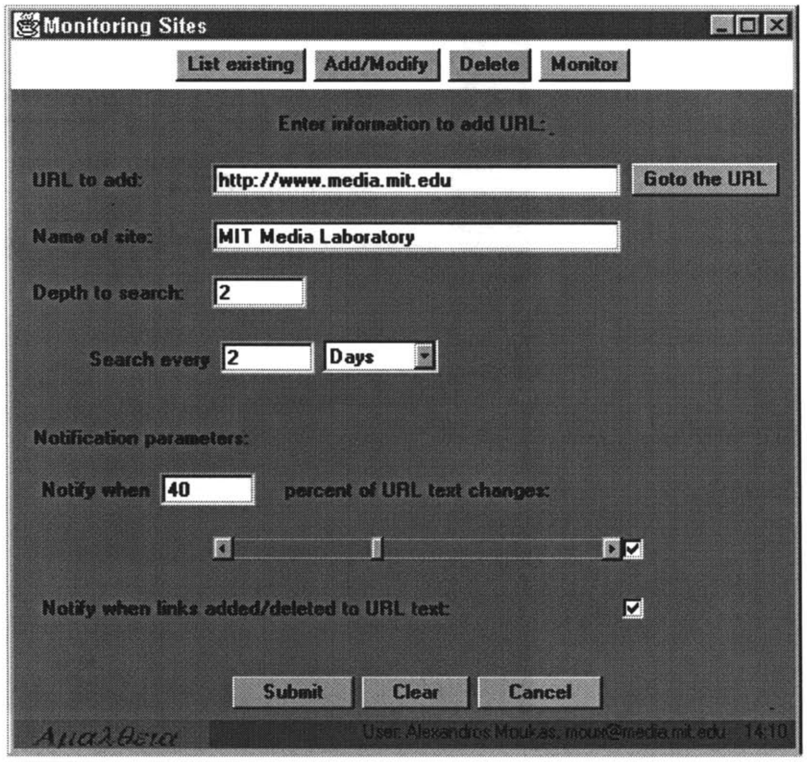
Monitoring Amalthaea's Operation

Information Filtering Agents		Information Discovery Agents	
Number of Agents	200	Number of Agents	25
Performance	15	Performance	23
Current Generation	412	Current Generation	412
Cloning %	25	Cloning %	5
Crossover %	5	Crossover %	15
Mutation %	10	Mutation %	5

Figure 3 - 1 The Expert Configuration Mode

3.3 Monitoring Sites

The user can specify that certain sites which are updated every certain time intervals should be monitored by Amalthaea. The user can enter information like how often should the site be checked, and whether he wants the monitoring to include URLs that are contained in that site as well (and up to what depth). Other configurable options include parameters like when should the user be notified (i.e. Notify when something in the site changed? When a new URL was added or deleted from that site? When the site changed above a certain percentage?) In a way this function of Amalthaea acts like "active bookmarks".



Enter information to add URL:

URL to add: http://www.media.mit.edu

Name of site: MIT Media Laboratory

Depth to search: 2

Search every 2 Days

Notification parameters:

Notify when 40 percent of URL text changes: ✓

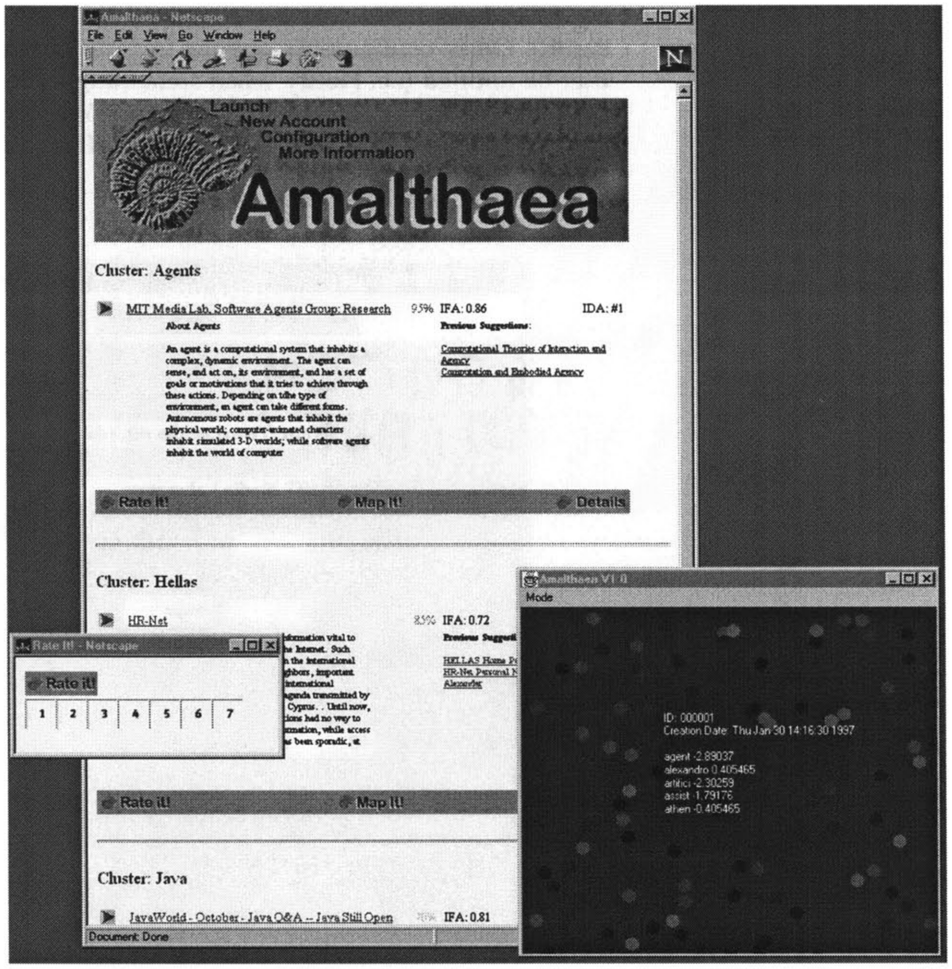
Notify when links added/deleted to URL text: ✓

User: Alexandros Moukas, moux@media.mit.edu 14:10

Figure 3 - 2 Monitoring Sites Window

3.4 The Digest

This is the part of the AUI where the system presents its recommendations to the user and she gives feedback. The recommended URLs are organized in sections (where each section contains the documents that were retrieved by a certain cluster of information filtering agents), and are displayed along with a small part of the actual contents of the site and the confidence level that the system has about each URL. The user can click on those URLs and the AUI opens the browser to that specific URL. He can check out the site and then select a rating for that suggestion (the AUI is always running in a separate window outside the browser.) This rating is going to be user's feedback to the system.



Cluster: Agents

MIT Media Lab, Software Agents Group: Research 95% IFA: 0.86 IDA: #1

About Agents

An agent is a computational system that inhabits a complex, dynamic environment. The agent can sense, and act on, its environment, and has a set of goals or motivations that it tries to achieve through these actions. Depending on tdhe type of environment, an agent can take different forms. Autonomous robots are agents that inhabit the physical world; computer-animated characters inhabit simulated 3-D worlds; while software agents inhabit the world of computer

Previous Suggestions:

Computational Theories of Interaction and Agency
Computation and Embodied Agency

Cluster: Hellas

HR-Net 85% IFA: 0.72

formation vital to
he Internet. Such
n the international
ghbors, important
international
agenda transmitted by
Cyprus. . Until now,
tions had no way to
rmation, while access
s been sporadic, at

Previous Suggesti

HELLAS Home P
HR-Net Personal N
Alexander

Cluster: Java

JavaWorld - October - Java Q&A -- Java Still Open IFA: 0.81

```
ID: 000001
Creation Date: Thu Jan 30 14:16:30 1997

agent -2.89037
alexandro 0.405465
artifici -2.30259
assist 1.79176
athen -0.405465
```

Figure 3 - 3

A user's digest with recommended sites.

On the lower left hand side there is the rating window (the higher the number, the more interesting was the recommendation) and on the lower right hand side a visualization of the Information Filtering Agents.
Each entry of the digest contains the URL of the recommended site, its title, the first 300 bytes of its contents as well as the IFA and the IDA that are suggesting this item and the confidence of the suggestion. On the right half of the browser, the system displays other sites (WWW pages) that were recommended by the same IFA at previous digests.

The user feedback is used by the system to allocate credit to the information filtering agents that were responsible for selecting that document. A seven-scale feedback mechanism is available to the user. A rating of 1 means that the suggested site was very bad while a rating of 7 means that the site was excellent. If the user chooses not to use the explicit form of feedback, the system tries to infer the "likability" of a page in an indirect way. A JavaScript process running in the browser checks how much time the user actually spends following a link, how big the contents of the link was, how much time the user's machine was idle, how much time the browser window was in focus, and computes an indirect feedback rating. The basic idea behind this mechanism is that if the user doesn't like a specific site she won't stay that long in there; the rest of the parameters (like for instance the idle time of the computer) are a way to check for possible caveats, like the user going for lunch directly after she entered a bad web page. Finally the user can also pick a number of keywords from the document vector that best describe the given document and the weights of these keywords will be reinforced.

The user has the option to get more information on a specific recommendation by clicking the "Details" button in the digest. That provides her with additional data like previous suggestions by the same Information Filtering Agent, other suggestions that had lower scores and didn't make it to the digest, etc.

3.5 Bootstrapping Amalthaea

Amalthaea is bootstrapped by the generation of a number of information filtering agents and an equal number of discovery agents. This first generation of information filtering agents has to be somewhat relevant to the user's interests. This can be done in one of the following ways:

- The user submits a list of his favorite bookmarks or documents. This is usually the bookmarks list. Each of the sites in the list is examined and for each site an Information Filtering Agent is created (for more information on this process refer to Section 4.4).
- Amalthaea checks the user's browser history files. Many browsers (like Netscape for instance) keep a history file (often several MBytes large) that contains all the URLs that the user has visited[1]. The system analyses that information and tries to infer certain patterns in order to decide if it has to monitor any sites.
- Users can point Amalthaea at a specific page (possibly while they are browsing) and request the generation of more agents that will find similar information. In that case, the contents of the page is retrieved and a new Information Filtering Agent is created based on it.

1. The history file is also helpful in the case that Amalthaea recommends sites: if the user has already been at a specific site, Amalthaea never recommends that site again, unless it has changed since the last user visit. The system does a CRC check of all those files to ensure this.

- Finally users can select pre-trained "packages" of agents (each package focuses on a particular topic; for instance "European soccer", "Greece", "Agents research" etc.) This method speeds up the learning curve of the system significantly.

The above methods explain the generation of Information Filtering Agents. Information Discovery Agents are generated by random assignment of WWW indexing engines to each agent. Each Information Discovery Agent also contains information on the amount of keywords provided to the discovery agents by the filtering agents that will actually be used in the queries and how those queries will be formulated in terms of logical operators between the keywords (for instance will a query be formed as "computers AND F1" or "computers OR F1" etc.) Different search engines have different ways of formulating queries and allow for different operators on them (AND, OR, NEARTO etc). Depending on the engine assigned to each Discovery Agent, the correct set operators is used.

3.6 Visualization of the User Profile

In order for a user to develop a sense of "trust" for the system she must understand when the agent system initiatives and more important why it performed a certain action. The latter is usually accomplished by having the user inspect the state of the system and the set of conditions that let it to that action. While it is quite easy to do something like that in a centralized top-down system (like a rule-based system or a memory-based reasoning system) this task is very difficult for a system like Amalthaea. A common problem of multiagent systems (and other bottom-up or complex systems, for instance neural networks) is their lack of ability to easily express their state to users. This is mainly the case because their behavior is not a result of a set of rules but the outcome of the dynamic interaction between the different agents. In Amalthaea we are dealing with this problem by visualizing the populations of agents in 2D space. The (x,y) coordinates of each agent are calculated by the following formulae:

$$x_j = \sum_{i=0}^{M} r_j \cdot \cos\left(\frac{2\pi}{M} \cdot i\right) \tag{1}$$

$$y_j = \sum_{i=0}^{M} r_j \cdot \sin\left(\frac{2\pi}{M} \cdot i\right) \tag{2}$$

Where j denotes the Information Filtering Agent, i the keywords in the vector and M is the number of keywords in `vectorfile`. The variable r_j is defined as:

$$r_j = \sum_{i=0}^{M} w_{ji} \tag{3}$$

where w_{ji} is the weight of the i^{th} keyword of the j^{th} Information Filtering Agent.

The user is be able to browse through their personal information landscape and see what the system thinks their interests are. When this visualization window initially is brought up, basic clusters of agents are seen, each one labeled using the most prevailing word shared between their elements (that is, agents). So for instance a landscape might contain clusters labeled "Greece", "Agents", "F1 motor racing". When the user does a "close-up" of a particular cluster she sees more details about the cluster, including individual cluster elements or Information Filtering Agents.

