Capturing knowledge of user preferences with recommender systems

Stuart E. Middleton

Intelligence, Agents, Multimedia Group (IAM group)

University of Southampton

Southampton, SO17 1BJ, UK.

Contact sem99r@ecs.soton.ac.uk

A Mini-Thesis submitted for transfer of

registration from M.Phil. to Ph.D.

July 2, 2001

Supervisor: Professor David De Roure

Table of Contents

Table of Figures

Table of Tables

Abstract

Capturing user preferences is a problematic task. Simply asking the users what they want is too intrusive and prone to error, yet monitoring behaviour unobtrusively and then finding meaningful patterns is difficult and computationally time consuming. Capturing accurate user preferences is however, an essential task if the information systems of tomorrow are to respond dynamically to the changing needs of their users.

This thesis examines the issues associated with building user profiles for a recommender system from unobtrusively monitored user behaviour. A novel multi-class approach to content classification is examined, allowing domain knowledge to be utilized during the profiling process. The effectiveness of using an "is-a" hierarchy to profile users domain interests is assessed. The utility of profile visualization is also assessed, in which users engage in a dialogue with the recommender system about their domain interests.

Two experimental systems are constructed, each testing one of the above approaches. Empirical evidence is gathered from these experiments, and conclusions drawn.

Chapter 1 Introduction

Chapter summary
The motivation behind this thesis is described.
The structure to this thesis is detailed.
The thesis contribution is explained.

1.1 Motivation

Capturing user preferences is a problematic task. Simply asking the users what they want is too intrusive and prone to error, yet monitoring behaviour unobtrusively and then finding meaningful patterns is difficult and computationally time consuming. Capturing accurate user preferences is however, an essential task if the information systems of tomorrow are to respond dynamically to the changing needs of their users.

This thesis examines issues associated with building user profiles for a recommender system from unobtrusively monitoring user behaviour. If the work presented here sheds some light on the path to truly personalized systems it will have been worthwhile.

1.2 Thesis structure

As recommender systems are a type of interface agent, the agent field is first reviewed in chapter 2, explaining how agents came about from their birth in artificial intelligence. The current state of the agent field is categorized and reviewed, and recommender systems clearly placed within it. Recommender systems are then examined in more detail within chapter 3, defining exactly what the requirements are for a recommender system and reviewing the systems available today.

Two experimental recommender systems are described in chapters 4 and 6. These systems are then used to conduct two experiments, the first described in chapter 5 and the second in chapter 6 (the experiment has yet to be conducted but is described). The first experiment evaluates how domain concept representation affects profiling and hence recommendation performance. The second experiment evaluates how profile representation and visualization affects recommendation performance (this experiment has yet to be attempted). In addition to these experiments overall performance metrics are measured and system performance compared to other systems within the literature.

Chapter 7 discusses the future direction of the work presented here, and a three-year plan is detailed. The work to-date is at the halfway point of the three-year plan and is proceeding on-track to competed in August 2002.

A glossary of terms is provided in Appendix A and a summary of each reviewed system provided in Appendix B. Design details of both experiments systems are listed in Appendices C and D.

1.3 Contribution

The following will be the contributions of my final thesis.

Novel approach

Both the Quickstep and Foxtrot systems use the novel idea of using a multi-class approach to recommender systems. This has not been attempted before as other recommender systems use a binary class ("interesting" and "not interesting") approach because of its better accuracy. My system explores coupling this multi-class representation with domain knowledge to compensate for the reduced accuracy of multi-class classification.

My thesis is that the rewards (in terms of profile accuracy) for having a dialogue with the user about their profile outweigh the cost of representing the profile in a way understandable to humans.

New evaluation results for an existing concept

The current literature seriously lacks quantitative evaluations of recommender systems addressing real world problems. Both the Quickstep and Foxtrot systems are evaluated using real people while performing a real task. This in itself is an important contribution to the knowledge of the recommender systems community.

The evaluation of different profiling techniques on the users log data is also a contribution of relevance. The result of this analysis will provide hard evidence for any future system contemplating a choice of technique.

Applying a known concept to a new domain

The Quickstep and Foxtrot recommender systems could be viewed as providing the users with links to web documents. The hypermedia community does not use recommender systems much so would be a novel application of my work. A claim such as “collaborative filtering applied to adaptive hypermedia” could be made.

Chapter 2 Agents: a review of the field

Chapter summary
The history behind software agents is described, charting major events from the 50's to the modern day.
Interface agents are defined and the challenges within the field identified.
An interface agent technology taxonomy is produced.
The current state of the art for interface agents is reviewed, and agent system classified using the technology taxonomy.
Current trends are described and the degree to which the interface agent challenges are met discussed.

The 1990's have seen the dawning of a new paradigm in computing - software agents. Many researchers are currently active in this vibrant area, drawing from more traditional research within the artificial intelligence (AI) and human computer interaction (HCI) communities. Leading figures [35] argue that one aspect of software agent systems, the interface agent, has the potential to revolutionize computing as we know it, allowing us to advance from direct manipulation of systems to indirect interaction with agents. Removing the requirement for people to manage the small details of a task liberates individuals, empowering them to accomplish impressive goals otherwise requiring teams of experts.

Software agents originated from the field of artificial intelligence, way back in the 1950's. The next section describes some of the important landmarks that happened along the way to where we are today. The current state of the interface agent community is then examined, providing a basis for categorization and a review of the field.

The work in this chapter has also been published as a technical report [56].

2.1 History behind software agents

Alan Turing, famous for his work on computability [100], posed the question "Can machines think?" [99]. His test, where a person communicates over a Teletype with either a person or a computer has became known as the Turing test. The goal of the

Turing test is to build a computer capable of fooling a human at the other end. It is the Turing test that inspired the artificial intelligence community.

The discipline of artificial intelligence (AI) was born in the 1950's. Marvin Minsky, after some research into neural networks that was deemed a failure at the time due to the difficulty of learning weights, teamed up with John McCarthy at MIT to work on symbolic search-based systems. At the same time at Carnegie-Mellon, Allen Newell and Herbert Simon were successfully exploring heuristic searching to prove logic theorems. With the scene thus set, initial successes led to heuristic search of symbolic representations becoming the dominant approach to AI.

The 1960's saw much progress. Now at Stanford, McCarthy [51] had just invented LISP and set about representing the world with symbols, using logic to solve problems within these worlds [53]. At the same time Newell [67] created the General Problem Solver which, given a suitable representation, could solve any problem. Problems solved were in simple, noise and error free symbolic worlds, with the assumption that such solutions would generalize to allow larger, real world problems to be tackled. Researchers did not worry about keeping computation on a human time-scale, using the massive increases in hardware performance to constantly increase the possible search space size, thus solving increasingly impressive problems.

During the 1970's, searching became better understood [68]. Symbolic systems still dominated, with continuing hardware improvements fuelling steady, successful progress. Robots were created, for example Shakey [69], that lived in special block worlds, and could navigate around and stack blocks sensibly. Such simplified worlds avoided the complexity of real world problems. The assumption, underpinning all the symbolic research, that simple symbolic worlds would generalize to the real world, was about to be found wanting.

In the 1980's, expert systems were created to try to solve real problems. McCarthy [52] had realized that "common sense" was required in addition to specialized domain knowledge to solve anything but simple microworld problems. A sub-field of AI, knowledge representation, came into being to examine approaches to representing the everyday world. Unfortunately the idea of "common sense" proved impossible to represent, and knowledge-based systems were widely viewed to have failed to solve real-world problems. At the same time, the backpropagation algorithm [83] caused a resurgence of interest in neural network approaches, and Minsky [58] examined an agent-based approach for intelligence.

The late 1980's and early 1990's saw the decline of search-based symbolic approaches. Brooks [13] convincingly challenged the basic assumptions of the symbolic approaches, and instead created embodied, grounded systems for robots using the "world as its own best model". This bottom up approach was termed *nouvelle AI*, and had some initial successes. However, it too failed to scale up to real-world problems of any significant complexity. Connectionist approaches were aided by new parallel hardware in the early 1990's, but the complexity of utilising a parallel architecture led such systems to fail in the marketplace.

Knowledge engineering, now widely seen as costly and hard to re-use, was superseded by machine learning techniques borrowed from AI. Towards the end of the 1990's, pattern-learning algorithms [59] could classify suitable domains of knowledge (such as news stories, examination papers) with as much accuracy as manual classification. Hybrids of traditional and nouvelle AI started to appear as new approaches were sought.

During the mid 1990's, Negroponte's [66] and Kay's [35] ideas for indirect HCI were coupled with Minsky's [58] ideas on intelligence, leading to the new field of agent-based computing. Experiments with interface agents that learnt about their user [44], and multi-agent systems where simple agents interacted to achieve their goals [107] dominated the research. Such agent systems were all grounded in the real world, using proven AI techniques to achieve concrete results.

User modelling changed in the 1990's too, moving from static hand crafted representation of the user characteristic in the 1980's knowledge representation approach to dynamic behaviour based models [36]. Machine learning techniques proved particularly adept at identifying patterns in user behaviour.

2.2 Issues and challenges for interface agents

Maes [44] describes interface agents as:

> "Instead of user-initiated interaction via commands and/or direct manipulation, the user is engaged in a co-operative process in which human and computer agents both initiate communication, monitor events and perform tasks. The metaphor used is that of a *personal assistant* who is *collaborating with the user* in the same work environment."

The motivating concept behind Maes' interface agents is to allow the user to delegate mundane and tedious tasks to an agent assistant. Her own agents follow this direction,

scheduling and rescheduling meetings, filtering emails, filtering news and selecting good books. Her goal is to reduce the workload of users by creating personalized agents to which personal work can be delegated.
There are many interface agent systems and prototypes, inspired by Maes [44] early work, situated within a variety of domains. The majority of these systems are reviewed and categorized in the next section using the interface agent technology taxonomy shown in figure 2.1. Common to these systems, however, are three important issues that must be addressed before successful user collaboration with an agent can occur:

- Knowing the user
- Interacting with the user
- Competence in helping the user

Knowing the user involves learning user preferences and work habits. If an assistant is to offer help at the right time, and of the right sort, then it must learn how the user prefers to work. An eager assistant, always intruding with irrelevant information, would just annoy the user and increase the overall workload.
The following challenges exist for systems trying to learn about users:

- Extracting the users' goals and intentions from observations and feedback
- Getting sufficient context in which to set the users' goals
- Adapting to the user's changing objectives
- Reducing the initial training time

At any given time, an interface agent must have an idea of what the user is trying to achieve in order to be able to offer effective assistance. In addition to knowing what the user's intentions are, contextual information about the user's current situation is required to avoid potential help causing more problems than it solves. Machine learning techniques help here, but which should be used and why?

Another problem is that regular users will typically have a set of changing tasks to perform. If an agent is to be helpful with more than one task, it must be able to discover when the user has stopped working on one job, and is progressing to another – but what is the best way to detect this?

Users are generally unwilling to invest much time and effort in training software systems. They want results early, before committing too much to a tool. This means that interface agents must limit the initial period before the agent learns enough about the user to offer useful help. What impact does this have on an agent's learning ability?