In a system like Amalthaea trust can be developed at different system levels, from the micro (single agent) to the macro (whole system) and in between the two at the cluster level. Our aim is for the users to develop trust at the cluster level: a user must be able to identify distinct clusters of her behavior and understand if those clusters make relevant and consistent recommendations. We believe that developing a sense of trust at the single agent level is difficult and not sufficient (thought the user can see other links that that a specific agent has retrieved and how they were rated, but there are too many agent in the system and identification by a single ID number is not very helpful). Trust at the whole system level is developed easier if the user trusts the individual clusters (although this is not a necessary and sufficient condition)

Sometimes the dynamics of the system are more important than a "snap-shot:" of its state at any given time. The last component of the visualization applet enables the user to see how the system's perception of her has changed over time. Amalthaea saves snapshots of its state at certain time intervals. The user can "play-back" those user interests space snapshots in the form of a movie and observe what the initial "bootstrap conditions" of the systems where, which clusters grew larger over times, which shrunk, what the future trends will be. Other efforts to visualize information retrieval results include the method introduced by (Bartell et al., 1994) which organizes retrieval term and phrase weights in the perimeter of a circle.

3.7 Client - Server Communication

The AUI is using standard socket communication to the Amalthaea server. When the Java-based AUI starts running, it creates a socket connection to the server and sends (in an encrypted form) the ID and the password of the user. The server responds by sending back the information it has collected for that user since the last time he logged on, and the AUI presents is accordingly.

Amalthaea supports users with both continuous and dial-up connections to the network. All the modules of the system are aware of this distinction and act accordingly. For instance if a site that has been monitored changes when the user is logged on to the system, this information is presented in a popup window. If he is not logged on then the information is either stored for the first time that he connects to the system or is sent to him via email. The same is true for the presentation of the suggested sites. They can either be presented and updated continuously or in the form of ``digests'' presented to the user each time he logs into the system. For users with a direct and continuous connection to the Internet Amalthaea follows a ``background running approach''. That is, the AUI is running in a corner of the user's screen and is continuously displaying and updating information. If the user is interested in that information he acts accordingly; if not he just ignores the system.

Chapter 4 Architecture

Chapter four describes the system in detail and the relationships between its sub-components: the document discovery engine, information filtering agents, information discovery agents, the interactions between them and the economic model methods used. The chapter concludes with a synopsis of credit assignment and information flow in Amalthaea.

4.1 Overview

Amalthaea's architecture assigns to each user her own Information Filtering and Information Discovery Agents and generates a closed ecosystem. More that one persons can use the system but all their files are separate; no interaction between different users is taking place in the system. All the components of the system that will be discussed in this chapter operate on a single user. For handling multiple users the system just uses multiple instances of those components.

As mentioned in Section 1.1.1 Amalthaea is composed of the following parts (illustrated in Figure 4 - 1):

- The user interface (described in the previous chapter), where the user is presented with the retrieved information and gives feedback on its relevance.
- Two distinct types of agents: information filtering agents and information discovery agents, and mechanisms that support the credit allocation and the evolution
- The engine for retrieving documents from the WWW.
- The engine that processes the documents and performs stemming and weighting operations in order to generate the keyword vectors.
- A database of the retrieved documents URLs.

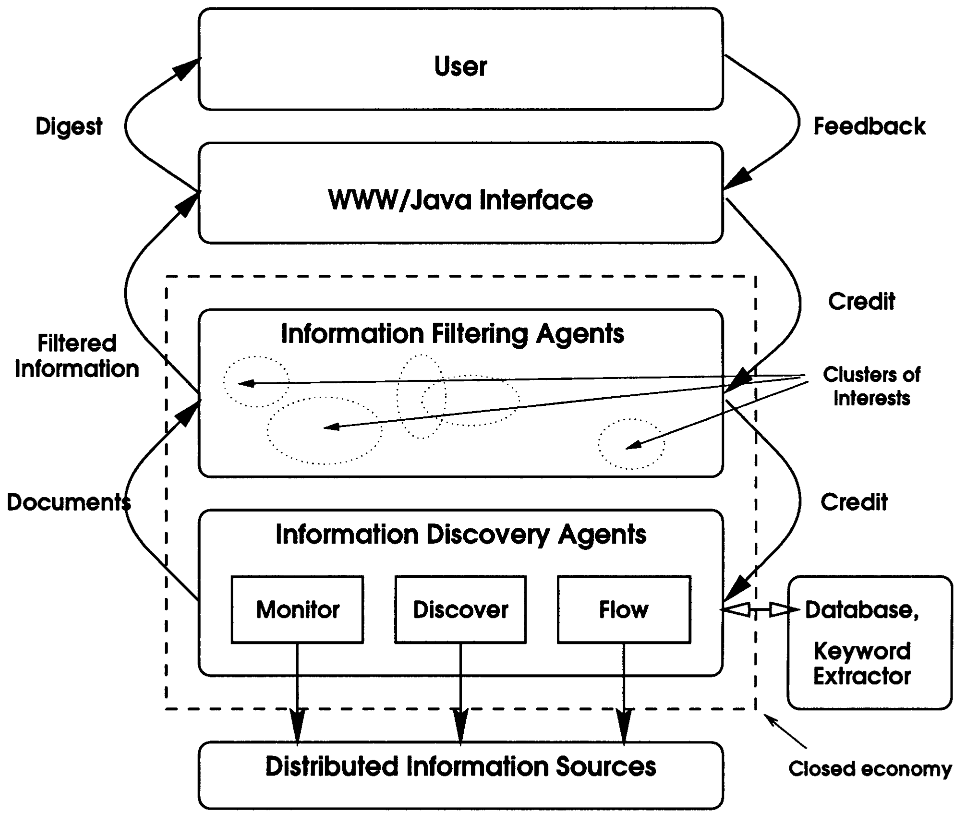


Figure 4 - 1

An overview of the architecture

The user is presented with a digest of sites by Amalthaea. She then provides feedback by rating the sites included in the digest. Based on the user's ratings, credit is assigned to the related filtering and discovery agents.

4.2 WWW documents and and Representation

All the sources of information[1] for the system can be accessed through the World-Wide-Web (http, ftp, news, gopher connections). The initial engine for document retrieval was based on the WWW Organization's `libwww` library (Frystyk and Lie, 1994). However, Amalthaea's latest version of the engine is a Java application. Although there is a slight performance loss, we found that the Java language's handling of outside connections is much more efficient and portable. On top of this engine, a library is built for normalizing URLs before they are stored in the "already retrieved" database.

1. By "information" we mean documents from various sources like WWW pages, gopher sites, ftp sites, WWW-based information services, newsgroups etc.

All retrieved URLs are kept in a database in the form of weighted keyword vectors. They are accompanied by a HTML checksum that enables the discovery agents to know if the specific URL has changed since last visited, in case it is a monitored site. A separate list of the links contained in each URL is generated (up to a user-defined depth, currently two) to assist the monitoring of certain sites that act as "starting points" in Internet exploration (for instance a list of all the agents-related sites). Finally the database is used for keeping track of the URLs already included in previous digests to prevent presentation of duplicate information to the user.

Amalthaea's internal representation of documents is based on a standard information retrieval technique called weighted vector representation (Salton and Buckley, 1987). The basic representation of the Information Filtering Agents and the parsed HTML files is the weighted keyword vector. When the HTML files are processed, a clear-text version is generated, the text is decomposed into its keywords, which are weighted and compose the keyword vector (as explained below). Document similarity and IFA selection of documents is based on the weighted keyword vector operation. After the HTML source code of a given URL is retrieved, a parser application removes the markup language tags, indentifies the title of the document and the hypertext links it contains and saves those data for later processing.

The original text is consequently processed in order to identify and rank the important words it contains. The following is an example of the evolution of a sentence through the different parts of the vector generator:

- Original Text:
  ```
  Agents are running objects with an attitude
  ```
- Words like "the", "it", "he", "will", which are the most common in the English language are removed from the text. After stop-word removal the sentence becomes:
  ```
  Agents running objects attitude
  ```
- The remaining words are stemmed, their suffixes are removed, leaving just the roots. The stemmer algorithm used here is a modified version of that introduced by Porter (see Porter, 1980):
  ```
  Agent run object attitud
  ```

The keywords that survive the stemming process are recorded in the form of an Mx2 matrix, where M is the number of the keywords. The first column of the matrix stores the keywords in alphabetical order and the second column stores their frequency in the document. The keywords of the title of the submitted WWW page and the URL of the page are assigned a weight equal to 0.05*M[2], and the URLs found in the HTML document are given a weight equal to half their frequency in the document.

2. A set of iterated experiments revealed that this is the optimal value when processing a pool HTML documents

Finally, each keyword is weighted by producing its "tfidf" measure. The "tfidf" acronym stands for term frequency times inverse document frequency and it is a standard information retrieval weighting mechanism:

$$W_k = H_c \cdot T_f \cdot idf_k \tag{4}$$

where T_f is the frequency of the keyword in the current document (*term frequency*), the H_c is the header constant, and idf_k is formally defined as:

$$idf_k = \log\left(\frac{N}{df_k}\right) \tag{5}$$

N is the total number of documents that have been already retrieved by the system and df_k is the document frequency of k. The term idf_k is the frequency of the keyword in the whole collection of documents (*document frequency*).

In this case the collection of documents is the set of all weighted keyword vectors, which are the internal representation of the retrieved documents. The header constant equals 1.0 if the keyword was found in the document's body text. If on the other hand the keyword was part of the title, its weight is multiplied by a constant greater than 1.0. In this way title keywords have more weight than plain body keywords.The process of the creation of the vector is completed by augmenting the weighted vector with some additional fields like the canonical URL of the document and the server and sometimes the author. Using the above method, all WWW documents are represented in a multidimentional space.

4.3 Evolution in the Multiagent System

The evolution of the agents is controlled by two elements: their individual fitness and the overall fitness of the system. Fitness measures of agents are described in Section 4.6. Only a variable number of the top ranked (the best performers) of the whole population is allowed to produce offspring. The rank of an agent is based solely on its fitness. The number of the agents that will be allowed to produce offspring is linearly related to the number of agents that will be purged because of poor performance (low fitness). These numbers are not constant and are related to the overall fitness of the system. If the overall fitness is diminishing then the evolution rate is increased in search for quicker adaptation to the user's new interests. If the overall fitness is increasing the evolution is kept at a lower rate to allow the system to slowly explore the search space for better solutions.

New agents are created by copying (or cloning), crossover or mutation (see Mitchell, 1996). All operators are applied to the evolvable part of the agents, the *genotype*. The other part of the agents, the *phenotype* contains information that should not be evolved, usually instructions on how to handle the evolvable part. The copying operator takes the best performing agents and creates more agents like them. The two point crossover operator, given two agents returns two new agents that inherit a part of the keyword vectors of the parents. This operator randomly selects two points in the keyword vector and exchanges all fields of the two parents that lie between these points creating two new agents.

For the sake of this project the distinction between the genotypes and phenotypes (as discussed in the next sections) is quite narrow. Usually a more complex development process occurs for deriving the phenotype from the genotype (Belew, 1989). In our case there is no individual learning or adaptation of individual agents, resulting in an one-point phenotypic search space.

Given two genotypes G_1 and G_2 the two point crossover operator is formally defined as:

$$G_1 \otimes G_2 \rightarrow G_3, G_4 \quad (6)$$

The algorithm that returns the two crossover points p_1 and p_2 works as follows:

$$p_1 = rand(1, sizeof(G) - 2)) \quad (7)$$

$$p_2 = rand(p_1, sizeof(G) - 1)) \quad (8)$$

The new genotypes inherit a part of the keyword vectors of their parents that lies between those numbers. The variable *i* represents the weighted keyword.

$$G_3 = \begin{matrix} G_{1_i} & 0 < i < p_1 and p_2 < i \le sizeof(G) - 1 \\ G_{2_i} & p_1 \le i \le p_2 \end{matrix} \quad (9)$$

$$G_4 = \begin{matrix} G_{1_i} & p_1 \le i \le p_2 \\ G_{2_i} & 0 < i < p_1 and p_2 < i \le sizeof(G) - 1 \end{matrix} \quad (10)$$

Mutation is another operator that can be used either on its own, or in conjunction with cloning and crossover. The mutation operator takes the genotype of an agent as argument and creates a new agent that is a randomly modified version of its parent. The weights of the mutated keywords are modified randomly while the new "mutated" keyword is a randomly selected keyword from an agent that belongs to another cluster or from a recently-retrieved highly-rated document.