A metaphor for indirect HCI has yet to reach maturity, so remains an open question. What is known are the lessons learnt from direct manipulation interfaces. Principally these include the need for users to feel in control, avoiding unreasonable user expectations and making safe systems that users can trust [70].

Interacting with the user thus presents the following challenges:

- Deciding how much control to delegate to the agent
- Building trust in the agent
- Choosing a metaphor for agent interaction
- Making simple systems that novices can use

It is known from direct manipulation interfaces that users want to feel in control of what their tools are doing. By the nature of an autonomous interface agent, some control has been delegated to it in order for it to do its task. The question is, how do we build the users' trust, and once a level of trust is established how much control do we give to the agents? Shneiderman [93] argues for a combination of direct manipulation and indirect HCI, promoting user understanding of agents and the ability for users to control agent behaviour directly. Can we build these guiding principles into our systems?

Interface metaphors served direct manipulation interfaces well (e.g. the desktop metaphor), guiding users in the formation of useful conceptual models of the system. New metaphors will be required for indirect HCI, presenting agents in a way helpful to users new to the system. Ideally, interface agents should be so simple to use that

delegating tasks becomes a natural way of working, amenable to the novice user – but what is a natural way of working with agents?
Lastly, there is the issue of competence. Once the agent knows what the user is doing and has a good interaction style, it must still formulate a plan of action that helps, not hinders, the user. The challenges are:

- Knowing when (and if) to interrupt the user
- Performing tasks autonomously in the way preferred by the user
- Finding strategies for partial automation of tasks

There is very little current research into how users can be best helped. Work from other disciplines such as computer supported co-operative working (CSCW) help, but real user trials are needed to demonstrate and evaluate the effectiveness and usefulness of the personalized services performed by interface agents [65]; if an agent does not reduce the workload of a real user in a real work setting it will be deemed by users as less than useful.

2.3 Taxonomy of interface agent systems

Several authors have suggested taxonomies for software agents as a whole (see [65] and [107] for examples), but they tend to address interface agents as a monolithic class, citing a few examples of the various prototypical systems. With the maturing of the agent field, and the growing number of interface agents reported in the literature, a more detailed analysis is warranted. Mladenić [62] goes some way to achieving this requirement, adopting a machine learning view of interface agents.
Interface agents can be classified according to the role they perform, technology they use or domain they inhabit. Interface agents are moving from the research lab to the real world, significantly increasing the roles and domains for agents, as entrepreneurs find new ways to exploit new markets. The fundamental technology behind the agents, however, is undergoing less radical change and thus provides a more stable basis on which to build a useful taxonomy.
On this basis, a survey of current interface agent technology has been performed. The result is a non-exclusive taxonomy of technologies that specific agents can support.

Appendix B details the specific systems reviewed and figure 2.1 shows the technology taxonomy applied to the survey.

- ❑ Character-based agents
- ❑ Social agents
    - o Recommender systems
- ❑ Agents that learn about the user
    - o Monitor user behaviour
    - o Receive user feedback
        - ▪ Explicit feedback
        - ▪ Initial training set
    - o Programmed by user
- ❑ Agents with user models
    - o Behavioural model
    - o Knowledge-based model
    - o Stereotypes

Figure 2.1 : Taxonomy of interface agent technology

Character-based agents employ interfaces with advanced representations of real world characters (such as a pet dog or a human assistant [45]). Such agents draw on existing real-world protocols, already known to even novice users, to facilitate more natural interaction. There are also applications in the entertainment domain, creating state of the art virtual worlds populated by believable agents.

Social agents talk to other agents (typically other interface agents of the same type) in order to share information. This technique is often used to bootstrap new, inexperienced interface agents with the experience of older interface agents (attached to other users).

Recommender systems are a specific type of social agent. They are also referred to as collaborative filters [76] as they find relevant items based on the recommendations of others. Typically, the user's own ratings are used to find similar users, with the aim of sharing recommendations on common areas of interest.

Agents employing a learning technology are classified according to the type of information required by the learning technique and the way the user model is represented. There are three general ways to learn about the user – monitor the user, ask for feedback or allow explicit programming by the user.

Monitoring the user's behaviour produces unlabelled data, suitable for unsupervised learning techniques. This is generally the hardest way to learn, but is also the least intrusive. If the monitored behaviour is assumed to be an example of what the user wants, positive examples can be inferred.

Asking the user for feedback, be it on a case-by-case basis or via an initial training set, produces labelled training data. Supervised learning techniques can thus be employed, which usually outperform unsupervised learning. The disadvantage is that feedback must be provided, requiring a significant investment of effort by the user.

User programming involves the user changing the agent explicitly. Programming can be performed in a variety of ways, from complex programming languages to the specification of simple cause/effect graphs. Explicit programming requires significant effort by the user.

User modelling [36] comes in two varieties, behavioural or knowledge-based. Knowledge-based user modelling is typically the result of questionnaires and studies of users, hand-crafted into a set of heuristics. Behavioural models are generally the result of monitoring the user during an activity. Stereotypes [81] can be applied to both cases, classifying the users into groups with the aim of applying generalizations to people in those groups.

Specific interface agents will often implement several of the above types of technology, and so would appear in multiple classes. Common examples are agents that learn about the user to also support a user model, or agents that monitor the user to also allow the user to give explicit feedback. The presented taxonomy ought to be robust to the increase in new systems, as the fundamental technology of machine learning and user modelling are unlikely to change as quickly.

2.4 Review of current interface agent systems and prototypes

A comparison of interface agents is difficult since there are no widely used standards for reporting results. Where machine learning techniques are employed, standard tests such as precision and recall provide useful metrics for comparing learning algorithms normally applied to benchmark datasets. However, the best test of an interface agent's ability to help a user is usually a user trial, where real people are studied using the agent system. Unfortunately, user trials in the literature do not follow a consistent methodology.

The analysis will focus on classifying the agent, identifying techniques used and reporting results published by the original author (if any). This will make quantitative comparison difficult, but allow qualitative comparisons to be made.

Appendix B has more details of each of the interface agent systems reviewed.

2.4.1 Classification of agent systems

Table 2.1 lists the agent systems reviewed and shows how they are classified. The distribution of technologies within today's agents can be clearly seen.

	Character-based agent	Social agent	Recommender system	Monitor user behaviour	Explicit feedback	Initial training set	Programmed by user	Behavioural model	Knowledge-based	Stereotypes
ACT	o			o				o		
ALIVE	o									
Cathexis	o									
CAP		o		o				o		
COLLAGEN		o								
Do-I-Care		o			o	o				
ExpertFinder		o		o				o		
Kasbah		o					o			
Maxims		o		o			o	o		
Meeting sch'ing agent (Maes)		o		o	o			o		
Sardine		o					o			
SOAP		o	o		o			o		
Yenta		o				o				
Campiello			o		o			o		
Community Search Assistant			o							
EFOL			o	o			o	o		
ELFI			o	o				o		
Expertise Recommender			o						o	
Fab			o	o	o					
GroupLens			o		o					
ifWeb			o		o			o		
MEMOIR			o	o			o	o		
PHOAKS			o	o						
ProfBuilder			o	o				o		
Referral Web			o			o				
Ringo			o		o					
Siteseer			o			o				
SurfLen			o	o				o		
Tapestry			o		o		o	o		
Virtual Reviewers (Tatemura)			o		o					
AARON				o	o			o		
Ad've web site agent (Pazzani)				o		o		o		
Amalthaea				o	o			o		
ANATAGONOMY				o	o			o		
CILA				o	o	o		o		
CIMA				o				o		
Coach				o					o	o
Eager				o				o		
GALOIS				o				o	o	o
GESIA				o				o		
Jasper				o	o			o		
Krakatoa Chronicle				o	o			o		
LAW				o	o			o		
Let's Browse				o		o		o		
Letizia				o				o		
MAGI				o	o			o		
Margin Notes				o		o				
NewsDude				o		o		o		
Open Sesame!				o				o		
Personal WebWatcher				o				o		
Remembrance agent				o				o		
Sentence comp'er (Schlimmer)				o				o		
Travel assistant (Waszkiewicz)				o	o			o		
WebACE				o				o		
WebMate				o	o			o		
WebWatcher				o		o		o		
WBI				o				o		
ARACHNID					o	o				
IAN					o					
Learning p'nl agent (Sycara)					o	o				
LIRA					o			o		
NewsWeeder					o	o		o		
NewT					o	o	o	o		
MailCat					o	o				
Re:Agent					o	o				
Syskill & Webert					o	o		o		
UCI GrantLearner					o	o		o		
Butterfly						o				
CiteSeer						o				
Grammex							o			
Meeting scheduler (Haynes)							o			
Mondrian							o			
Softbot							o			
SAIRE									o	o

Table 2.1 : Classification of interface agent systems

2.4.2 Results published for interface agent systems

Table 2.2 lists the agent systems reviewed and shows what results have been published. As there is no standard way to report results, this analysis only allows qualitative comparisons to be made.