4.4 Information Filtering Agents (IFAs)

An information filtering agent is based on an augmented keyword vector (which is the major part of its genotype). Keyword vectors are used to assess the similarity of two documents as well as the match between an information filtering agent and a particular document. These vectors are augmented with other information such as the author of the document (if possible) and whether it was created by the user explicitly or not. If it was, then the long-term interest field (a boolean parameter) is activated indicating that the documents proposed by this agent are treated more favorably.

The genotype of the information filtering agents is essentially a weighted keyword vector. The phenotype of these agents contains the non-evolvable part of the agent like its fitness, the long-term interest field and of course the commands that enable the agents to exchange information with each other and the system[3]. Essentially the phenotype resembles a fixed template that is "filled" with the genotype information and then "executed". Figure 4 - 2 visualizes the relation between the genotype and the phenotype of the information filtering agents:

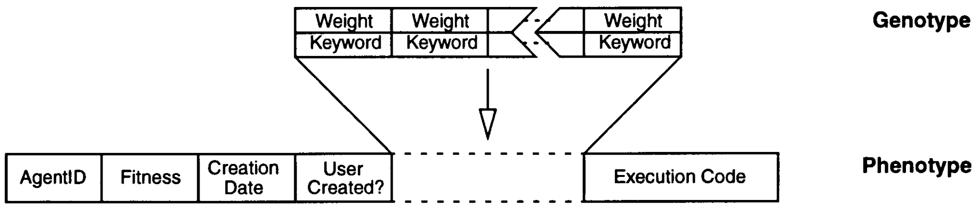


Figure 4 - 2

The information filtering agent genotype and phenotype

The information filtering agents are presented some documents by the information discovery agents (described later on.) The former act as "masks" that allow only the documents that are close to their weighted keyword vectors to pass through. Each filtering agent selects one document that is closest to its vector and calculates how confident it is that the specific document will interest the user. In this decentralized approach each agent believes that it is a perfect model of part of the user's interests, so if a document matches its vector completely, the agent's confidence is `1.0`.

In order for an Information Filtering Agent to assess its similarity to a given WWW page, it has to compare its genotype to the vector representation of the text inside that page. As mentioned above, each document is represented by a multi-dimensional vector (each vector can have different a dimension). In order to make the document comparison and clustering feasible we project each vector in the multidimensional space of the hypervector `vectorfile` that includes all the keywords found in the whole collection of the Information Filtering Agents. Their dimensionality is expanded by adding to them all the additional keywords of `vectorfile` and setting their weight to zero.

3. This exchange of information is simple and should not be considered communication between agent entities in a higher cognitive level.

When two documents are compared for similarity, we evaluate the cosine of the angle between the vectors representing the two Information Filtering Agents. This is done by evaluating the dot product of the two vectors and dividing it by the product of their magnitudes. The formula that returns the distance between two keyword weighed vectors *a* and *b* is the following:

$$D_{IFA_{a,b}} = \frac{\sum_{k=1}^{j} w_{ak} \cdot w_{bk}}{\sqrt{\sum_{k=1}^{j} (w_{ak})^2 \cdot \sum_{k=1}^{j} (w_{bk})^2}} \qquad (11)$$

As mentioned in the previous section, not all documents introduced by filtering agents make it into the digest. The system decides if the agent is going to present something to the user by ranking the proposed documents using the following formula (confidence level):

$$C_i = D_{IFA} \cdot F_i \qquad (12)$$

where *i* is the document number, and *F* is the fitness of the filtering agent that proposed the document. The top *n* documents are selected from the ranked list, where *n* is a user-definable number that indicates the amount of items that the user is interested in including in the digest.

The file format of the Information Filtering Agents is text-based. Each IFA has a unique ID, a creation date, a fitness, a genotype that contains the weighted keywords, the coordinates of that agent in a 2D space (as discussed in Chapter 3) and the number of times this agent has been invoked, presented something to the user and the user gave feedback. The format of the files is the following:

```
***Amalthaea-IFA***AgentID
000001
***Amaltahea-IFA***Creation-Date
March 25, 1996 14:45 EST
***Amaltahea-IFA***Fitness
578.69
***Amaltahea-IFA***Keywords
Smokey 1.823
Bear 2.357
***Amaltahea-IFA***Coords
x,y
***Amaltahea-IFA***Invoked
13
***Amaltahea-IFA***Present
7
***Amaltahea-IFA***WithFeedback
6
***Amaltahea-IFA***End
```

4.5 Information Discovery Agents (IDAs)

Each information filtering agent issues "requests" (and standing orders, that is long-term commands) to information discovery agents about the type of documents they are interested in finding. Discovery agents select which contract they want to take on.

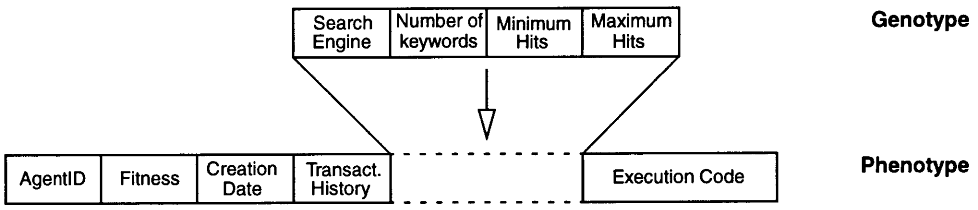


Figure 4 - 3 **The information discovery agent genotype and phenotype**

An information discovery agent is based on a genotype that contains information on the keywords it should utilize when querying the WWW indexing engines, along with the canonical URL of the engine (or information source) that it contacts. The aim here is to create a diverse body of agents that will allow different types of documents to be discovered through different search engines. The information discovery agents that are monitoring sites operate a bit differently than discovery agents specializing in WWW and those specializing in steady information flow. They are not using any search engine as mediator but instead visit directly the site of interest and analyze the document to discover if it has changed and by how much, using the database where the URLs are stored.

Distinct characteristics of information discovery agents include that they search alternative information sources in remote computers and that most are parasitic (in the sense that they are utilizing existing WWW search engines to find information and not dig it up on their own, a type of meta-search).The IDAs do not receive credits directly from the user but indirectly, from the information filtering agents that "employ" them.

The Information Discovery Agents are responsible for spawning the spiders that post queries to various Internet Search Engines, collect the results and present them to the Information Filtering Agents that requested them. The number of spiders that concurrently post and process the queries is configurable by the user. In our current implementation we are running 64 spiders at any given time.

Depending on the search engine it uses, each spider uses different query types. In this case a querytype "1" means AND-ing all the keywords together, "2" ORing them etc. Not all engines have the same keyword combination abilities (for instance Altavista has a "NEAR TO" query operator, unlike Lycos etc.) The spider forms a query according to its parameters and appends each engine's wild-character at the end of the stemmed words. If the number of hits returned from that query is greater than the MaxHits or less than the MinHits parameter then the spider re-posts the query but with more or fewer keywords until it reaches the level of hits requested by the Discovery Agent. If a link is already visited by the user or already proposed to him or her, it is not considered again. For each of the remaining links, the spider fetches its HTML contents, processes them and saves them into separate files for consideration by the related IFA later on.

The Information Discovery Agents have a unique ID number. The following of their parameters are part of their genotype and are evolved: the search engine they use when searching for relevant information, the number of keywords to submit in an initial query, the method they should use (boolean AND or OR, NEAR TO, etc) and the range of URLs to expect as an answer (that should be less than the MaxHits variable but greater than the MinHits). Some variables like the history of transactions with Information Filtering Agents and the fitness are not evolved. The format of the IDAs is presented in the next table:

```
***Amalthaea-IDA***AgentID
000001
***Amaltahea-IDA***Creation-Date
March 25, 1996 14:45 EST
***Amalthaea-IDA***EngineID
4
***Amaltahea-IDA***Fitness
578.69
***Amaltahea-IDA***NumberOfKeywords
15
***Amaltahea-IDA***CombiningMethod
4
***Amaltahea-IDA***MinHits
50
***Amaltahea-IDA***MaxHits
235
***Amaltahea-IDA***History
000014,300,15
000345,790,3
000893,213,4
***Amaltahea-IDA***End
```

The history of transactions is a table of finite size; each record has the form of (IFA_ID, Total Credit, number of transactions).

IFA_ID (int)	Credit (float)	Transactions (int)
000005	763	4
000123	501	5

The Information Discovery Agents select which Information Filtering Agents' requests to fulfill. From an implementation point of view this is happening in the following way: All Information Filtering Agents' requests are placed on a table. When an Information Discovery Agent selects a request, that request is erased from the table. The Information Discovery Agents refer to their history logs and check if the Information Filtering Agents which has been the most profitable in doing business with has posted any requests. If no, it proceeds to the next preferred Filtering Agent and checks again. If yes, that request is selected. Then the system proceeds to the next Information Discovery Agent, until all Information Filtering Agents' requests are fulfilled (so usually an IDA serves more than one IFA). The Information Discovery Agents use the above-mentioned method for 80% of the time. The 20% of the time the selection is random in order for the system to explore its search space and identify potential new interesting matches. The Information Discovery Agent that gets to pick an Information Filtering Agent first is selected in three different ways: randomly, best-fitness first and worst-fitness first. Each of those methods has their advantages and disadvantages.

4.6 The Ecosystem

The interactions between filtering agents and discovery agents, as well as among themselves control the global behavior of the system. The technique is inspired by an approach called "Market-Based Control". As Clearwater put it "Market-Based Control is a paradigm for controlling complex systems that otherwise would be very difficult to control, maintain or expand" (Clearwater, 1996). Our form of control views the system as a miniature economy. We are trying to yield desirable global behavior in a complex system on the basis of agents acting on local information.

The agents that compose the ecosystem operate under a penalty/reward strategy, supported by the notion of "credit" that is assigned indirectly by the user based on the system's performance. The user is giving feedback on the suitability of an item in the digest. The system relates this feedback to the filtering agent that proposed the item and the discovery agent that retrieved it and assigns the credit. Credit serves as the fitness function in both populations which are evolved separately. The higher the fitness of an agent, the more chances it gets to survive and produce offspring.

If the user feedback is positive then the information filtering agent that proposed the item is awarded an amount of credit directly related to its proposal's confidence level. If an agent is confident that the user would like the item it proposes then it receives positive credit, but not as much as when it would be very confident. If on the other hand it is very confident that the document would be of interest and the user's feedback is negative then it receives a lot of negative credit, which is bad for its fitness. Information filtering agents "pay" a fixed percentage of the amount of the credits they receive to the information discovery agents whose outputs they used.

It is evident that not all filtering agents are able to present something at each digest. The documents proposed by the agents are ranked by confidence level and the top are selected and presented to the user. Although the agents that present items are not always the same, they usually represent the top 40% of the population, fitness-wise. The rest of the population is there for diversity purposes. One would notice that if an information filtering agent doesn't present anything to the user then its credit would remain constant. In order to accelerate the destruction of non-competent agents and the evolution of new ones we introduced a linear decay function which can be seen as a type of "rent"

(Baclace, 1992). In order for the agents to inhabit the ecosystem they have to pay something. If the credits they gain exceed this "rent" then they live. Otherwise they are removed and new ones are created. Moreover, if two agents propose the same document then they receive a penalty in order to discourage that and increase the diversity of the population. The information filtering agents receive ratings directly from the user depending on their performance. They, in turn, assign some credit to the information discovery agents that helped them locate and retrieve the information rated by the user.

The Information Filtering Agents evolution has one particularity: the whole population is not evolved together; instead, Filtering Agents compete with each other inside a given cluster. So all the IFAs that belong to cluster "Greece" are evolved together, the "Computer Science" IFAs together, etc. This yielded better performance in terms of user feedback, merely because it is to the interest of the system to create niches of agents, and inside those niches to keep the best IFAs. The size of the clusters is a result of the overall feedback that the user has given to its members.

The number of agents that are purged in each system cycle is not fixed but instead it is based on the amount of positive feedback the user is giving to the system. If the overall negative feedback of the current digest is higher than the average of the past ten digests (a user-definable parameter) then the percentage of purged agents increases. If on the other hand there is more positive feedback this number decreases. Even in the best-case scenario of very positive feedback, there is still a percentage of the population that is purged in order to explore the search space for possible new user interests. A sample requests and information flow in Amalthaea goes as follows:

- The filtering agents send requests for documents to the discovery agents in the form of a keyword vector.
- The discovery agents select a filtering agent's request and based on their phenotype they try to retrieve relevant documents utilizing the WWW indexing engine they are assigned to.
- Each discovery agent presents to its filtering agent the set of retrieved documents and the filtering agent selects the ones that best match its keyword vector.
- A number of filtering agents include their selected documents in the user's digest based on their fitness confidence in the documents.
- The user rates the documents presented in the digest, thus giving feedback to the system.

4.7 Interaction between IFAs and IDAs

The overall system behavior depends a lot on the way the filtering and the discovery agents interact. As mentioned before, the filtering agents do not impose commands on the discovery agents. Instead, they post requests to the whole population of discovery agents. Then, individual discovery agents choose one of those requests based on their experience. The phenotype of the discovery agents includes a history of transactions with different filtering agents and how well the filtering agents performed in those transactions (that is whether the user gave positive feedback to the items discovered, filtered and presented as a result of that transaction/cooperation). Depending on this history an IDA creates a credit-sorted ranked list of all the filtering agents they worked for and they try to get "contracts" from the highest ranked ones to maximize their own fitness. This way the discovery agents figure out by themselves which filtering agents they best serve, based on the WWW indexing engine they use and the combination of the keywords they query. Since the number of the discovery agents is smaller than the number of the filtering agents, after a discovery agent fulfills the request of a filtering agent, it goes on and selects the next request, if available. In order to achieve a better exploration vs. exploitation ratio (match up different filtering and discovery agents) 20% of the filtering agents requests are served by more than one discovery agent.

Since some filtering agents are closer to the user interests than others, they tend to gather more credit in a typical document presentation. It is in the interest of the discovery agents to select the filtering agent that will bring them the most credit. This is an interesting issue: in what order do discovery agents pick up a filtering agent request? Several different methods exist: one could try ranked-based selection. That is, the discovery agent with the most credit selects first the request from the filtering agent of its choice. The second method is that of random selection: a random discovery agent is picked and allowed to select a request; the process continues until all discovery agents are selected and then restarts. The third method is that of inverse-rank selection: to enforce diversity and keep a balance in the population, the discovery agent with the least credit is selected and is allowed to pick a request first with its favorite filtering agent.

Chapter 5 Implementation Issues

This chapter explains the implementation techniques used in detail. It introduces the different sections of Amalthaea and their input and output mecha nisms from a software engineering point of view. Furthermore, it discusses certain security and privacy considerations on the design of the system. We try to reveal possible privacy attack methods, and offer suggestions of different encryption-based and architecture-based solutions.

5.1 Overview

Amalthaea is written in ANSI C++ and Java. Versions of the code are running in standard Unix boxes using GNU's g++ compiler and SunSoft's Java Software Development Kit (SDK). Our latest implementation is compiled using Microsoft Visual C++ 5.0 and Microsoft Visual J++ 1.1 on a Windows NT 4.0 system. We decided to switch to the Windows environment because of the much better multithreaded support of the Java virtual machine on a multi-processor system, like the dual PPro we were using for our server.

The C++ part of Amalthaea deals primarily with handling the various parts of the agent ecosystem: the Information Filtering and Discovery Agents are generated and their populations are being evolved; the credit-distribution mechanisms and the management of the simple economy/ecosystem; the text-processing and the vector generation routines. In general the code that needed to be fast and efficient is written in C++.

The first Java segment (which is a Java stand-alone application, not an applet) of the code deals chiefly with network-access related tasks: the multithreaded spiders that open URL connections and fetch information; the parasite spiders that pose queries to the major search engines; the HTML parsing routines that extract title information, internal and external links. The second Java segment is an applet that is fetched by the user's browser when he/she reads the digest or re-configures the system. The main advantage of Java applets in the design of Amalthaea is the portability that this language offers through the virtual machine.

5.2 WWW Server, Java and CGI Scripts design

The Amalthaea HTML server backbone is based on Microsoft's IIS server and a combination of C++ and Java script files that handle user interaction. When a user opens the main Amalthaea homepage she has to create an account, or if she already has one, log into the system. The first of the scripts authenticates the user and the password by comparing the provided data with those stored in Amalthaea's database. Provided that those are correct the user can select her course of action from a variety of options, like digest presentation, new agent creation, system configuration etc.

Most of the scripts work and provide services to the user in real time. However, a few scripts generate overview descriptions of the Filtering and Discovery Agents operations so that the user can view those even in slow connectivity situations. For instance, one of the scripts summarizes the actions and the contents the Information Filtering and Discovery Agents in a single compressed file, so that it can be downloaded to an applet running on a remote computer even if the user has a slow network connection. This file contains a compact version of the Information Filtering and Discovery Agents landscape.

5.3 Typical Operation Cycle

Amalthaea saves the various user-specific files (like Information Filtering and Discovery Agents, incoming HTML files, text vectors) in text format so it can be easily integrated into different agent applications (British Telecom is currently integrating Amalthaea into one of their prototypes, Radar (Crabtree, 1997). A description of how different modules of the system interact and which files each uses, along with a typical operation cycle of the system follows:

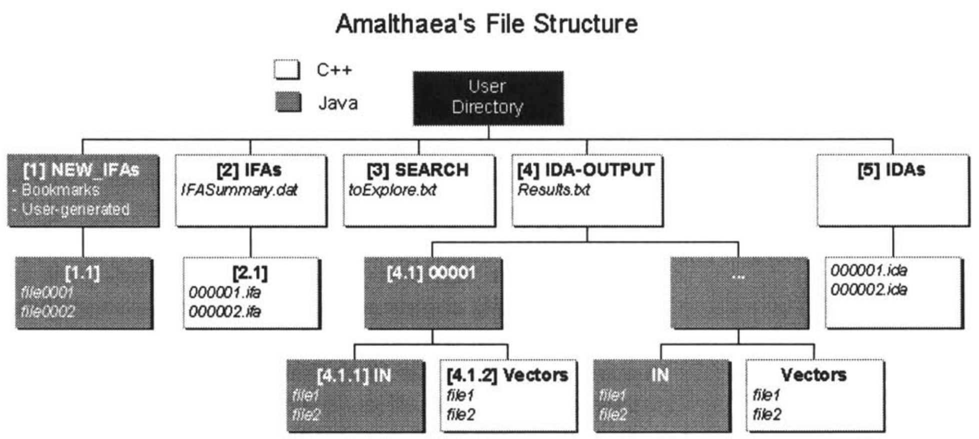


Figure 5 - 1

A typical user filestructure. The gray directories are created/processed by the C++ modules, the white ones by the Java modules.

- Check with the interface if user requests to create new agents (either while bootstrapping the system or by explicitly asking so). If yes, get a list of the URLs, fetch the files and place them in the directory NEW_IFA.
- Check the directory NEW_IFA [Figure 5-1, Box 1] to see if any files [Figure 5-1, Box 1.1] are there. If yes, create their keyword vector, generate new agents in directory IFAs [Figure 5-1, Box 2]. The new agent filenames should be in the form: 000001.ifa, 000002.ifa etc [Figure 5-1, Box 2.1]
- Create an array that contains the top keywords and weights of the selected IFAs. Using the history of the IDAs, select the appropriate IDA for each IFA. Note that since IDAs are fewer than IFAs, each IDA gets to select more than one IFA request. Save the results in a file toExplore.txt [Figure 5-1, Box 3].
- Read the file toExplore.txt and the IDAs directory. For each entry get keywords and the IDA ID and create a thread that goes to the related search engine, enters a query and fetches the URLs of the returned documents. For each URL create a thread and fetch its contents. Create under IDA_OUTPUT a directory with the agent's ID number and then one called IN and store the documents there (i.e. IDA_OUTPUT/000001/IN/file1.txt.) [Figure 5-1, Box 4.1.1]
- Read all the IDA_OUTPUT/.../IN directories, create the vector files of all the documents, store them in the VECTOR directory (ie IDA_OUTPUT/ 000001/VECTOR/file1.vec). [Figure 5-1, Box 4.1.2]
- Compare all vectors with their related IFAs and get the top matches (n=1 for now) into a file called Results.dat [Figure 5-1, Box 4].
- Read the file Results.txt [Figure 5-1, Box 4] and construct a digest in the appropriate directory. Digest also contains the first few lines of each document.
- Based on user feedback generate a feedback file (Feedback.dat) that contains the user's reactions to suggestions
- Read the file Feedback.dat, update the weights of all the agents.

5.4 File Formats

Amalthaea uses several different file templates to store its state. This section gives an overview of the most important of them along with possible bootstrapping variable values.

5.4.1 Vectorized HTML Files

After being fetched by a spider, the processed content of a URL is saved in the following format:

```
***Amalthaea***URL
http://www.media.mit.edu
***Amalthaea***Title
The MIT Media Laboratory
***Amalthaea***Body
```

```
Click here for People,Sponsors, Research,
Academic Programs, Information
***Amalthaea***URLs
http://agents.www.media.mit.edu/
http://....
ftp://....
***Amalthaea***Fitness
0
***Amalthaea***EOF
```

5.4.2 Web Discovery Files

The Information Discovery Agents choose which Information Filtering Agents requests to fulfill, and generate a file that the Amalthaea spiders use for actually fetching the information. That file contains the IFA and IDA IDs, and the keywords that need to be searched. All the other data needed to spawn the actual spiders that would query the search engines are provided by the IDA files (for instance, which engine, how many keywords etc).

```
***Amalthaea-ToSearch***Begin
IFA_ID,IDA-ID,Keyword1,Keyword2,...Keyword N
***Amalthaea-ToSearch***End
```

5.4.3 Results Files

For each Information Filtering Agent, the spiders save a number of files that they have been instructed to retrieve by the Information Discovery Agents. Each Information Filtering Agent selects the most prominent site and a new file is created that contains all the information needed for the generation of a new user digest. The top n files that best match the vector of the respective agent are selected. Their URLs are saved in a file along with a similarity quantity (between the IFA and the document). The resulting file is called Results.txt:

```
***Amalthaea-Results***Begin
IFA_ID,IDA_ID,URL,Title,Abstract,Similarity,Confidence,FitnessIFA,FitnessIDA
IFA_ID,IDA_ID,URL,Title,Abstract,Similarity,Confidence,FitnessIFA,FitnessIDA
***Amalthaea-Results***End
```

5.4.4 User Interface Related Files

The HTML code for the digest is created using the final results file that was described in the previous subsection. The user's digest HTML code is generated in such a way that it contains information on the actual user and details on the suggested items, so that proper credit can be assigned to the Information Filtering and Discovery Agents.

A "summary" file of all the Information Filtering Agents is generated when the user file is updated. This file is a collection of the most basic information of each IFA for visualization purposes. If a user requests a visual map of the state of the system, it is very difficult to use the raw data of the IFAs, mainly because of their size. The summary file contains just the Agent ID, the Creation Date, the Fitness, a few keywords and the x and y coordinates of the 2D representation of the agent. The `IFASummary.dat` file is generated in the following way:

- Read ifa files in the user's "IFAs" directory.

- For each file, create an IFAgentSpec object that stores information about each agent. Store objects in a vector.
- Enumerate through all IFAgentSpec objects and generate a summary file. Each entry in the summary file has the following format:

```
AgentID, Creation Date, Fitness, first 10 keywords, x-coord, y-coord
```

5.5 Issues on Security and Privacy

In general, in order for an agent to serve the user better it has to know aspects of the interests of the user that he or she might not be willing to share with other people, or even interests that the user is not even aware of. This situation brings forward several privacy issues in the design of agents that assist the users. We will discuss what those privacy issues are and how they relate to the current implementation of Amalthaea.

Amalthaea was designed in such a way so as to allow maximum design flexibility in terms of where its components run. Right now, the Java environment imposes certain limitations on the capabilities of the user interface that runs on the user's machine. Two of those limitations directly affect Amalthaea: The first one is Java security: a Java applet when downloaded from a remote machine can either write data to the local machine *or* communicate with the originator machine through the network but cannot do both. In our implementation we chose to have the AUI communicate with the Amalthaea server and store all the information there. The second is the speed of the language (or lack thereof): Java is interpreted and consequently quite slow. Although a number of improvements are being worked out by the designers (like compilation of the virtual machine code to the machine's native code) right now running the whole Amalthaea system under Java (so everything can run on the AUI on the user side) in infeasible speed-wise.

Ironicaly, the current security provisions of Java lower the security standards of Amalthaea: under the current implementation of the system all the user profiles (the Information Filtering agents) are stored in a centralized place, namely the Amalthaea server. This method encompasses several security problems that might affect the privacy of the user. Two different categories of security problems arise:

- *Network Security*: Sensitive information about the user may be passed through the network and can be easily obtained by a third party.

- *Server Security*: Although a case of compromised security on the server part (either from a malicious administrator or from an outside attack) is rarer, its consequences if it happens are much more severe.

There are several ways to address those problems in Amalthaea: they range from simple additions to the communication protocol to support encryption to complete re-distribution of the components of the system from the Amaltahea server to the user side.

5.5.1 Sockets Encryption Layer

A simple modification is to add an encryption layer in the socket communication model between the client and the server. This way, no clear-text information is going to be transmitted via an open channel.

5.5.2 One-way encryption of IFAs

When new filtering agents are generated, encrypt their genotypes using one-way encryption techniques. In this case, the vectors have to be encrypted as well before compared with the filtering agents. Even if the security of the system is compromised, decyphering the filtering agents will be extremelly difficult.