Interface agent system	*Results*
Amalthaea	After 5000 user feedback instances, error averaged 7% with a large scatter (0 – 30%). Sudden changes in user interest were tracked after about 20 generations.
ANATAGONOMY	1-10% error after 3 days settling time
CAP	31-60% accuracy (average of 47%) not sufficient for automation, rules were human readable which improved user understanding
CILA	Constructive induction was most accurate (only artificial results however)
Coach	Student performance improved, knowledge of functions improved by a factor of 5
Eager	Users felt a loss of control; macros for some irrelevant small patterns were created
EFOL	12 people (all researchers) used the system on two separate occasions. Half the users reported other people's recipes influenced them, and the pictures of food made them feel hungry.
ELFI	220 users, divided into 5 groups. The user activity logs were used as training/test data using a cross validation method. simple Bayes classifier 91-97% accuracy, kNN 94-97% accuracy
Fab	ndpm measure (distance of user rankings from profile rankings) 0.2 - 0.4 using Fab system, 0.75 – 0.5 using random selection
GroupLens	Various Usenet use figures are presented.
IAN	Accuracy – C4.5 broad topics 70%, narrow topics 25-30% IBPL broad topics 59-65%, narrow topics 40-45%
IfWeb	Results on tests using 4 subjects on a limited set of documents (4-6). 9 sessions were conducted, with learning from feedback occurring between each session. Precision 65%, ndpm 0.2
Kasbah	Users wanted more "human like" negotiation from the agents, otherwise well received
LAW	Accuracy TF 60-80%(best), TF-IDF 55-70%, term relevance 60-65% and accuracy IBPL 65-83%(best), C4.5 55-65%
Let's browse	50 (as opposed to 10 in Letizia) keywords needs, reflecting a groups wider interests; system well received by users (no controlled experiments however)
LIRA	LIRA matched human performance; pages were very similar to each other
MailCat	0.3 second classification time, 60-80% accuracy giving user one choice, 80-98% accuracy giving user 3 choices of folder.
Margin Notes	6% of suggestions were followed; users found suggestion summaries useful (even without following links)
Meeting scheduling agent	Confidence for correct predictions settles to 0.8 to 1.0. Confidence for incorrect predictions settles to 0 to 0.2. Some rouge confidence values remain after settling time.
NewsDude	Accuracy 60-76% (using hybrid of long and short term models), F1 measure 25-60%
NewsWeeder	TF-IDF precision was 37-45%, MDL precision was 44-59% (best)
NewT	Results: Users liked the system and found it useful; the simple keyword representation was a limitation
Open Sesame!	2 out of 129 suggestions were followed – system deemed to have failed; action patterns do not generalize across situations well
Pannu's learning personal agent	Neural network precision 94% recall 60%, TD-IDF precision 93% recall 100% (best)
Pazzani's adaptive web site agent	68% increase in publications downloaded (tech papers domain), 16% increase (goat domain)
Personal WebWatcher	Classification accuracy Bayes 83-95%, nearest neighbour 90-95% (best)
PHOAKS	The filter rules have a precision of 88% with a recall of 87%. When compared to the newsgroups FAQ, the 1st place URL had a 30% probability of appearing in the FAQ.
Re:Agent	Classification accuracy – neural network 94.8 ± 4.2%, nearest neighbour 96.9 ± 2.3%; high accuracy due to simple classification task (into "work" or "other" categories)
Remembrance agent	Email most useful for up-to-date contextual information, RA preferred over a search engine or Margin notes [77].
Ringo	A real/predicted scatter plot is presented
Schlimmer's text completion agent	FSM compares well with ID4 and Bayes, with a hybrid of FSM and ID4 working best. Accuracy of 12-82% was seen, depending on the topic of the notes being taken.
Siteseer	1000 users, 18% confidence recommending 88% of the time.
SurfLen	Some quantitative figures for 100 simulated users (based on Yahoo log data)
Syskill and Webert	Average precision ratings are TF-IDF 85%, Bayes 80%, nearest neighbour 80%, ID3 73%. Nearest neighbour is thought to be best overall (being more consistent than TF-IDF), especially if many examples are available.
WebACE	Speed to find new low entropy pages, Autoclass 38 mins, HAC 100 mins, PCDP and association rule < 2 mins (best)

WebMate	Average accuracy 52% for top 10 recommendations, 30.4% for all recommendations; Accuracy lowered by web advertisements and irrelevant text surrounding articles.
WebWatcher	TF-IDF accuracy 43%, Human accuracy 48%

Table 2.2 : Published results of interface agent systems

2.5 Conclusions from agent classification

Behavioural user modelling dominates the interface agent field. Behavioural user models are usually based on monitoring the user and/or asking the user for relevance feedback. The statistical information generated by these approaches is usually fed to some form of machine learning algorithm.

Almost all the non-social interface agents reviewed use a textual, content-based learning approach deriving information from user emails, web documents, papers and other such sources. The “bag of words” document representation dominates the field, with TF-IDF proving to be a popular choice of word weighting algorithm. Relevance feedback is normally used to provide labels for documents, allowing supervised learning techniques to be employed.

Social interface agents, using collaborative learning approaches, are in the minority, but have proved useful when systems have many users. The main problem with a purely social system is that performance is initially poor until such a time as enough people are using the system. Hybrid systems, using content-based techniques to kick-start the learning process, do address this problem to some extent and comprise about half the social agents reviewed.

Experimental results, where published, tend to be either qualitative user studies or quantitative measurements against benchmark document collections. Leading figures in the field [65] have observed that it is yet to be proven that interface agents actually help people at all. To gain evidence to this end, experiments with real users in real work settings must be performed, and ways found to compare different approaches with criteria such as helpfulness. Only then will the interface agent community be able to put some scientific weight behind the many claims made over the last few years for “intelligent agents”.

So, how do current systems measure up to the challenges previously identified within the interface agent field?

Knowing the user

Supervised machine learning techniques require large labelled document corpuses (with 100,00's of documents) to be highly effective. Since most users will not have 100,000 examples of what they like, interface agent profile accuracy falls below what most people find acceptable. Unsupervised learning techniques do allow the web's millions of unlabelled documents to be used, but currently have poor accuracy compared with supervised learning.

Explicit user feedback allows users to label document examples, which can increase profile accuracy significantly. Unfortunately, users are typically unwilling to commit much effort to a system unless it gives them a reward in a reasonable timeframe. This problem exists with both collaborative and content-based systems.

Current systems do learn about the user, but not to the accuracy that would allow confident delegation of full control to important tasks. Today's interface agents can make useful suggestions – but they still need a human to ultimately check them.

Interacting with the user

Experiments in believable agents build realistic agents so people can interact with them in ways they feel are natural. Unfortunately, natural interaction leads to the expectation that the agent will behave in a completely natural way. When, for example, an Einstein agent appears before them people will assume they can speak to the agent as they would a human. Agent systems are nowhere near that sophisticated, so users are left disappointed or confused.

Most agent systems avoid presenting an image at all, preferring to work in the background. This is the most practical approach given today's technology, but can leave users feeling out of control of the "hidden" agents.

There is still much debate over the best way to interact with agents [93], with no sign of a conclusion in the near future. Only time, and plenty of experimental interfaces, will tell how best to proceed.

Competence in helping the user

Most interface agent work has concentrated on learning about the user, with the assumption that once an agent knows what the user wants it can provide effective help. Planning and CSCW techniques can be utilised, but experiments are required to demonstrate competence and show which techniques are best used in various types of situation.

Chapter 3 Recommender systems

Chapter summary
The problem domain recommender systems seek to solve is presented.
Common approaches to recommender systems are described, with respect to both the recommendation process and user profiling techniques.
A set of requirements any recommender systems must meet is laid out.
A review of the state of the art is conducted, both for commercial systems and those published within the research literature. Systems are categorized according to the set of requirements.
Trends in the field are identified and discussed.

Recommender systems have become popular since the mid 90's, offering solutions to the problems of information overload within the World Wide Web. There are several varieties of approach employed, each with its own benefits and drawbacks. Since recommender systems are normally grounded to solve real world problems, the field is both exciting and rewarding to business and academics alike.

Within this chapter, the problem domain within which recommender systems work is discussed and approaches used by recommender systems today are described. A set of requirements for a recommender system is defined and a review the current state of the art conducted.

3.1 The problem of information overload

The mass of content available on the World-Wide Web raises important questions over its effective use. With largely unstructured pages authored by a massive range of people on a diverse range of topics, simple browsing has given way to filtering as the practical way to manage web-based information – and this normally means search engines.

Search engines are very effective at filtering pages that match explicit queries. Unfortunately, most people find articulating what they want extremely difficult, especially if forced to use a limited vocabulary such as keywords. The result is large lists of search results that contain a handful of useful pages, defeating the purpose of filtering in the first place.

What is needed is a way to automatically filter the type of information we want (good web pages, interesting book reviews, movie titles etc.) and present only the items useful to the user at the time of the query.

3.2 Recommender systems can help

People find articulating what they want hard, but they are very good at recognizing it when they see it. This insight has led to the utilization of relevance feedback, where people rate web pages as interesting or not interesting and the system tries to find pages that match the "interesting" examples (positive examples) and do not match the "not interesting" examples (negative examples). With sufficient positive and negative examples, modern machine learning techniques can classify new pages with impressive accuracy; text classification accuracy on a par with human capability has been demonstrated in some cases [39].

Obtaining sufficient examples is difficult however, especially when trying to obtain negative examples. The problem with asking people for examples is that the cost, in terms of time and effort, of providing the examples generally outweighs the reward they will eventually receive. Negative examples are particularly unrewarding, since there could be many irrelevant items to any typical query.

Unobtrusive monitoring provides positive examples of what the user is looking for, without interfering with the users normal activity. Heuristics can also be applied to infer negative examples, although generally with less confidence. This idea has led to content-based recommender systems, which unobtrusively watch users browse the web, and recommend new pages that correlate with a user profile.

Another way to recommend pages is based on the ratings of other people who have seen the page before. Collaborative recommender systems do this by asking people to rate pages explicitly and then recommend new pages that similar users have rated highly. The problem with collaborative filtering is that there is no direct reward for providing examples since they only help other people. This leads to initial difficulties in obtaining a sufficient number of ratings for the system to be useful.

Hybrid systems, attempting to combine the advantages of content-based and collaborative recommender systems, have proved popular to-date. The feedback required for content-based recommendation is shared, allowing collaborative recommendation as well.

3.3 User profiling in recommender systems

User profiling is typically either knowledge-based or behaviour-based. Knowledge-based approaches engineer static models of users and dynamically match users to the closest model. Questionnaires and interviews are often employed to obtain this domain knowledge. Behaviour-based approaches use the users behaviour itself as a model, often using machine-learning techniques to discover useful patterns of behaviour. Some sort of behavioural logging is usually employed to obtain the data necessary from which to extract behavioural patterns.

Kobsa [36] provides a good survey of user modelling techniques.

The typical user profiling approach for recommender systems is behavioural-based, using a binary model (two classes) to represent what users find interesting and uninteresting. Machine-learning techniques are then used to assess potential items of interest in respect to the binary model. There are a lot of effective machine learning algorithms based on two classes. Sebastiani [89] provides a good survey of current machine learning techniques.

3.4 Recommender system requirements

There are five main issues a recommender system must address (Figure 3.1 lists them all). A knowledge acquisition technique must be employed to gather information about the user from which a profile can be constructed. This knowledge is processed to provide the basis for an individual's user profile; it must also be represented in a convenient way. There must be a knowledge source from which items can be recommended. Since recommender systems are collaborative in nature, information will be shared among the users to enhance the overall recommendation performance; this shared information must be clearly defined. The final requirement is for an appropriate recommendation technique to be employed, allowing recommendations to be formulated for each of the users of the system.

Knowledge can either be explicitly or implicitly acquired from the user. Implicit knowledge acquisition is often the preferred mechanism since it has little or no impact on the users normal work activity. Unobtrusive monitoring of the user yields behavioural data about the users normal work activity over a period of time. This data can be used to imply preferences for frequently occurring items. Heuristics can also be employed to infer facts from existing data (which itself can be implicitly or explicitly obtained). Implicitly acquired knowledge usually requires some degree of interpretation to understand the users real goals. This is inherently an error prone process, reducing the overall confidence in the derived user profiles.

Explicit knowledge acquisition requires the user to interrupt their normal work to provide feedback or conduct some sort of programming. Explicit knowledge is generally accurate information, since it is provided by the users themselves and not acquired from indirect inference. Feedback types include item relevance, interest and quality. User programming occurs when the user is asked to create filter rules (either visually or via a programming language) or tell the system about groups or categories of items that exist in the domain.

For the purposes of recommendation, it is common to share user feedback as well as domain knowledge. If collaborative filtering is to be used, other users feedback on unseen items will be used as the basis of recommendations for a particular user. Examples of interesting items can be shared between similar users to increase the size of the training set and hence improve classification accuracy. Previous navigation patterns are also useful to share, as they allow new users to receive the benefit from other people's previous mistakes and successes.