5.5.3 Place IFAs on the user side

This is a more drastical change in the architecture of the system. By moving the IFAs from a centralized server to a completely distributed system where each user will run his/her own IFA agents on their own machine the system doesn't have any more a single point of attack and security failure.

It is evident that the above modifications are not mutually exclusive; they can be used in parallel to provide maximum security. We are gradualy moving towards a more distributed version of the system by placing the IFAs in the Java applet running on the user's machine. Apart from the enhancement of security of the system, this change will assist the scalability of Amalthaea since the computational requirements of the server will not be as great as in the centralized version. Since Amalthaea's design is completely modular only minimal changes in the code are needed to implement this move. In addition to that, we are implementing encryption methods like the one mentioned above to enhacethe security between the Amalthaea server and local machines.

Chapter 6 Testing and Evaluation

This chapter is devoted to testing and evaluation. We introduce two main categories of evaluation techniques, virtual users and real users. Virtual users utilize the profiles of real users and their interests evolve over time. They allow us to perform many iterations on the same data set. Real user experiments are based on a set of seven people that used the system over a certain time period.

6.1 Testing and Evaluation Techniques

The experiments we conducted to validate the hypotheses presented in this thesis, were developed along two axes. One group of experiments focused on testing the ability of the system to evolve and stabilize into meaningfull equilibria positions and infering the optimal distribution of agents. For those experiments we used the notion of "virtual users", profiles created by real user interests that automatically tested and provided feedback to the system. The virtual users enabled us to iterativelly perform big scale lenghty experiments without the use of people.

The second axis which we worked along is that of testing the performance of the whole system with real users. We tested if the system could actually find useful information on the WWW and present it to its users. We used a group of seven people and used the data from their interaction with the system to measure quantities like the mean absolute error between an agent's suggestions and the user's feedback, its standard deviation, and the correlation coefficient between Amalthaea's predictions and actual user ratings.

One important aspect of the testing phase was to select a set of metrics for evaluating the performance of the system. Those metrics combine overall user satisfaction from using the system (as reported by the users themselves at the end of the experiments) as well as system-recorded user feedback for its recommendations.

6.2 Experiments with Virtual Users

In order to provide an objective and consistent evaluation of the system's performance we had to use fixed points of reference in the two external factors that were influencing the system's performance: the user interests and the results of the queries posted to the WWW indexing engines. We compiled several different user profiles with different interests each and we also collected a number of HTML pages. The user profiles had the form of a number of different keyword weighted vectors. We then compared the items created by the filtering agents in the digest with those profiles and provided positive or negative feedback based on their similarity. Depending on the feedback, the agents received an amount of credit that was added to their existing credit. The items that the agents were presenting to the user profiles were selected from a fixed collection of HTML documents arranged in different directories to resemble the different search engines. The use of those local documents provided us with a quick response time and more importantly with a constant reference frame for evaluating the system.

6.2.1 Virtual Profile Evolution over Time

The purpose of this set of experiments was to evaluate the ability of the system to reach stable equilibria conditions and adapt to slowly changing user interests. At the beginning of the experiments a random set of user interests was created. Those interests were not static, but they were changing at a rate of 5% per system iteration. We performed seven different sets of runs, each one consisting of five trials for averaging the randomness of the genetic operators. Figure 6 - 1 visualizes the mean value of fitness (or credit) of the agents for each set.

The only difference between the sets was the initial user profiles. The first curve (with the lowest fitness/credit) indicates the average fitness of all the agents in the system, while the second curve represents the average fitness of the agents that actually present documents in the user's digest). The two curves exhibit in general the same behavior, although (as expected) the curve that represents the fitness of all agents is smoother. The number of generations is that high because the initial user profiles used are random (regular users use their bookmark list to bootstrap the system.)

The results demonstrate that the system is able to converge starting from different (random) initial populations.

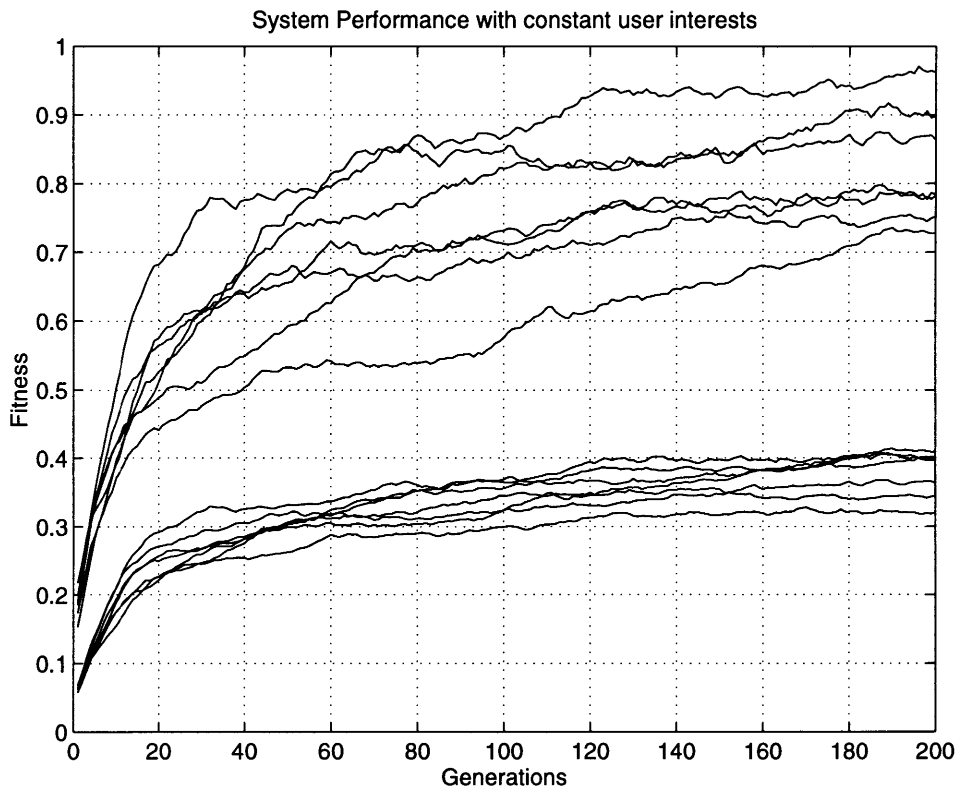


Figure 6 - 1 **The system is converging as the user interests change smoothly.**

The top group of lines represents the average fitness of the IFAs that presented digest items to the user; the bottom group represents the average fitness of all the IFAs. The system is able to increase its overall fitness regardless of the different starting points (user profiles.)

In the second set of experiments we introduced sudden changes in the interests of our virtual users. At random time intervals there was a bin change in the user profile. We configured the system in such a way as to alter the user's profile by a mean of 50% and a standard deviation of 15% (i.e. on the average change half of the user profile's interests randomly). These random changes were necessary in order to test if the system would be able to adapt in abrupt changes on its equilibrium position. As Figure 6 - 2 shows, the system is able to follow abrupt changes in the user's interests and after a sudden decrease of its fitness (because of the negative feedback it is receiving, since a big percentage of the user interests has changed) it is able to quickly adapt to the changes, by retrieving documents that interest the user more. Figure 6 - 3 shows the same technique applied to a different user profile.

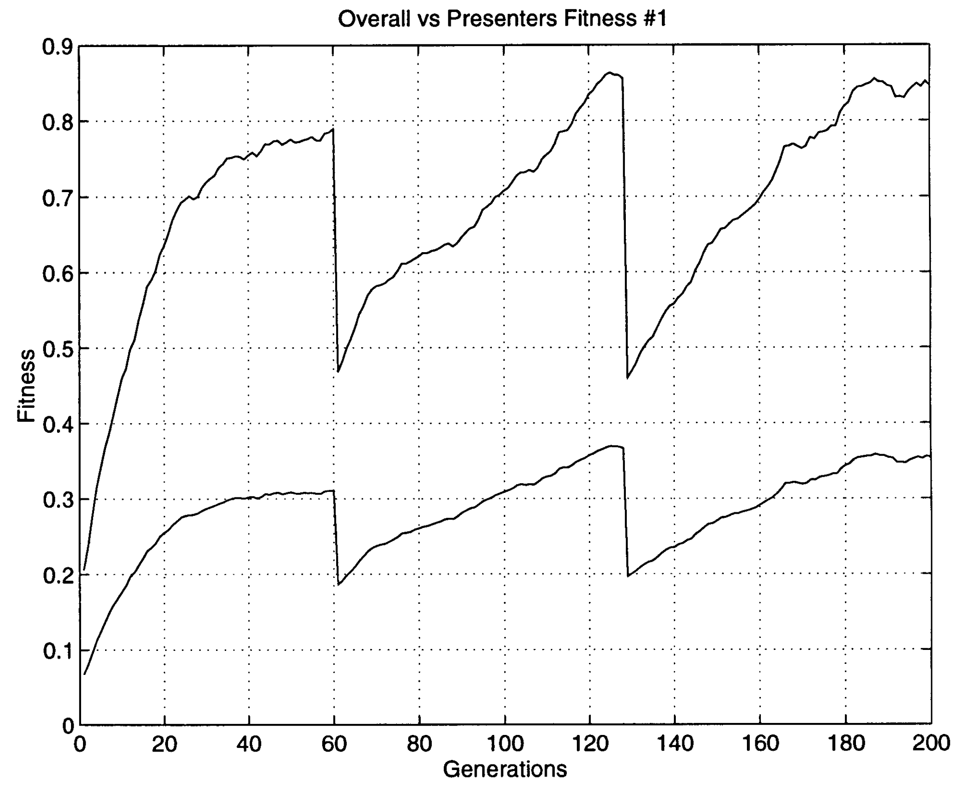


Figure 6 - 2

Changing suddenly the user interests: Evaluation of Amalthaea's performance for user profile one.

The top group of lines represents the average fitness of the IFAs that presented digest items to the user; the bottom group represents the average fitness of all the IFAs.

The third set of experiments involving virtual users had to do with the evaluation of the system's performance when the user was not giving feedback at each system iteration, but in a sporadic fashion. We have tried evolving the system for a number of steps between user feedback. The result when using a random interval with a mean of five steps can be seen in Figure 6 - 4. The fitness curves of the agents are not as smooth as in the previous figures as a result of the irregular user feedback. These results show that a user can have the system evolve more frequently even "on the background", without any direct feedback. The above three sets of experiments show that Amalthaea can learn the user interests and adapt to them over time as they evolve on a set of different user scenarios with different agents' configurations.

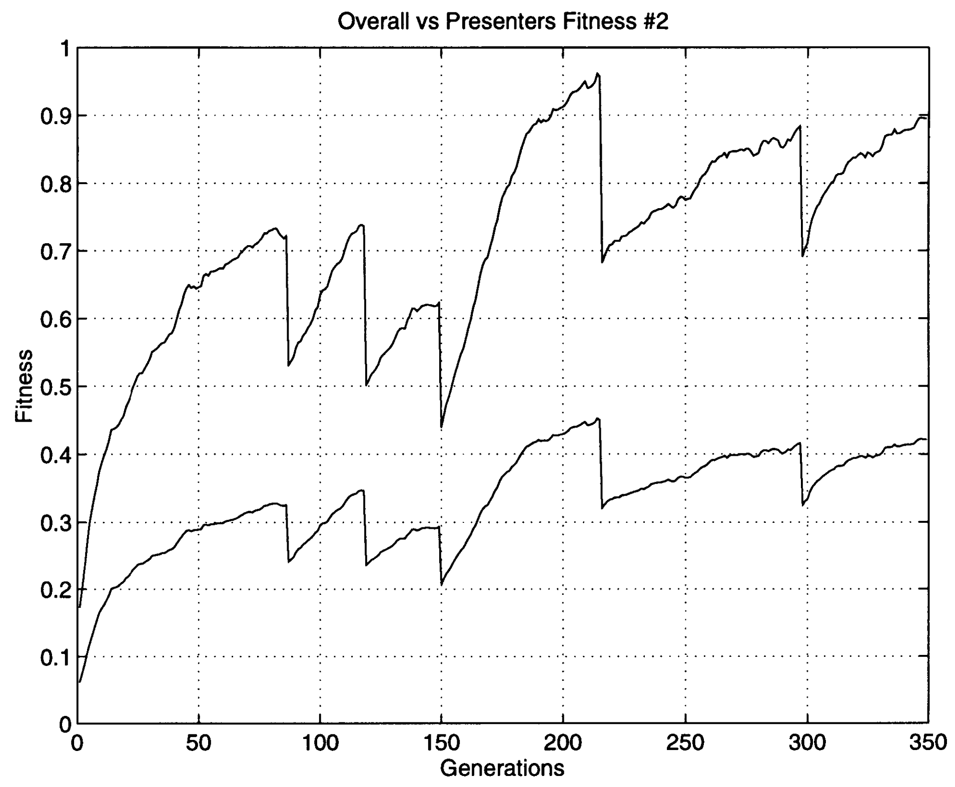


Figure 6 - 3

Changing suddenly the user interests: Evaluation of Amalthaea's performance for user profile two.

The top group of lines represents the average fitness of the IFAs that presented digest items to the user; the bottom group represents the average fitness of all the IFAs.

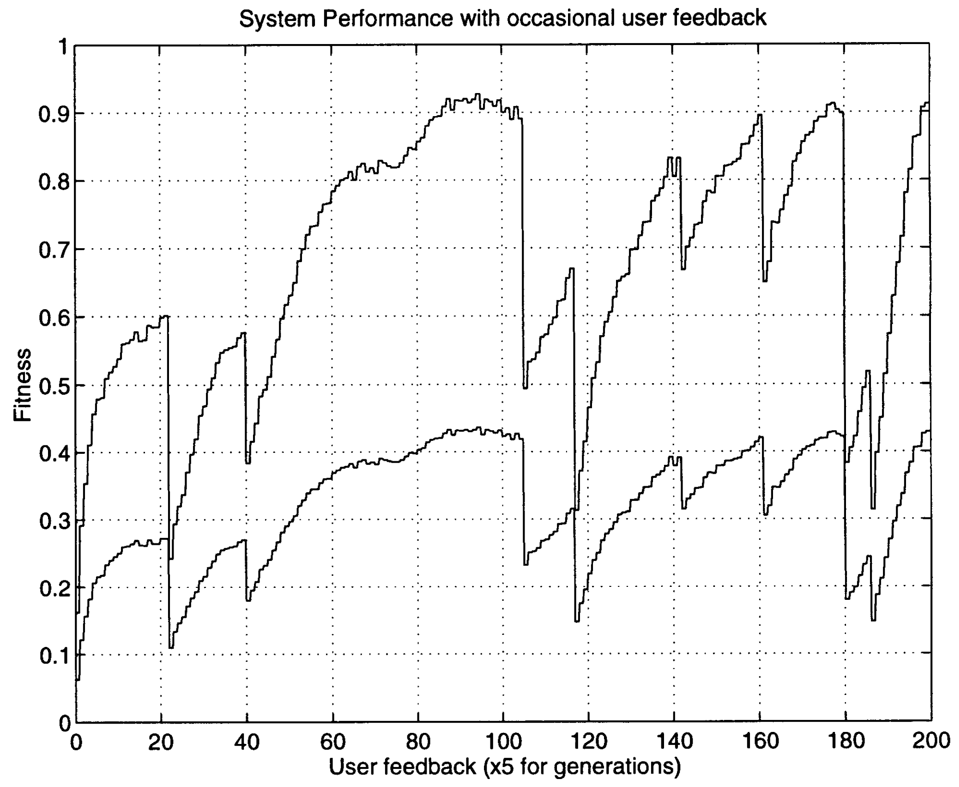


Figure 6 - 4 **Evaluation of system performance when the user is providing feedback at random time intervals with a mean of five generations.**

The top group of lines represents the average fitness of the IFAs that presented digest items to the user; the bottom group represents the average fitness of all the IFAs.

6.2.2 Ratio of Filtering Agents and User Interests

Extensive experiments have been performed on the relations of the number of filtering agents to the number of user interests and the fitness of the presenting filtering agents of the system. We tried to identify the different behaviors of the system when the ratio of user interests over filtering agents varies. We compiled different user profiles using data from actual users and then we clustered the generated filtering agents into groups by keyword vector similarity. The definition of an "interest" varies from very narrow-focused when the radius that defines the cluster around the centroid of the cluster is quite small resulting in an interest cluster that contains only a few agents, to very wide-spread when the radius is greater an contains a large number of agents. We set the radius equal to a value that gave us the best results in a number of preliminary test runs.

Figure 6 - 5 shows how the different number of user interest clusters effects the performance of the system. Notice the non-linear increase in the performance of the system as the number of interest clusters is dropping. Also notice how good the bookmark-list bootsrapping method works when there are few interests clusters. With 10 clusters the system reaches over 60% performance in just 10% of the total generations.

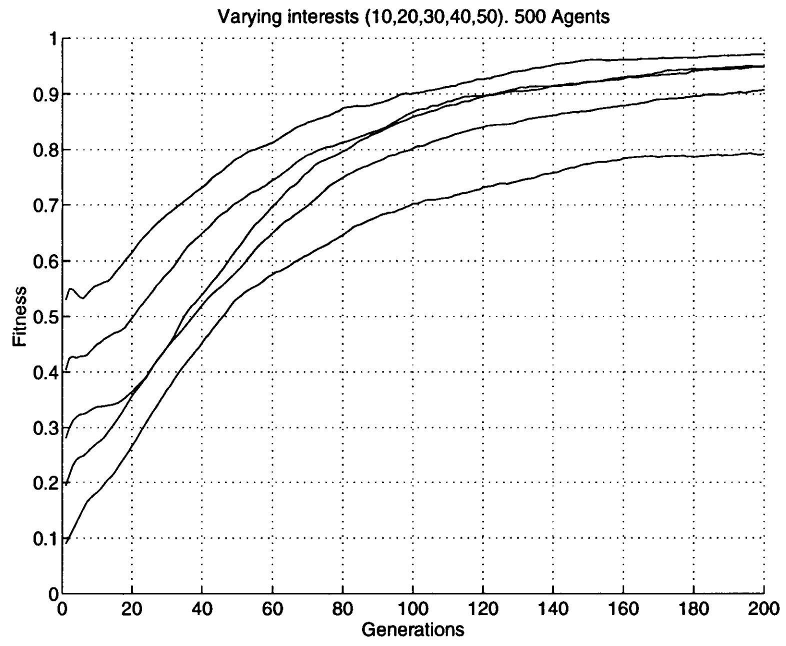


Figure 6 - 5

Filtering Agents and User Interests

The number of filtering agents in the system remains constant while the number of user interest clusters is varying. The five lines (from top to bottom) represent 10, 20, 30, 40 and 50 interest clusters respectively. The performance of the system is over 90% for a number interest clusters less or equal to 40. There is a 13% deterioration in the performance from 40 to 50 interests while this number is less than 5% in the other cases (i.e. 10-20, 20-30, 30-40).

This experiment was conducted in order to assess the performance of the system when the number of the interests the users had increased without a corresponding increase in the number of agents.

Figure 6 - 6 illustrates an overview of all the experiments. The results verify the facts that: *i)* the more agents the better the performance of the system; *ii)* the number of interest clusters does affect the overall fitness but less than one expects (except in the case of many = greater than 40) user interest clusters and *iii)* there seems to be a flattening of the performance curve above 225 agents. Most probably the diversity of the population need to be enforced in large numbers of agents.

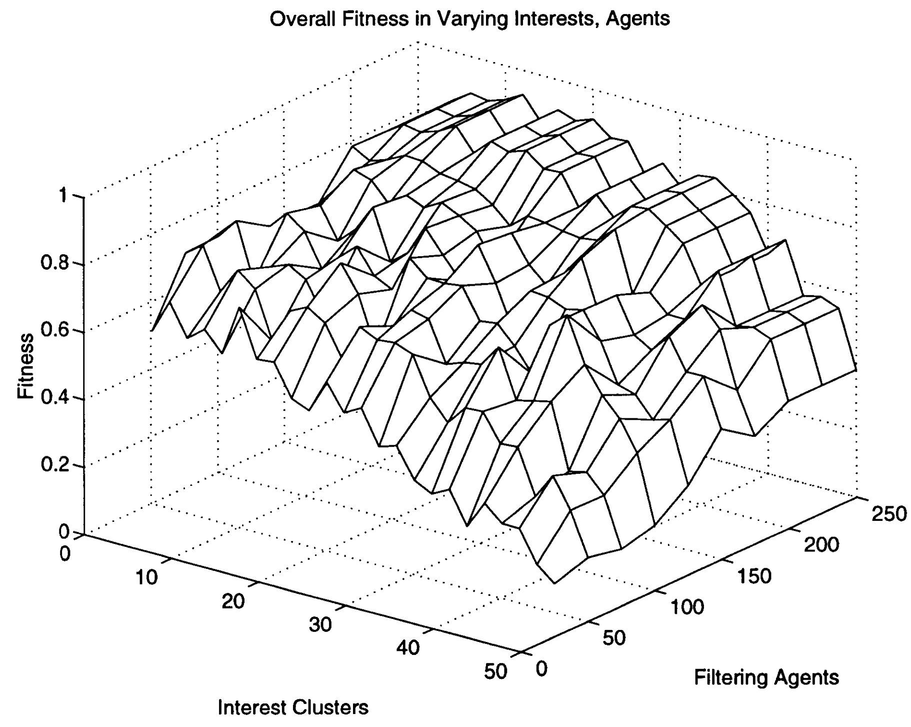


Figure 6 - 6

Overall System fitness vs. Number of User Interest Clusters vs. Number of Agents.

The value displayed for each data point is the maximum of all generations (0-200). The number of User Interest vectors per cluster, totaling more than 550 vectors). It is quite unlikely that a user will have that many interests at the same time. Note that when the number of agents is small and the number of interest clusters is large (greater than 40) the results are not satisfactory. Nevertheless, when the number of agents increases above 100 the overall fitness rises significantly.

6.3 Experiments with Real Users

After validating the assumption that the multiagent system can reach equilibria points and adapt to the user interests regardless the degree that those were changing with, we performed a set of experiments that involved a set of seven users. Our experimentation methods closely follow those introduced by the NewT and Webhound research projects (Sheth and Maes, 1993 and Lashkari, 1995) here at the Media Lab.

One of the goals of this research project was to produce a useful system, utilized frequently by internet users. Experiments to assess the positive or negative feedback of the users were an important part of the evaluation process. A group of seven people were used to test the system. The testers were given a set of instructions on how to submit their bookmark lists to the system, how to manually generate agents, a few overall instructions on how to interact with the system. Because the parameter tuning space in Amalthaea is very big (number of agents, mutation rates, crossover rates, cloning rates, etc.) all the tests were conducted using a fixed set of parameters (because of time and computational resources limitations.)

6.3.1 Evaluation Criteria

The evaluation criteria for the first set of the experiments are similar to Lashkari, 1995. Assuming that set $C = \{ c_1, c_2, c_3, \dots , c_N \}$ represents the confidence (or rating) of the agents' recommendations and set $F = \{ f_1, f_2, f_3, \dots , f_N \}$ the user feedback on the system's suggestions, we define the error set $E = \{ e_1, e_2, e_3, \dots , e_N \} = \{ c_1 - f_1, c_2 - f_2, c_3 - f_3, \dots , c_N - f_N \}$. The measured quantities are the following:

- *Mean Absolute Error.* The smaller this error, the better the performance of the system.

$$|\bar{E}| = \frac{\sum_{i=1}^{N} |e_i|}{N} \tag{13}$$

- *Standard Deviation of Error.* This quantity measures the consistency of the algorithm's performance over the data set. The smaller the standard deviation, the better the algorithm. The standard deviation of the error is defined as:

$$\sigma = \sqrt{\frac{\sum_{i=1}^{N} (E - \bar{E})^2}{N}} \tag{14}$$

- *Correlation Coefficient.* The higher the correlation of the agent confidence prediction to the user rating, the better the algorithm according to Hill et al., 1995

$$r = \frac{\sum_{i=1}^{N} Covariance(c_i, f_i)}{\sigma_c \cdot \sigma_f} \tag{15}$$

- *Extreme Values.* Lashkari, 1995 and Shardanand and Maes, 1995 assert that the confidence of the agents for extreme values (that is weighted values above six or below two) "indicate very strong user preferences" and "are probably more important than other values". The ability of the system to maintain low absolute mean error, standard deviation and high correlation coefficient both in regular and in extreme confidence values is important.
- *Precision:* The percentage of the articles presented to the user that were relevant. Precision is a standard performance measuring quantity in the Information Retrieval community.