Domain knowledge is often shared since it is generally static in nature and simply loaded into the system. Categorizations of items are used to provide order to a domain, and common sets of domain heuristics (sometimes part of a knowledge base) can be useful when computing recommendations.

Profiles are often represented as a feature vector. This is a standard representation and allows easy application of machine learning techniques when formulating recommendations. For content-based recommendation the features in the vectors may be the term (word) frequencies of interesting documents to a user, while for collaborative filtering the features could be the keywords commonly used by users in their search queries. Navigation trails can be used to represent time variant user behaviour. If some initial knowledge engineering has been conducted there may also be static knowledge about the users (normally held within a knowledge-base) available for a profile.

The domain itself will contain sources of information that must be recommended to the users. These could be from a database held by the recommender system (such as movie titles or the pages of a web site) or available dynamically via the web (such as

links from the currently browsed page). Other systems rely on external events, such as incoming emails, to provide items for recommendation.
There is a wide variety of recommendation techniques employed today, with most techniques falling into three broad categories. Rule filters apply heuristics to items to rank them in order of potential interest. Machine learning techniques employ similarity matching between domain items and the user's profile to rank items in order of interest. Collaborative filtering finds similar users and recommends items they have seen before and liked.

To summarize, the requirements for a recommender system are:

- Knowledge acquisition technique
    - Implicit
        - Monitoring behaviour
        - Heuristics to infer information
    - Explicit
        - User feedback
        - User programming
            - Filter rules
            - Creating groups or categories
- Shared information
    - User feedback
        - Item feedback (quality ratings, interest/relevance ratings)
        - Examples of items
        - Navigation history
    - Domain knowledge
        - Item groups / categorizations
        - Heuristics to filter items
- Profile representation
    - Vector space model of interests
    - Navigation trials
    - Static knowledge-based profile
- Knowledge source
    - Internal database of items
    - Crawled links from the web
    - External domain events
- Recommendation technique
    - Filter rules
    - Similarity matching of content to profile
    - Collaborative filtering

Figure 3.1 : Recommender systems requirements

3.5 Classification of recommender systems

Table 3.1 lists the recommender systems reviewed in the previous section and shows how they are classified using the requirements detailed previously. This provides a clear representation of the recommender system domain as it is today (entries are sorted by knowledge acquisition technique). Entries in table 3.1 marked by a "." are due to the information not being available. This is common with commercial systems that hold their technologies secret to maintain commercial advantage. The commercial systems are highlighted in a bold typeface.

For completeness, the Quickstep and Foxtrot recommender systems are included within table 3.1. Details of these experimental systems can be found in chapters 4 to 6.

	Knowledge acquisition technique					Shared information					Profile representation			Knowledge source			Recommendation technique		
	Monitoring behaviour	Heuristics to infer information	User feedback	Filter rules	User created groups/categories	Item feedback	Examples of items	Navigation history	Item groups/categories	Heuristics	Vector space model	Navigation trials	Static knowledge-based profile	Internal database of items	Crawled links from web	External domain events	Filter rules	Similarity matching	Collaborative filtering
Comm'ty search ass'nt	o							o				o		o					o
ELFI	o						o				o			o				o	
Foxtrot	o		o			o	o	o			o	o		o				o	o
MEMOIR	o		o					o				o		o					o
ProfBuilder	o							o			o	o		o				o	o
Quickstep	o		o		o	o	o	o	o		o	o		o				o	o
Referral Web	o	o							o					o	o	o	o		
SOAP	o	o	o			o					o			o					o
Tapestry	o			o				o		o				o		o	o		o
Expertise Recom'er		o	o	o	o	o					o		o	o			o		
PHOAKS		o												o			o		
Siteseer		o					o							o					o
SurfLen		o						o						o			o		
Amazon.com			o			o		o			.	.	.	o					o
Campiello			o			o					o			o					o
CDNOW			o			o		o			.	.	.	o					o
eBay			o			o					.	.	.	o					o
EFOL			o		o	o			o					o					o
Fab			o			o			o		o			o	o			o	o
GroupLens			o			o								o					o
ifWeb			o			o					o				o			o	
Levis			o			o					.	.	.	o					o
Moviefinder.com			o			o					.	.	.	o					o
Reel.com			o			o					.	.	.	o					o
Ringo			o			o					o			o					o
Virtual rev's (Tatemura)			o		o	o					o			o					o

Table 3.1 : Classification of recommender systems

Most recommender systems today explicitly ask users for feedback (interest, ratings, etc.), and share that feedback to provide collaborative recommendation from an internal database of items (products, cached web pages etc.).

This trend is understandable when you consider the technology available. Eliciting user feedback is an optional task and normally requires little effort from the user. The feedback device is often associated with the reward (e.g. a feedback control next to a recommended web page) to encourage participation.

Most casual users are reluctant to either register or install software, making monitoring of their behaviour difficult. It is also unlikely that users will accept the invasion of privacy that occurs with monitoring unless the reward offered is substantial. Heuristics can successfully infer information about users but they normally need a database of data to work on (be it from existing logs or monitoring). Filter rules are generally too complex to define in any useful detail, and creating categories or groups for a system requires another substantial investment of effort for little immediate reward.

Perhaps the most compelling evidence for the benefits of a simple feedback/collaborative filtering approach is the marketplace. All the commercial recommender systems reviewed use this technology. It seems clear that this approach is the only one to have yet reached maturity.

This is not the full story however. While unsuitable for the type of mass-market operation the commercial systems are targeting, other approaches would work for a smaller user base (corporate down to a group of individuals).

Monitoring user behaviour can be useful in a corporate style set-up, where software is installed for everyone and computers are used for work purposes only. Several of the reviewed systems do indeed use monitoring techniques, which tend to share navigation histories and implicit user feedback. Here a mix of collaborative and content-based approaches to recommendations are seen, with the content-based systems using vector-space models to perform similarity matches between the domain content and user profiles.

For information systems with large databases of information, heuristics can be used to infer knowledge about the users. If a large corpus of information exists about users and their relationships, this can be mined and recommendations created from it. User relationships can found from mail messages, newsgroup archives can be mined for web references, bookmarks utilized etc. These systems tend use filter rules to select appropriate items for recommendation, avoiding the need for user feedback completely.

3.6 Conclusion

Current recommender systems use a wide variety of techniques, often adopting a hybrid approach with both collaborative and contend-based filtering techniques being employed. Most commercial systems however use a very similar subset of the potential techniques available, concentrating on offering a collaborative filtering service to sell products.

In the next few chapters, two experimental recommender systems are detailed and some evaluations performed to measure their effectiveness. The results of the evaluations are compared to the systems reviewed. In addition to measuring the overall performance of these recommender systems, each system tests a single hypothesis using a comparison type experiment. These experiments give some insight into the profiling process within a recommender system, and provide some guidance for future recommender systems.

Chapter 4 The Quickstep recommender system

Chapter summary
The Quickstep problem domain is presented.
An overview of the Quickstep system is detailed, and the proposed empirical evaluation summarized (chapter 5 addresses these experiments in more detail).
Detailed descriptions of the approaches used by Quickstep are laid out.
The detailed design of the Quickstep system is presented using data flow diagrams to describe the multi-process architecture Quickstep uses.

Quickstep is an experimental recommender system. It addresses a real world problem and assesses the effectiveness of using domain knowledge, in the form of a research paper ontology, within the profiling process.
The Quickstep system is also described in the K-CAP publication [57].

4.1 The Quickstep problem domain

As the trend to publish research papers on-line increases, researchers are increasingly using the web as their primary source of papers. Typical researchers need to know about new papers in their general field of interest, and older papers relating to their current work. In addition, researchers time is limited, as browsing competes with other tasks in the work place. It is this problem the Quickstep recommender system addresses.
Since researchers have their usual work to perform, unobtrusive monitoring methods are preferred else they will be reluctant to use the system. Also, very high recommendation accuracy is not critical as long as the system is deemed useful to them.
Evaluation of real world knowledge acquisition systems [92] is both tricky and complex. A lot of evaluations are performed with user log data (simulating real user activity) or with standard benchmark collections. Although these evaluations are useful, especially for technique comparison, they must be backed up by real world studies so we can see how the benchmark tests generalize to the real world setting. Similar problems are seen in the agent domain where, as Nwana [65] argues, it has yet

to be conclusively demonstrated if people really benefit from such information systems.

This is why a real problem has been chosen upon which to evaluate the Quickstep recommender system.

4.2 Overview of the Quickstep system

Quickstep unobtrusively monitors user browsing behaviour via a proxy server, logging each URL browsed during normal work activity. A machine-learning algorithm classifies browsed URLs overnight, and saves each classified paper in a central paper store. Explicit feedback and browsed topics form the basis of the interest profile for each user.

Each day a set of recommendations is computed, based on correlations between user interest profiles and classified paper topics. Any feedback offered on these recommendations is recorded when the user looks at them.

Users can provide new examples of topics and correct paper classifications where wrong. In this way the training set improves over time.

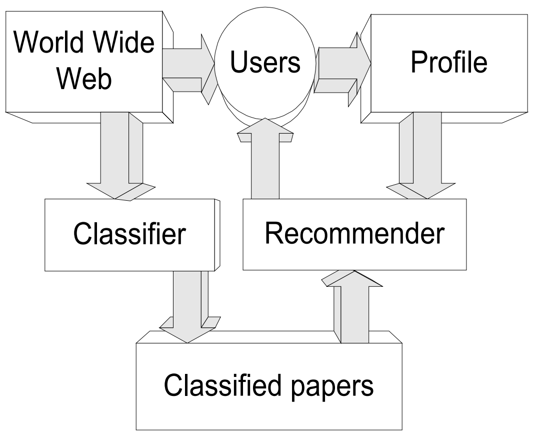


Figure 4.1 : The Quickstep system

4.3 Empirical evaluation

The current literature lacks many clear results as to the extent knowledge-based approaches assist real-world systems, where noisy data and differing user opinions exist. For this reason we decided to compare the use of an ontology against a simple flat list, to provide some empirical evidence as to the effectiveness of this approach.

Two experiments are detailed in the next chapter. The first has 14 subjects, all using the Quickstep system for a period of 1.5 months. The second has 24 subjects, again over a period of 1.5 months.

Both experiments divide the subjects into two groups.

The first group uses a flat, extensible list of paper topics. Any new examples, added via explicit feedback, use this flat list to select from. The users are free to add to the list as needed.

The second group uses a fixed size topic ontology (based on the dmoz open directory project hierarchy [19]). Topics are selected from a hierarchical list based on the ontology. Interest profiles of this group take into account the super classes of any browsed topics.

Performance metrics are measured over the duration of the trial, and the effectiveness of both groups compared.

4.4 The Quickstep system approach

Quickstep is a hybrid recommendation system, combining both content-based and collaborative filtering techniques. Since both web pages and user interests are dynamic in nature, catalogues, rule-bases and static user profiles would quickly become out of date. A recommender system approach thus appeared well suited to the problem.