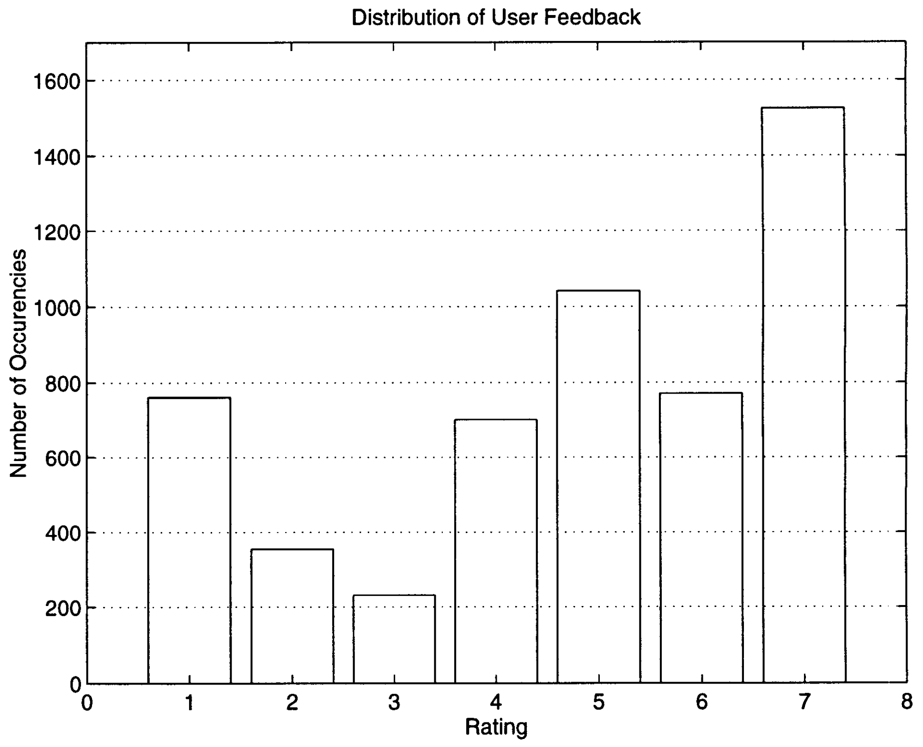


Figure 6 - 7

Distribution of Rating Values in the User Feedback Data Set.

The y-axis denotes the number of documents the users that participated in the experiment rated, while the x-axis denotes the sum of the ratings that the user gave to those document.

6.3.2 Results

During the testing period we were logging the behavior of the system and the users' feedback in order to perform an analysis after the conclusion of the experiments. Figure 6 - 7 shows the distribution of the user feedback over a scale of 1 (for bad) to 7 (for excellent). The x axis measures the feedback scale and the y axis the number of occurrences. In general, users gave more positive feedback to the system than negative. When users disliked a recommendation, they preferred to give the absolute negative rating (1) rather than a somewhat negative response (3 or 2).

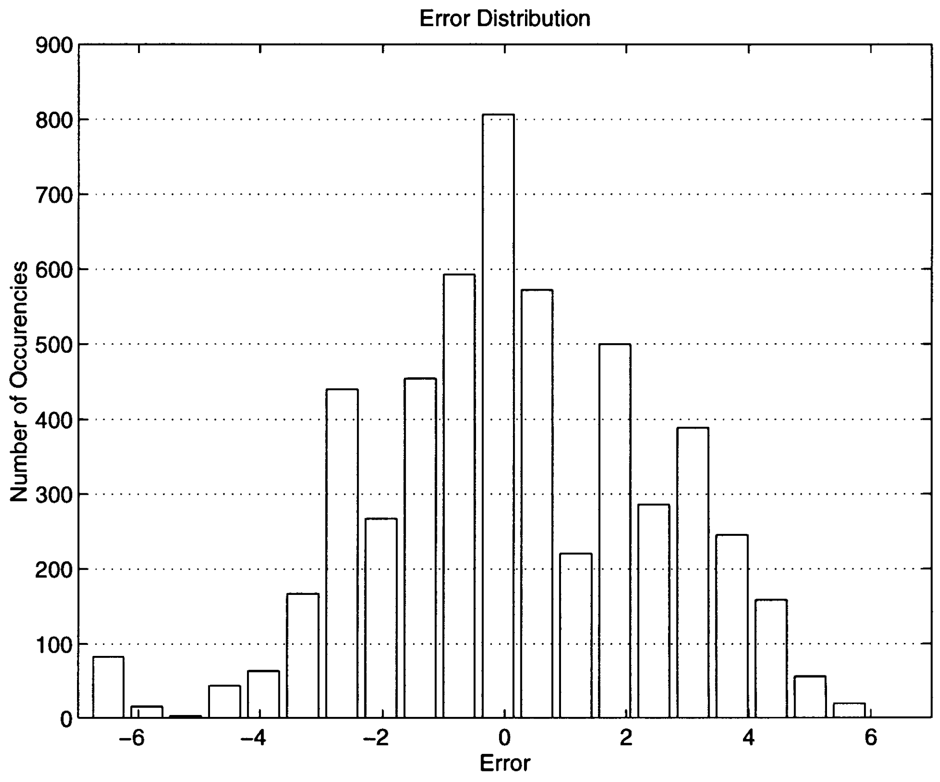


Figure 6 - 8 **Error Distribution in the whole data set.**

The y-axis expresses the number of occurencies of the errors in each range. The distribution has an gaussian-like form.

Figure 6 - 8 above shows the distribution of the error in the whole data set. The error distribution has a gaussian form with a couple high peaks around +2 and -2. One feature of the error distribution is the usually high peak in the negative

end of the x axis, around -6.5. As the figure shows, the system's error distribution is concentrated around 0.

Figure 6 - 9 depicts the evolution of the feedback that the users provided to the system over time. In this scatter plot, the x axis represents the number of times the users provided feedback to the system while the y axis represents the error. The line represents a least-squares fit on the data set. At the beginning of the experiments the absolute error was quite high, around 2.5 (or 35.71%). However, as time passed, users provided feedback and the system evolved, the error dropped to nearly 0.5 (or 7.14%). Also, at the beginning of the experiments users provided both positive and negative feedback: their response to the system's performance had a lot of variations. However, later on as the system's performance increased, the user ratings improved a lot and at the end were mostly positive.

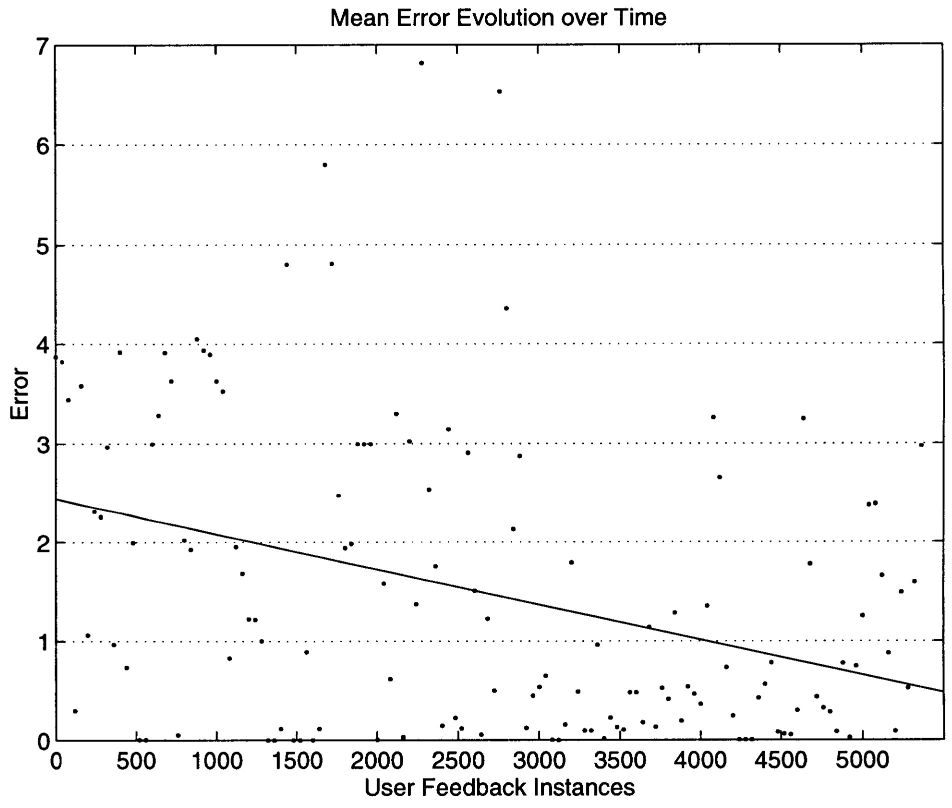


Figure 6 - 9

Plot of the mean absolute error over time.

As the number of feedback instances from the users increases, the system is providing better recommendations.

Table 1 summarizes the performance of the system using the metrics introduced in Section 6.3.1. The mean absolute error of all the users throughout the experiment was roughly 1.5 in a scale of 7. So the average, an agent's recommendation will be within 1.5 rating points from the user's actual interest; in the percentage scale this is translated to an error of 22%. The standard deviation of the absolute error is 1.4 and the correlation between the agents recommendations and the user's interests (as expressed by their feedback) is 0.57. The mean error in the extreme values ratings (1 and 7) is slightly higher (approximately 24%). Standard deviation exhibits the same increasing behavior. The correlation between the mean error and the user interests increases in the case extreme values to 0.62.

Table 1: Mean Absolute Error

	All Values	Extreme Values
Mean Absolute Error	1.5536 (22.19%)	1.6874 (24.11%)
Standard Deviation	1.4015	1.6239
Correlation Coeff.	0.5728	0.6214

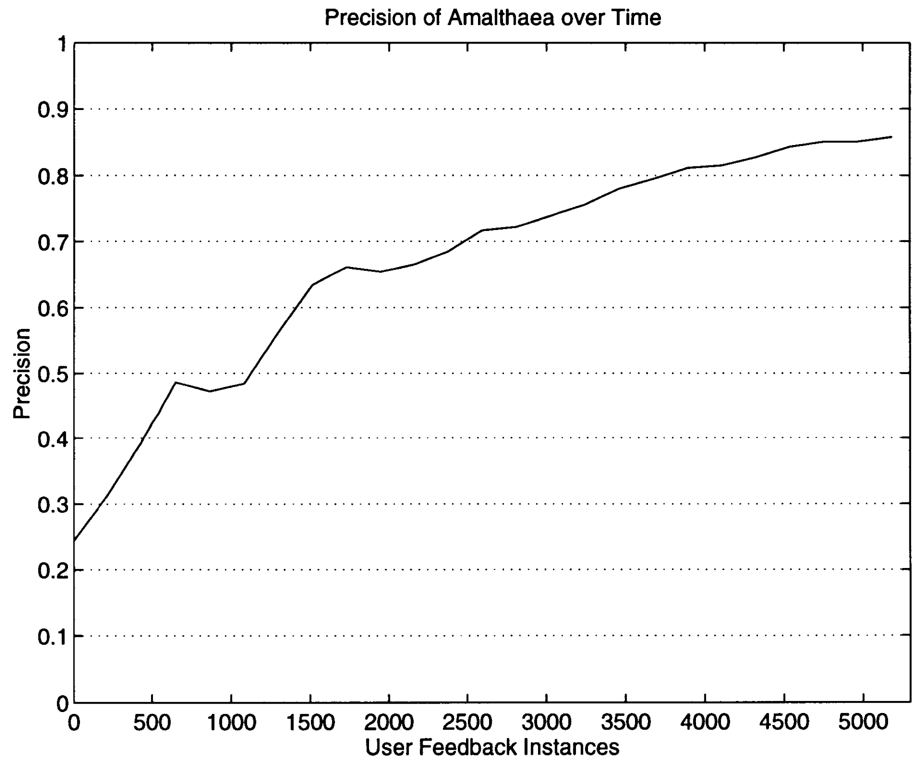


Figure 6 - 10 **Precision rate over time.**

A set of standard performance measures of information retrieval system are precision and recall. Precision is defined as the percentage of the retrieved articles that is relevant and recall as the percentage of relevant articles that were retrieved. In this case, the whole World-Wide-Web constitutes our document collection so we cannot compute the recall performance of the system.

In order to compute the progress of the precision rate of the system over time, we counted as relevant all the documents for which the users gave a rating of more than four. The results are displayed in Figure 6 - 10 and show that the precision of the system is increasing as time passes and Amalthaea is modelling the users better. Although precision is a useful measurement for comparison purposes, we cannot draw many conclusions for the system's performance because we cannot compute the recall quantity. In information retrieval systems, precision improves at the expense of the recall. In this case (where the recall cannot be computed), we believe that the mean error is a better quantitative method for evaluating the ability of the system to perform better over time, since it takes into consideration the agents' confidence when proposing a document.

The experiments performed with real users suggest that Amalthaea can be a useful tool that serves the everyday information needs of its users. The overall performance of the system was good and it was improving over time. Compared to other WWW filtering systems that use different filtering techniques the performance of Amalthaea proved to be as good. Media Laboratory's Webhound project (Lashkari, 1995) is a WWW filtering system that uses featured-guided automated collaborative filtering (FGACF) techniques. Although the underlying technology is different, the similar assessment methods used in both Amalthaea and Webhound enable us to compare the performance of the two systems. Webhound's mean absolute error ranged from 21.18 to 24.77 in the FGACF algorithms, while the correlation coefficient ranged from 0.55 to 0.64 in the same algorithms. Overall, the performance of Amalthaea, a multiagent-based, evolutionary, content-based filtering system was comparable to that of Webhound, a collaborative filtering system, that uses its users' intelligence to provide recommendations.