Explicit feedback on browsed papers would be too intrusive, so unobtrusive monitoring is used providing positive examples of pages the user typically browses. Many users will be using the system at once, so it is sensible to share user interest feedback and maintain a common pool of labelled example papers (provided by the users as examples of particular paper topics).

Since there are positive examples of the kind of papers users are interested in, a labelled training set is available. This is ideal for supervised learning techniques, which require each training example to have a label (the labels are then used as classification classes). The alternative, unsupervised learning, is inherently less accurate since it must compute likely labels before classification (e.g. clustering techniques). A term vector representation is used to represent research papers, a common approach in machine learning [62]. A term vector is a list of word weights, derived from the frequency that the word appears within the paper.

A binary classification approach could have been used, with classes for "interesting" and "not interesting". This would have led to profiles consisting of two term vectors, one representing the kind of thing the user is interested in (computed from the positive examples) and the other what the user is not interested in (computed from the negative examples). Recommendations would be those page vectors that are most similar to the interesting class vector and least similar to the not-interesting class vector. The binary case is the simplest class representation, and consequently produces the best classification results when compared with multi-class methods.

One problem with such a representation is that the explicit knowledge of which topics the user is interested in is lost, making it hard to benefit from any prior knowledge we may know about the domain (such as the paper topics). With Quickstep a multi-class representation has been chosen, with each class representing a research paper topic. This allows profiles that consist of a human understandable list of topics (each class representing a topic). The classifier assigns each paper a class based on which class vector it is most similar to. Recommendations are selected from papers classified as belonging to a topic of interest.

The profile itself is computed from the correlation between browsed papers and paper topics. This correlation leads to a topic interest history, and a time-decay function allows current topics to be computed. A more complex function, such as polynomial curve fitting or a machine learning technique, would have trouble discriminating between the multiple interests people have.

4.4.1 Research paper representation

Research papers are represented as term vectors, with term frequency / total number of terms used for a terms weight. To reduce the dimensionality of the vectors, frequencies less than 2 are removed, standard Porter stemming [75] applied to remove word suffixes and the SMART [94] stop list used to remove common words such as "the". These measures are commonly used in information systems; [101] and [29] provide a good discussion of these issues.

To give a rough idea of the size of the vectors, 10-15,000 terms were used in the trials with training set sizes of about 200 vectors. Because of the training set size, further dimensionality reduction was not deemed necessary. Had more dimensionality reduction been needed, term frequency-inverse document frequency (TF-IDF) weighting is useful (term weights below a threshold being removed) and latent semantic indexing (LSI) could also have been used.

4.4.2 Research paper classification

The classification requirements are for a multi-class learning algorithm learning from a multi-labelled training set. To learn from a training set, inductive learning is required. There are quite a few inductive learning techniques to choose from, including information theoretic ones (e.g. Rocchio classifier), neural networks (e.g. backpropagation), instance-based methods (e.g. nearest neighbour), rule learners (e.g. RIPPER), decision trees (e.g. C4.5) and probabilistic classifiers (e.g. naive Bayes). Multiple classifier techniques such as boosting exist as well, and have been shown to enhance the performance of individual classifiers.

After reviewing and testing many of the above options, a nearest neighbour technique as chosen. The nearest neighbour approach is well suited to the problem, since the training set must grow over time and consists of multi-class examples. Nearest neighbour algorithms also degrade well, with the next closest match being reported if the correct one is not found. The IBk algorithm [1] was chosen as it outperformed naive Bayes and a J48 decision tree in our tests. The boosting technique AdaBoostM1 [22] is also used, as it works well for multi-class problems if the boosted classifier is strong enough. In tests, the boosting algorithm always improved the base classifiers performance.

Being a nearest neighbour algorithm, IBk stores instances of example paper vectors in memory. To classify a new paper, the vector distance from each example instance is calculated, and the closest neighbours returned as the most likely classes. Inverse distance weighting is used to decrease the likelihood of choosing distant neighbours.

AdaBoostM1 extends AdaBoost to handle multi-class cases since AdaBoost itself is a binary classifier. AdaBoostM1 repeatedly runs a weak learning algorithm (in this case the IBk classifier) for a number of iterations over various parts of the training set. The classifiers produced (specialized for particular classes) are combined to form a single composite classifier at the end.

4.4.3 Profiling algorithm

The profiling algorithm performs correlation between the paper topic classifications and user browsing logs. Whenever a research paper is browsed that has a classified topic, it accumulates an interest score for that topic. Explicit feedback on recommendations also accumulates interest values for topics. The current interest of a topic is computed using the inverse time weighting algorithm below, applied to the user feedback instances.

$$\text{Topic interest} = \sum_{1..\text{no of instances}}^{n} \text{Interest value}(n) \,/\, \text{days old}(n)$$

Interest values	Paper browsed = 1
	Recommendation followed = 2
	Topic rated interesting = 10
	Topic rated not interesting = -10

The profile for each user consists of a list of topics and the current interest values computed for them (see below). The interest value weighting was chosen to provide sufficient weight for an explicit feedback instance to dominate for about a week, but after that browsed URL's would again become dominant. In this way, the profile will adapt to changing user interests as the trial progresses.

```
Profile = (<user>,<topic>,<topic interest value>)*

e.g.  ((someone,hypertext,-2.4)
       (someone,agents,6.5)
       (someone,machine learning,1.33))
```

If the user is using the ontology based set of topics, all super classes gain a share when a topic receives some interest. The immediate super class receives 50% the main topics value. The next super class receives 25% and so on until the most general topic in the is-a hierarchy is reached. In this way, general topics are included in the profile rather than just the most specific ones, producing a more rounded profile.

4.4.4 Recommendation algorithm

Recommendations are formulated from a correlation between the users current topics of interest and papers classified as belonging to those topics. A paper is only recommended if it does not appear in the users browsed URL log, ensuring that recommendations have not been seen before. For each user, the top three interesting topics are selected with 10 recommendations made in total (making a 4/3/3 split of recommendations). Papers are ranked in order of the recommendation confidence before being presented to the user.

$$\text{Recommendation confidence} = \text{classification confidence} * \text{topic interest value}$$

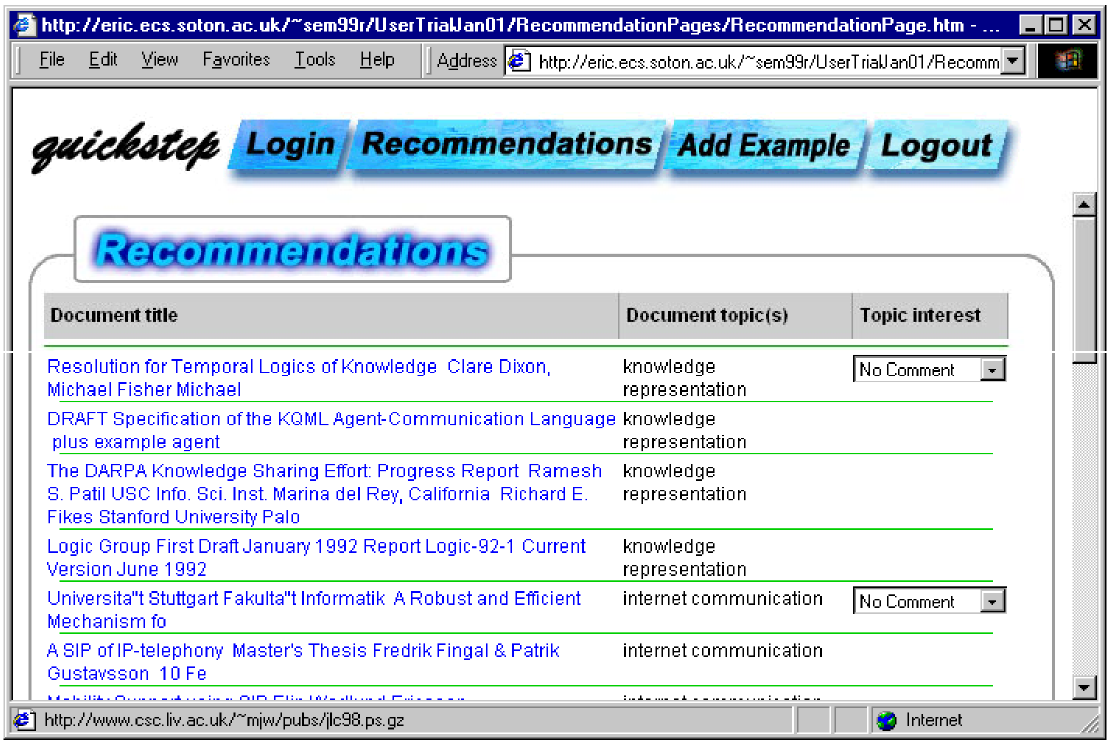
The classification confidence is computed from the AdaBoostM1 algorithm's class probability value for that paper (somewhere between 0 and 1).

4.4.5 Research paper topic ontology

The research paper topic ontology is based on the dmoz [19] taxonomy of computer science topics. It is an is-a hierarchy of paper topics, up to 4 levels deep (e.g. an "interface agents" paper is-a "agents" paper). Pre-trial interviews formed the basis of which additional topics would be required. An expert review by two domain experts validated the ontology for correctness before use in the trials.

4.4.6 Feedback and the quickstep interface

Recommendations are presented to the user via a browser web page. The web page applet loads the current recommendation set and records any feedback the user provides. Research papers can be jumped to, opening a new browser window to display the paper URL. If the user likes/dislikes the paper topic, the interest feedback combo-box allows "interested" or "not interested" to replace the default "no comment".

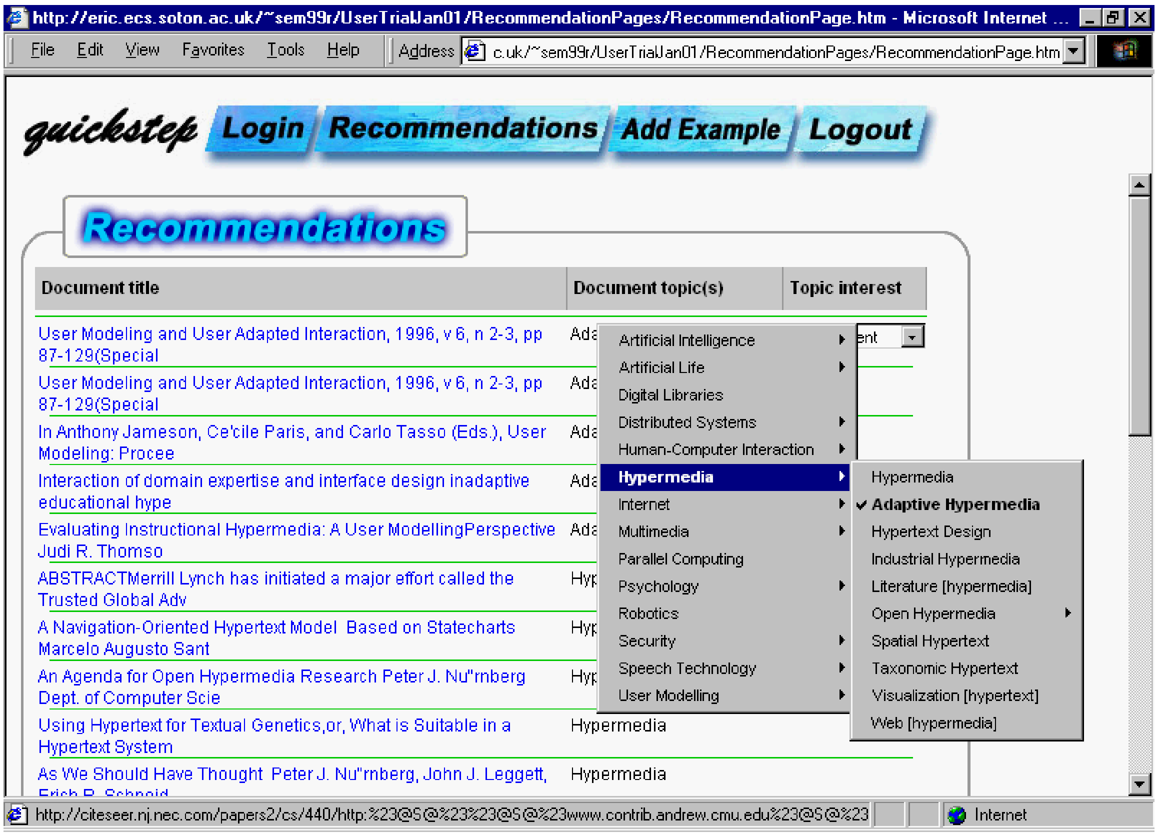


http://eric.ecs.soton.ac.uk/~sem99r/UserTrialJan01/RecommendationPages/RecommendationPage.htm - ...
File Edit View Favorites Tools Help
Address http://eric.ecs.soton.ac.uk/~sem99r/UserTrialJan01/Recomm
quickstep
Login
Recommendations
Add Example
Logout
Recommendations
http://www.csc.liv.ac.uk/~mjw/pubs/jlc98.ps.gz
Internet

Document title	Document topic(s)	Topic interest
Resolution for Temporal Logics of Knowledge Clare Dixon, Michael Fisher Michael	knowledge representation	No Comment
DRAFT Specification of the KQML Agent-Communication Language plus example agent	knowledge representation	
The DARPA Knowledge Sharing Effort: Progress Report Ramesh S. Patil USC Info. Sci. Inst. Marina del Rey, California Richard E. Fikes Stanford University Palo	knowledge representation	
Logic Group First Draft January 1992 Report Logic-92-1 Current Version June 1992	knowledge representation	
Universita"t Stuttgart Fakulta"t Informatik A Robust and Efficient Mechanism fo	internet communication	No Comment
A SIP of IP-telephony Master's Thesis Fredrik Fingal & Patrik Gustavsson 10 Fe	internet communication	

Figure 4.2 : Quickstep's web-based interface

The topic of the paper can be changed by clicking on the topic and selecting a new one from the popup list. The ontology group has a hierarchical popup menu, the flat list group has a single level popup menu.


http://eric.ecs.soton.ac.uk/~sem99r/UserTrialJan01/RecommendationPages/RecommendationPage.htm - Microsoft Internet ...

Document title	Document topic(s)	Topic interest
User Modeling and User Adapted Interaction, 1996, v 6, n 2-3, pp 87-129(Special	Ada	ent
User Modeling and User Adapted Interaction, 1996, v 6, n 2-3, pp 87-129(Special	Ada	
In Anthony Jameson, Ce'cile Paris, and Carlo Tasso (Eds.), User Modeling: Procee	Ada	
Interaction of domain expertise and interface design inadaptive educational hype	Ada	
Evaluating Instructional Hypermedia: A User ModellingPerspective Judi R. Thomso	Ada	
ABSTRACTMerrill Lynch has initiated a major effort called the Trusted Global Adv	Hyp	
A Navigation-Oriented Hypertext Model Based on Statecharts Marcelo Augusto Sant	Hyp	
An Agenda for Open Hypermedia Research Peter J. Nu"rnberg Dept. of Computer Scie	Hyp	
Using Hypertext for Textual Genetics,or, What is Suitable in a Hypertext System	Hypermedia	
As We Should Have Thought Peter J. Nu"rnberg, John J. Leggett, Erich R. Schneid	Hypermedia	

http://citeseer.nj.nec.com/papers2/cs/440/http:%23@S@%23%23@S@%23www.contrib.andrew.cmu.edu%23@S@%23

Figure 4.3 : Changing paper topics in Quickstep

New examples can be added via the interface, with users providing a paper URL and a topic label. These are added to the groups training set, allowing users to teach the system new topics or improve classification of old ones.

All feedback is stored in log files, ready for the profile builder's daily run. The feedback logs are also used as the primary metric for evaluation. Interest feedback, topic corrections and jumps to recommended papers are all recorded and time stamped.

4.4.7 Design choices made in the Quickstep system

Since an increasing number of research papers are published in postscript and PDF format, it was decided to only deal with these formats (along with gzipped, zipped and Z compressed versions). The reason was to filter noisy HTML pages, of which only a

fraction are research papers, and thus make the classification task easier. The drawback is research areas that publish primarily in HTML will be ignored.
The Weka machine learning libraries [106] are used to implement the AdaBoostM1 classification algorithm and the IBk classifier.
Through empirical evaluation, 100 boosting iterations with 5% of the training set used per iteration proved best and was chosen for the trial. A k value of 5 was selected for the boosted IBk classifier.
The users were asked, in a pre-trial interview, to provide a few bookmarks to publication pages of important authors. The bookmarks provided the Quickstep web crawler algorithm with somewhere to look initially. An initial set of papers was thus loaded into the system, making a pool of classified papers that could be recommended.
In addition to bookmarks, a manual bootstrap training set was created for each of the topics mentioned in the pre-trial interview. The use of a bootstrap training set reduces the burden on users to train the system before it becomes useful. Both the ontology group and the flat list group started both trials with an identical bootstrap training set, which was then allowed to diverge as the trial progressed.
To keep the trial simple (both to develop and analyse), feedback on the quality of individual papers was not elicited, only feedback on the interest of the paper topic to the user.

Chapter 5 Experimental evaluation with Quickstep

Chapter summary
Experimental set-up is described along with subjects selection, experimental conditions and the metrics recorded.
A causation analysis of the effect being measured is presented.
The experimental data is detailed and significant trends identified.
The trends seen are discussed and hypotheses offered for the effects seen. Comparison is made with other published data from similar systems.
Conclusions are drawn.

Two experiments have been performed using the Quickstep system. Both compared the use of a flat list of paper topics to the use of a hierarchical "is-a" taxonomy. Profiling effectiveness was measured, as well as the overall usefulness of the system. Evaluation of the Quickstep system is also described in the K-CAP publication [57].

5.1 Details of the two trials

Two trials were conducted to assess empirically both the overall effectiveness of the Quickstep recommender system and to quantify the effect made by use of the ontology.

The first trial used 14 subjects, consisting of researchers from the IAM research laboratory. A mixture of 2nd year postgraduates up to professors was taken, all using the Quickstep system for a duration of 1.5 months.

The second trial used 24 subjects, 14 from the first trial and 10 more 1st year postgraduates, and lasted for 1.5 months. Some minor interface improvements were made to make the feedback options less confusing.

The pre-trial interview obtained details from subjects such as area of interest and expected frequency of browser use.

The purpose of the two trials was to compare a group of users using an ontology labelling strategy with a group of users using a flat list labelling strategy. Subject selection for the two groups balanced the groups as much as possible, evening out topics of interest, browser use and research experience (in that order of importance).

Both groups had the same number of subjects in them (7 each for the pilot trial, 12 each for the main trial).

In the first trial, a bootstrap of 103 example papers covering 17 topics was used.

In the second trial, a bootstrap of 135 example papers covering 23 topics was used. The bootstrap training set was updated to include examples from the final training sets of the first trial. The first trials classified papers were also kept, allowing a bigger initial collection of papers to recommend from in the second trial.

Both groups had their own separate training set of examples, which diverged in content as the trial progressed. The classifier was run twice for each research paper, classifying once with the flat list groups training set and once with the ontology groups training set. The classifier algorithm was identical for both groups; only the training set changed.

The system interface used by both groups was identical, except for the popup menu for choosing paper topics. The ontology group had a hierarchical menu (using the ontology); the flat list group had a single layer menu.

The system recorded the times the user declared an interest in a topic (by selecting "interesting" or "not interesting"), jumps to recommended papers and corrections to the topics of recommended papers. These feedback events were date stamped and recorded in a log file for later analysis, along with a log of all recommendations made. Feedback recording was performed automatically by the system, whenever the subjects looked at their recommendations.

A post-trial questionnaire was filled out after each trial, asking qualitative questions about the Quickstep system.

5.2 Causation analysis

Table 5.1 shows an analysis of possible causations of differences between the two subject groups. The effect the causation may have is described, and the measures taken to reduce the significance of the effect detailed.

Causation	Effect	Significance to trial / measures taken to remove causation
Choice of bootstrap training set	An initial training set that's good in one area, will bias the recommendations in that area and hence improve the system in that area via a positive feedback loop.	The training set should contain an equal number of documents for each of the users research areas. An expert review should be performed on them to ensure they are equal in quality.
Interface ease of use	The harder the interface is to use, the less use the users will make of it. This will reduce feedback, hence recommendation accuracy and usefulness in a negative loop.	Both groups have the same interface, so this will not cause a bias with comparisons. An expert review will be performed on the interface to try to catch problems pre-trial, and hence encourage more feedback from users in general.
Learning bias towards different domains	The domain may feature term occurrence patterns that are well/ill-suited to the learning technique. This will effect recommendation accuracy and hence usefulness.	Subject interest domains are balanced between the two groups, so comparison is valid. Absolute comparisons with other recommenders can only be made with high validity against similar domains.
Learning bias towards labelling strategy	The learning technique may favour one labelling strategy, hence make better recommendations, and hence bias usefulness measurements.	Both groups use the same learning strategy. The training data format is identical, so the effect of the labelling strategy is subtle. To some extent, this is what we are measuring.
Subject browsing domain	Usefulness, accuracy etc biased to how well domain matches learning techniques.	Usefulness etc is compared (not absolutely measured) so still valid. Subject groups will have a balanced set of research interests.
Subject browsing time	The amount of browser usage effects the amount of data available to the profiler. The more data, the better the profile and hence recommendation usefulness.	Balancing work types should even out groups. The number of URL's browsed is logged, so post-trial this can be assessed to see if the groups were balanced. Like enthusiasm, analysis can thus be skewed to cope with unbalanced groups.
Subject enthusiasm	Enthusiastic subjects will use the system more, providing more feedback to learn from. This in turn improves accuracy in a positive feedback loop.	Monitor (via feedback logs) the number of times feedback was provided. Enthusiastic users can thus be identified post-trial, and the effect of this cause analysed. If one group has more enthusiastic subjects, the analysis can be adjusted with this data in mind to suggest what a balanced group would be like.
Subject experience	Experts will workaround errors increasing system usage.	Subject experience balanced between two groups during subject selection.
Subject labelling strategy	Usefulness of system.	Variable to be measured.
Subject's job / type of work	Work type (e.g. a PhD student doing a literature survey or RA undertaking system building) biases browsing style. System will perform better for document browsing (as opposed to say software downloads / bug reports).	Subject work types balanced between two groups during subject selection.
Subject's sex, race, age	Many	Balanced as much as is possible during subject selection for two groups.
Subjects browser	IE or Netscape features may hinder ease of use of system. This may reduce usefulness and enthusiasm.	IE and Netscape functionality and look and feel should be identical, removing this causation.
Subjects English language skills	Foreign document browsing will be noise for English speakers, effecting system usefulness. Browsing may be slower for non-English speakers, hence less for system to observe.	Balance English speakers during subject selection between two groups.
Subjects equipment specification	A slower computer may encourage less browsing, or less patience for users. This will reduce enthusiasm.	System minimum spec should be fine for all subjects.