Our experiments were concluded by the distribution of a set of questionnaires to the users of the system for them to fill out after the experiments. The questionnaire inquired the users about their interaction with the system and was divided into four categories.

- General performance evaluation (overall evaluation)
- Agents Issues (adaptation of the system)
- Interface Issues (communication with the system, ease of use)
- Trust Issues (privacy concerns, authority delegation)

The testers had to choose between five different options when answering the questions (Very Good/4, Good/3, Fair/2, Bad/1, Don't know/0). Although the duration of the experiment was quite small, the majority of the testers felt that the general sense on performance of the system was "Good" and that the recommended articles' relevancy was "Very Good". The feelings were mixed when the tester were asked about the adaptation of the system: a few people responded "Don't know" whereas the some found it "Good". As far as the UI in concerned, most reviewers agreed that it was "Good". Furthermore, most of the testers agreed that the privacy model was "Good" and they claimed they could easily delegate authority to the agents in the system. This is not surprising given the relatively low risk (in terms of damage that might occur to the user) task of Amalthaea. In a more critical application (like an agent that manages personal finances and makes automatic payments) users would be more reluctant to delegate authority to their agents.

Chapter 7 Concluding Remarks and Future Work

Chapter seven contains the concluding remarks of this work. It identifies possible areas of future work. The thesis wraps up with conclusions drawn during the progress of the research and a summary.

This thesis has discussed the idea of using evolving populations of agents for personalized information filtering and discovery. In particular, we introduced the idea of integrating two different populations of agents, the Information Filtering Agents and the Information Discovery Agents into an ecosystem. The two different populations competed and cooperated as the ecosystem worked its way towards equilibria points. We have shown that an evolving multiagent system can converge to loci stable and useful to their users. Based on the above-described architecture, we have built a working system, Amalthaea, that provides to its users personalized information from the World-Wide-Web. In Amalthaea agents that are of service to users or other agents will run more often, reproduce, and survive while incompetent agents will be removed from the population.

The experiments we conducted suggested that the ecosystem of Amalthaea can reach stable equilibria states with both varying and suddenly changing user interests.Moreover the real-users testing phase showed that the system is indeed useful to its users, that the error rate on its predictions goes down and that the precision rate of its predictions goes up after receiving user feedback.

During the progress of this project we have touched upon several interesting research questions that need further investigation. The system equilibria issue is central to the success of Amalthaea and different types of equilibria can be defined; the "overnight" equilibrium refers to a balanced system state before the system presents the digest of articles to the user. A longer-term equilibrium refers to the re-adaptation of the system to the user's interests after a sudden change in those interests. However, other types of equilibria, namely those that emerge as the user learns about a subject area or the speed at which an area of interests is changing as reflected by the changing documents? (Belew, 1997)

The issue of trust which the user must have towards the agent is a very important one; it is quite difficult in a multiagent setting to express the state of the system to the user and to be able to show why the system performed a certain action. Several enhancements can be made to the user interface in order to support the capabilities described above. The visualization solution we are offering can be greatly enhanced in a variety of ways, one of them is by using self-organizing topographical maps (Kohonen, 1989). The idea behind them is a network composed of nodes that would store multi-dimensional weight vectors. When a multidimensional vector is presented to the network, the node whose weight vector is the closest to the presented vector is updated and it gets a bit closer to the presented vector. As certain node areas attract similar vectors, all the sections of the map can be labelled according to the vector type they attract, providing another way of expressing the state of the system. Another UI enhancement has to do with the evolution of the users interests over time. The "user interests playback mechanism" described in Chapter 3 can be augmented with graphs on the flow of information segmented by country or domain. That type of visualization would enable the user to see from where she is drawing from most of the information she is using, what information flow patterns are forming and how they are changing compared to her interests.

When we discussed our privacy concerns over the centralized storage of the users profiles we briefly mentioned ways of deploying a distributed version of the system by placing the IFAs at the user side. A distributed version would have the advantages of keeping the user profiles at a trusted machine, reducing the load of the server and would require minimal communication between the user's machine and the server in the form of the toexplore.dat file. Such a distributed version can be implemented without breaching our requirement of non-continuous internet connection, since when the user dials-up and opens his digest in a browser, the new "toexplore" commands would be uploaded to the server.

The way Amalthaea accesses the search engines right now is quite indirect: it formulates queries based on keywords and the search engines return related sites based on those keywords. If search engines provided an API for directly querying their databases, Amalthaea would be able to pinpoint documents with much greater accuracy: it could utilize the term weights or the eigenvectors of the engines (that are now opaque to any outside observer). Moreover, in the case of the engines that are using term weights to assess similarity, there would be no need to fetch that document over the network: the comparison could be done directly be using the data of the search engine.

Most information discovery systems exhibit the so-called serendipity problem: when reading a newspaper, people often read articles that are of interest to them but they would never have think of asking to read them. In Amalthaea's architecture this problem is partly addressed by the inherent randomness of the evolution process, but this alone is not enough. If Amalthaea, a content-based system was combined with a collaborative filtering system like Webhound (Lashkari, 1995) the system could offer recommendations on novel subjects that were not part of the original IFAs repertoire.

References

Armstrong, R., Freitag, D., Joachims, T., and Mitchell, T. (1995). Webwatcher: A learning apprentice for the world wide web. In *Proceedings of the Symposium on Information Gathering from Heterogeneous, Distributed Environments*. AAAI Press.

Baclace, P. (1992). Competitive agents for information filtering. *Communications of the ACM*, 35 (12):50.

Balabanovic, M. and Shoham, Y. (1995). Learning information retrieval agents: Experiments with automated web browsing. In *AAAI Technical Report SS-95-08, Proceedings of the 1995 AAAI Spring Symposium Series.*.

Bartell, B., Cottrell, G., and Belew, R. (1994). Automatic combination of multiple ranked retrieval systems. In *Proceedings of the 1994 SIGIR Conference.*

Belew, R. (1989). *Evolution Learning and Culture: Computational metaphors for adaptive algorithms.* UC at San Diego Technical Report CS89-156.

Belew, R. (1997). Personal communication.

Belew, R. and Mitchell, M. (1996). *Adaptive individuals in evolving populations.* Addison-Wesley.

Belkin, N. and Croft, B. (1992). Information filtering and informatiuon retrieval. *Communications of the ACM*, 35, No. 12:29–37.

Best, M. (1997). Corporal ecologies and population fitness on the net. *submitted to Journal of Artificial Life.*

Chavez, A. and Maes, P. (1996). Kasbah: An agent marketplace for buying and selling goods. In *Proceedings of the First International Conference on the Practical Application of Intelligent Agents and MultiAgent technology (PAAM), London 1996.*

Chavez, A., Moukas, A., and Maes, P. (1997). Challenger: A multiagent system for distributed resource allocation. In *Proceedings of the First International Conference on Autonomous Agents, Los Angeles, 1997.*

Chesnais, P., Mucklo, M., and Sheena, J. (1995). The fishwrap personalized news system. In *Proceedings of the 2nd International Workshop on Community Net*. IEEE Press.

Clearwater, S. (1996). A comparative-developmental approach to understanding imitation. In Clearwater, S., editor, *Market Based Control: a paradigm for distributed resource allocation.*

Crabtree, B. (1997). Personal communication.

Decker, K. and Lesser, V. (1995). Macron: An architecture for multi-agent cooperative information gathering. In *CIKM Conference, Workshop on Intelligent Information Agents.*

Decker, K., Pannu, A., Sycara, K., and Williamson, M. (1997). Designing behaviors for information agents. In *Proceedings of the First International Conference on Autonomous Agents, Los Angeles, 1997.*

Etzioni, O. (1995). Results from using the metacrawler. In Varela, F. and Bourgine, P., editors, *Proceedings of the Fourth WWW Conference.* MIT Press.

Etzioni, O. (1996). Moving up the information food chain: deploying softbots on the www. In *Proceedings of the AAAI-96.* AAAI Press.

Frystyk, H. and Lie, H. (1994). Towards a uniform library of common code. In *Proceedings of the Second WWW Conference.*

Grosof, B. (1995). Reusable architecture for embedding rule-based intelligence. In *CIKM Conference, Workshop on Intelligent Information Agents.*

Hill, W., Stead, L., Resenstein, R., and Furnas, G. (1995). Recommending and evaluating choices in a virtual community of use. In *Proceedings of CHI '95, Denver, CO.*

Holland, J. (1962). Outline for a logical theory of adaptive systems. *JACM*, 9:297–314.

Holland, J. H. (1975). *Adaption in natural and artificial systems.* The University of Michigan Press.

Kohonen, T. (1989). *Self-Organization and Associative Memory.* Springer-Verlag, Berlin.

Koza, J. (1992). *Genetic Programming.* MIT Press.

Labrou, Y. and Finin, T. (1994). A semantics approach for kqml - a general purpose communication language for software agents. In *Proceedings of Conference on Information and Knowledge Management 1994.* ACM Press.

Lang, K. (1995). Newsweeder. In *Proceedings of the 12th International Conference on Machine Learning.*

Lashkari, Y. (1995). Webhound? Master's thesis, MIT Media Laboratory.

Lieberman, H. (1995). Letizia, an agent that assists web browsing. In *Proceedings of IJCAI-95.* AAAI Press.

Maes, P. (1994). Agents that reduce work and information overload. *Communications of the ACM*, 37, No. 7:31–40.

Maes, P. (1997). Personal communication.

Malone, T., Grant, K., Turbak, F., Brobst, M., and Cohen, M. (1987). Intelligent information sharing sytems. *Communications of the ACM*, 30, No. 5:390–402.

Mataric, M. (1993). Designing emergent behaviors: From local interactions to collective intelligence. In *From Animals to Animats II*. MIT Press.

Menczer, F., Belew, R., and Willuhn, W. (1995). Artificial life applied to adaptive information agents. In *Working Notes of the AAAI Symposium on Information Gathering from Distributed, Heterogeneous Databases*. AAAI Press.

Miller, G. (1985). Wordnet: A dictionary browser. In *Proceedings of the First Conference of the UW Centre for the New Oxford Dictionary. Waterloo, Canada.*

Mitchell, M. (1996). *An Introduction to Genetic Algorithms*. MIT Press.

Moukas, A. (1996). Amalthaea: Information discovery and filtering using a multiagent evolving ecosystem. In *Proceedings of the First International Conference on the Practical Application of Intelligent Agents and MultiAgent technology (PAAM), London 1996.*

Moukas, A. (1997). Amalthaea, an intelligent agent for information overload. *to appear in International Journal of Applied Artificial Intelligence.*

Moukas, A. and Hayes, G. (1996). Synthetic robotic language acquisition by observation. In *From Animals to Animats*. MIT Press.

Moukas, A. and Zacharia, G. (1997). Evolving multiagent filtering solutions with amalthaea. In *Proceedings of the First International Conference on Autonomous Agents, Los Angeles, 1997.*

Porter, M. (1980). An algorithm for suffix stripping. *Program*, 14(3):130–138.

Resnik, P., Iacovou, N., Sushak, M., Bergstrom, P., and Riedl, J. (1994). Grouplens: An open architecture for collaborative filtering of netnews. In *Proceedings of CSCW'94.*

Riordan, A. O. and Sorensen, H. (1995). An intelligent agent for high-precision information filtering. In *Proceedings of the CIKM-95 Conference.*

Salton, G. (1971). *The SMART retrieval System. Experiments in automatic document processing.* Englewood Clifs, NJ.

Salton, G. (1983). *Introduction to Modern Information Retrieval.* McGraw-Hill.

Salton, G. and Buckley, C. (1987). *Text Weighting Approaches in Automatic Text Retrieval.* Cornell University Technical Report 87-881.

Shardanand, U. and Maes, P. (1995). Social information filtering: Algorithms for automating 'word of mouth'. In *Proceedings of the CHI-95 Conference, Dencer, CO.* ACM Press.

Sheth, B. and Maes, P. (1993). Evolving agents for personalized information filtering. In *Proceedings of the Ninth Conference on Artificial Intelligence for Applications, 1993.* IEEE Computer Society Press.

Sycara, K. (1995). Intelligent agents and the information revolution. In *UNICOM Seminar on Intelligent Agents and their Business Applications.* November 8-9, London.

Sycara, K. and D., Z. (1996). Coordination of multiple intelligent software agents. *International journal of Intelligent and Cooperative Information Systems,* 5(2-3):181–211.

Sycara, K., Decker, K., Pannu, A., Williamson, M., and D., Z. (1996). Distributed intelligent agents. *IEEE Expert,* 11(6).