Table 5.1 : Causation analysis for the Quickstep experiments

5.3 Experimental data

Since feedback only occurs when subjects check their recommendations, the data collected occurs at irregular dates over the duration of the trial. Cumulative frequency of feedback events is computed over the period of the trial, allowing trends to be seen as they develop during the trial. Since the total number of jumps and topics differ between the two groups, the figures presented are normalized by dividing by the number of topics (or recommendations) up to that date. This avoids bias towards the group that provided feedback most frequently.

Figure 5.1 shows the topic interest feedback results. Topic interest feedback is where the user comments on a recommended topic, declaring it "interesting" or "not interesting". If no feedback is offered, the result is "no comment".

Topic interest feedback is an indication of the accuracy of the current profile. When a recommended topic is correct for a period of time, the user will tend to become content with it and stop rating it as "interesting". On the other hand, an uninteresting topic is likely to always attract a "not interesting" rating. Good topics are defined as either "no comment" or "interesting" topics. The cumulative frequency figures are presented as a ratio of the total number of topics recommended. The not interesting ratio (bad topics) can be computed from these figures by subtracting the good topic values from 1.

The ontology groups have a 7 and 15% higher topic acceptance. In addition to this trend, the first trial ratios are about 10% lower than the second trial ratios.

Figure 5.2 shows the jump feedback results. Jump feedback is where the user jumps to a recommended paper by opening it via the web browser. Jumps are correlated with topic interest feedback, so a good jump is a jump to a paper on a good topic. Jump feedback is an indication of the quality of the recommendations being made as well as the accuracy of the profile. The cumulative frequency figures are presented as a ratio of the total number of recommendations made.

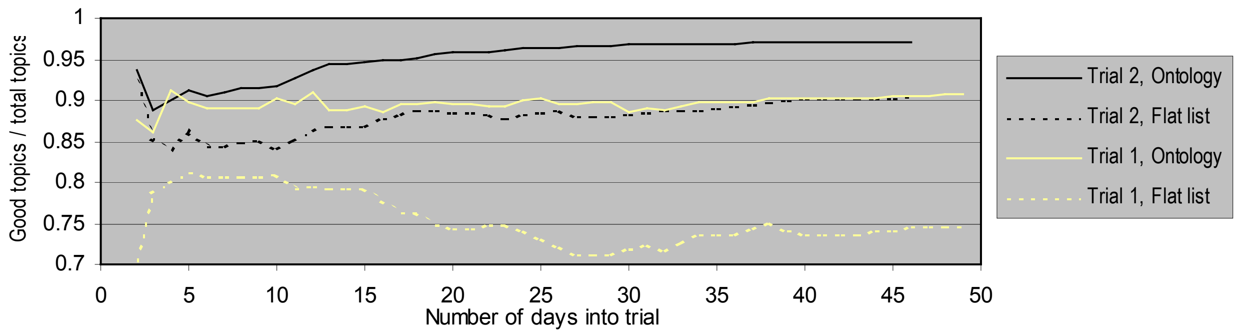


Figure 5.1 : Ratio of good topics / total topics

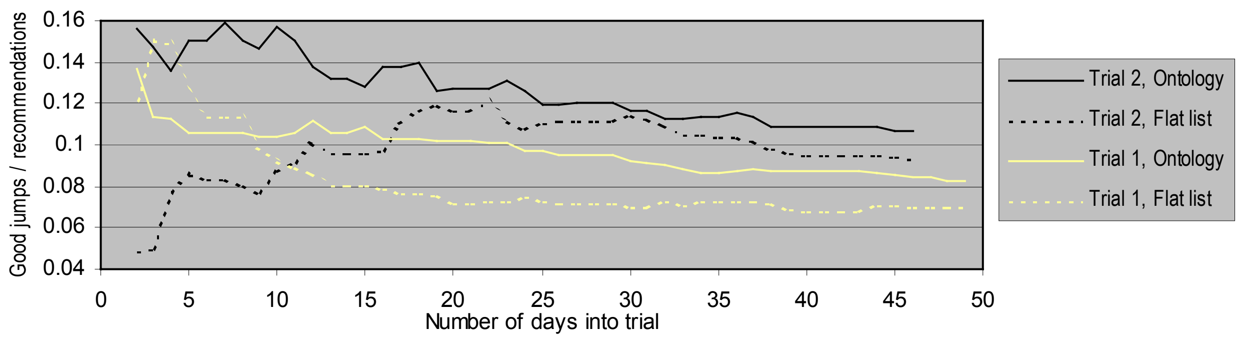


Figure 5.2 : Ratio of good jumps / total recommendations

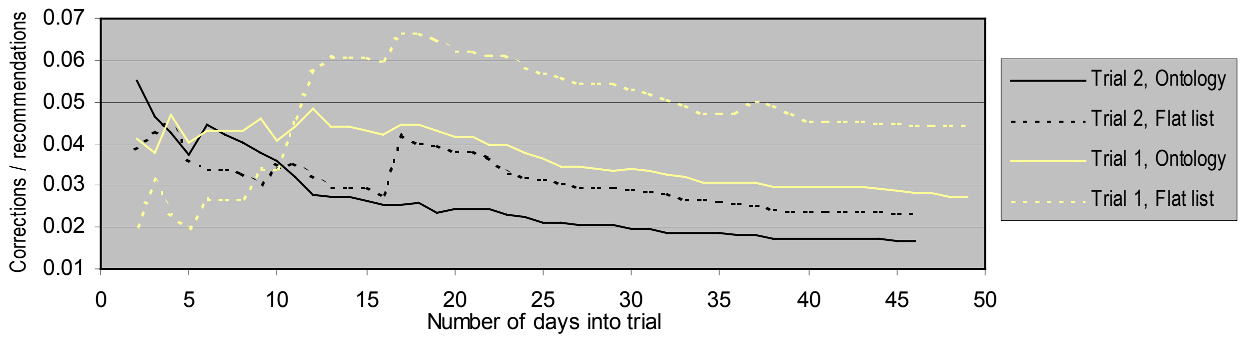


Figure 5.3 : Ratio of topic corrections / total recommendations

There is a small 1% improvement in good jumps by the ontology group. Both trials show between 8-10% of recommendations leading to good jumps.

Figure 5.3 shows the topic correction results. Topic corrections are where the user corrects the topic of a recommended paper by providing a new one. A topic correction will add to or modify a groups training set so that the classification for that group will improve. The number of corrections made is an indication of classifier accuracy. The cumulative frequency figures are presented as a ratio of the total number of recommended papers seen.

Although the flat list group has more corrections, the difference is only by about 1%. A clearer trend is for the flat list group corrections to peak around 10-20 days into the trial, and for both groups to improve as time goes on.

At the end of the first trial, the ontology group training set had 157 examples, and the flat list group had 162 examples. The paper repository had about 3000 classified research papers.

At the end of the second trial, the ontology group training set had 209 examples, and the flat list group had 212 examples. The paper repository had about 3500 classified research papers.

A cross-validation test was run on each group's final training set, to assess the precision and recall of the classifier using those training sets. The results are shown in table 5.2.

Group (trial)	Precision	Recall	Classes
Trial 1, Ontology	0.484	0.903	27
Trial 1, Flat list	0.52	1.0	25
Trial 2, Ontology	0.457	0.888	32
Trial 2, Flat list	0.456	0.972	32

Table 5.2 : Classifier recall and precision upon trial completion

5.4 Post-trial questionnaires

Each subject was given a post-trial questionnaire to fill out. The questionnaire asked users to select topics that were of interest during the trial from a list of all topics known to the system. Some 5-point scale questions were also asked, to elicit qualitative information about the usefulness of the system. A comment section was also included so improvements could be suggested to the system.
The second trials question replies are presented in table 5.3 (some subjects felt unable to answer the questions, so left some of them blank). The 5-point scale ranged from the lowest 1 to highest 5 value of answer, with textual comments associated with each value to guide the users.

Question	1	2	3	4	5	Mean
How much paper searching/reading did you do during the period of the trial?		11	8	2	2	2.78
How much did you use quickstep during the period of the trial?		10	7	6		2.83
Overall, were you interested in the topics recommended by quickstep?		4	11	7		3.14
Overall, how useful were the papers recommended by the quickstep system?		7	11	4		2.86
How accurate was quickstep at classifying papers?		1	13	8		3.32

Table 5.3 : Post-trial answers for Quickstep's second trial

The results indicate that the system recommended papers that were fairly interesting, but not quite as useful. This is to be expected since the recommendation algorithm does not consider paper quality.

5.5 Discussion of trends seen in the experimental data

From the experimental data of both trials, several suggestive trends are apparent. The initial ratios of good topics were lower than the final ratios, reflecting the time it takes for enough log information to be accumulated to let the profile settle down. The ontology users were 7-15% happier overall with the topics suggested to them.

A hypothesis for the ontology group's apparently superior performance is that the is-a hierarchy produces a rounder, more complete profile by including general super class topics when a specific topic is browsed by a user. This in turn helps the profiler to discover a broad range of interests, rather than just latching onto one correct topic.

The first trial showed fewer good topics than the second trial (about a 10% difference seen by both groups). This is probably because of interface improvements made for the second trial, where the topic feedback interface was made less confusing. Subjects were sometimes rating interesting topics as not interesting if the paper quality was poor. As there are more poor quality papers than good quality ones, this introduced a bias to not interesting topic feedback resulting in a lower overall ratio.

About 10% of recommendations led to good jumps. Since 10 recommendations were given to the users at a time, on average one good jump was made from each set of recommendations received. As with the topic feedback, the ontology group again was marginally superior but only by a 1% margin. This smaller difference is probably due to people having time to follow only 1 or 2 recommendations. Thus, although the ontology group has more good topics, only the top topic of the three recommended will really be looked at; the result is a smaller difference between the good jumps made and the good topics seen.

The flat list group has a poor correction / recommendation ratio 10-20 days into the trial. This is probably due to new topics being added to the system. Most new topics were added after the users became familiar with the system, and know which topics they feel are missing. The familiarization process appeared to take about 10 days. The classification accuracy of these new topics is poor until enough examples have been entered, typically after another 10 days.

The ontology group has about 1% fewer corrections for both trials. This is small difference may indicate the utility of imposing a uniform conceptual model of paper topics on the subjects (by using the common topic hierarchy). Classifying papers is a subjective process, and will surely be helped if people have similar ideas as to where topics fit in a groups overall classification scheme.

These preliminary results need to be extended so as to enable the application of more rigorous statistical analysis. Nevertheless, we believe the trend in the data to be encouraging as to the utility of ontologies in recommender systems.

An informal result was seen where most incorrect classifications were in the roughly correct topic area (such as a "multi-agent-systems" paper being classified as an

"interface-agents" paper). This is probably due to the nearest neighbour algorithm, and was liked by the subjects since they could see that the system, although not perfect, was working along the right lines.
When compared with other published systems, the classification accuracy figures are similar, if on the low side (primarily because of the use of multi-class classification). Nearest neighbour systems such as NewsDude [8] and Personal Webwatcher [61] report 60-90% classification accuracy based on binary classification. The higher figures tend to be seen with benchmark document collections, not real-world data. NewsWeeder [38] reports 40-60% classification accuracy using real user browsing data from two users over a period of time, so this would be the best comparison. If the number of classes classified is taken into consideration, the system compares well.
Multi-class classification is not normally applied to recommender systems making direct comparison of similar systems difficult. A comparison of the usefulness of the recommender to that of other systems was attempted, but the lack of published experimental data of this kind meant that only classification accuracy could be usefully compared.

5.6 Conclusions from the Quickstep trials

Most recommender systems use a simple binary class approach, using a user profile of what is interesting or not interesting to the user. The Quickstep recommender system uses a multi-class approach, allowing a profile in terms of domain concepts (research paper topics) to be built. The multi-class classification is less accurate than other binary classification systems, but allows class specific feedback and the use of domain knowledge (via an is-a hierarchy) to enhance the profiling process.
Two experiments were performed in a real work setting, using 14 and 24 subjects over a period of 1.5 months. The results suggest how using an ontology in the profiling process results in superior performance over using a flat list of topics. The ontology users tended to have more "rounder" profiles, including more general topics of interest that were not directly suggested. This increased the accuracy of the profiles, and hence usefulness of the recommendations.
An informal result was seen in the nearest neighbour classifier's robustness. Even when it made a mistake (60% of the time in fact) the class it chose was normally in the correct area. For example, for an "interface agent" paper the classification would more likely be "agent" than "human computer interaction". The users liked this as it

showed the system was at least making a reasonable attempt at classification, even if it was getting things wrong.

Although hard to compare, owing to the lack of a standard for reporting results, the overall recommender performance does appear to compare reasonably with other recommender systems in the literature.

Chapter 6 The Foxtrot recommender system

Chapter summary
The Quickstep system is critically examined. User feedback is summarised and a list of problems presented.
A set of requirements for the Foxtrot system is formulated from the problems seen in the Quickstep system.
Initial suggestions for techniques to use in Foxtrot are described, along with an interface suggestion. A potential profiling algorithm for using direct profile feedback is also introduced.
An experimental analysis of the Foxtrot system is suggested.
A Foxtrot design is suggested.

Foxtrot is the next evolution of the Quickstep system. It is still being designed, but the requirements and initial approach are presented here.

6.1 Problems with the Quickstep system

All the subjects on the two Quickstep trials filled in a post-trial questionnaire. One of the questions on the questionnaire allowed users to provide feedback as to what could be improved with the Quickstep system as it is. This feedback has been analysed and used to formulate the problems seen in table 6.1.

Category	Quickstep problems as reported by the subjects
Classification	The subjects suggested a finer grain classification. The topics provided were too broad by about 1 level.
Feedback	The interface did not provide enough feedback when subjects provided both relevance feedback and topic feedback. If the users, for example, stated they were interested in a paper topic and they re-checked their recommendations a few hours later, the "interesting" feedback is visually "lost". It is recorded by the system, but the subjects have no way to know this from the visual feedback.
	A way to provide more direct feedback on the profile was suggested. Some subjects wanted to tell the system from the start what they were interested in, rather than have it guess.
Interface	Subjects felt the interface was not very intuitive, particularly the way feedback was provided. The difference between paper interest and paper topic interest was not appreciated, causing a few mistakes when providing feedback. The second trial interface change helped, but did not fully solve this issue.
	Some subjects suggested a search facility would be useful, so specific papers could be found rather than having to wait for them to be recommended.
	Email notification of recommendations was suggested.
	A "proxy last used" date was suggested for the interface to tell users if their browser was using the proxy. This is useful when the proxy has been turned off and not turned back on again (due to forgetfulness).
Proxy	The proxy was widely thought of as too slow and unreliable. Complex web pages were not supported, which encouraged subjects to turn if off and then forget to turn it back on again (resulting in blanks in the URL history logs).
Reliability	Some URL's became dead after they had been processed by the system. This left subjects tantalised by a paper they could not read. Caching papers was suggested.
	The subjects did not like receiving identical papers with different URL's.
	Corrupt paper titles (caused by the PS to text process failing to convert the paper) caused irritation, as the subjects could not see what the paper was about. It also caused these corrupt papers to be miss-classified. The papers themselves could be read as a PS document, its just the system could not extract the text to correctly process them.
Summaries	Document summaries were of insufficient length. Some subjects wanted to see the abstract of the paper before they downloaded it, so they would know if it is worth reading.
	Bibliographic references were suggested for the papers, so subjects know where the papers came from (and hence can cite them). Other services (e.g. Google search, research index) can do this but it was inconvenient.
Other	HTML support was suggested, as some research areas published primarily in HTML.

Table 6.1 : Quickstep problems reported by subjects on both trials

6.2 Requirements for the Foxtrot system

Foxtrot is the next evolution of the Quickstep system. In order to structure the development of the Foxtrot system some requirements must be identified. Once clearly identified, a design for the Foxtrot system can be formulated and the system built allowing empirical evaluation to be performed.

One of the key insights gained from the Quickstep system is that by using domain concepts the recommender system can communicate its ideas in terms familiar to its users. This was achieved in Quickstep by the simple use of research paper topics instead of a binary interest model. What if this communication was taken a step further, and the actual internal profile of the recommender system communicated in a way understandable to the users? This would allow users to provide quality feedback directly on the profile, and hence improve further still the profile accuracy. It is this thesis that the Foxtrot system will try to test.

Foxtrot should visualize profiles in terms users will understand and allow users to provide direct, high quality feedback on the profile itself.

In addition to this main requirement, several other requirements can be formulated from the post-trial questionnaire finds of the Quickstep experiments.
HTML page support should be included. While making the problem domain easier, filtering just PS and PDF papers is clearly inadequate for research areas that publish primarily in HTML. Some heuristics needs to be created to identify when a HTML page is a research paper, which should filter out the majority of irrelevant HTML pages.

Foxtrot should support HTML papers.

Paper quality assessments should be elicited from users, so once an interesting topic has been discovered good quality papers can be recommended before poorer quality papers. This is clearly desirable to the users, and was commented on frequently in the post-trial questionnaires. Many recommender systems support this type of collaborative filtering and there is no reason why Foxtrot should not too.

Foxtrot should elicit paper quality assessments from users and use them to improve recommendations.

A search engine facility will be included within Foxtrot. This will encourage occasional users to utilize the system, and allow users to follow up a recommended topic and find more of the same. The database of research papers is already there in the Quickstep system, so it should be relatively easy to provide a front end to facilitate search.

Foxtrot should support a paper search facility.

The current Quickstep research paper database needs to have finer grained labels. This can be achieved by creating an extra level of depth to the paper ontology, and manually re-classifying the training set papers to use the finer grained labels. If Foxtrot is to be used as a search facility, better quality labels should greatly improve the filtering capabilities of the search engine.

Foxtrot should support a one level finer grained topic ontology than Quickstep.

The Quickstep proxy server is clearly inadequate. Foxtrot should support a quicker proxy that fully supports HTTP protocols. Using a proxy written in a compiled language (such as C++) would be a start to improve speed, along with the ability to utilize UNIX processes (as opposed to inefficient java threading). In addition to improving the proxy, the interface should indicate the proxy status so as to tell users when they have switched it off.

Foxtrot should include a proxy that fully supports HTTP and is comparable in speed to commercial firewalls. The interface should also report on the status of the proxy.

The interface must clearly show what relevance or quality feedback the users have provided in the past. This will improve the users sense of control and trust in the system, since when they provide feedback it will be visually clear that the system has accepted and used it.

Foxtrot should clearly show previous feedback (such as topic interest or paper topic feedback).

The overall interface should be made more intuitive. An expert review should be conducted to gain additional confidence in the interface before any experiments are conducted.

The Foxtrot interface should be more intuitive and an expert review of it conducted.

Heuristics should be added to the Foxtrot system to prevent garbage paper conversions being used. The pstotext UNIX utility does not report errors itself, but there are telltale signs when a paper has not been converted to text correctly. Content analysis should also be performed to prevent identical papers being loaded into the system under different URL's. Dead URL's can also be checked for.

Foxtrot should employ heuristics to prevent garbage paper conversion, duplicate papers and dead URL's.

The Quickstep document representation (URL, title, term frequencies) does not allow for abstracts. Foxtrot could extend the document format to store abstract text, and heuristics created to extract the abstract from PS papers. This will not be a formal requirement however, since it is unclear as to the benefit/development cost ratio.
Bibliographic information is not really available from just the URL's, which Quickstep used to identify papers. The development cost of creating a system to track bibliographic data down would far outweigh the benefit to users; this will not be a requirement.
Email notification is not compatible with the search engine interface metaphor that Foxtrot will present.

To summarize, table 6.2 lists the requirements for Foxtrot.

Category	Requirement
Interface	Foxtrot should visualize profiles in terms users will understand and allow users to provide direct, high quality feedback on the profile itself.
	Foxtrot should support a paper search facility.
	Foxtrot should clearly show previous feedback.
	Foxtrot should elicit paper quality assessments from users and use them to improve recommendations.
	The Foxtrot interface should be more intuitive and an expert review of it conducted.
Content	Foxtrot should support a one level finer grained topic ontology than Quickstep.
	Foxtrot should employ heuristics to prevent garbage paper conversion, duplicate papers and dead URL's.
Technical	Foxtrot should support HTML papers.
	Foxtrot should include a proxy that fully supports HTTP and is comparable in speed to commercial firewalls. The interface should also report on the status of the proxy.

Table 6.2 : Foxtrot requirements

6.3 The Foxtrot recommender system

The current proposal for the Foxtrot design involves making a search engine type interface, with recommendations offered as initial search results on start-up. This way the users can use Foxtrot as a search engine, but still view recommendations each time they visit the web page. If the users choose, they will be able to see their profile via a visualization, and provide direct feedback.

6.3.1 The Foxtrot system interface

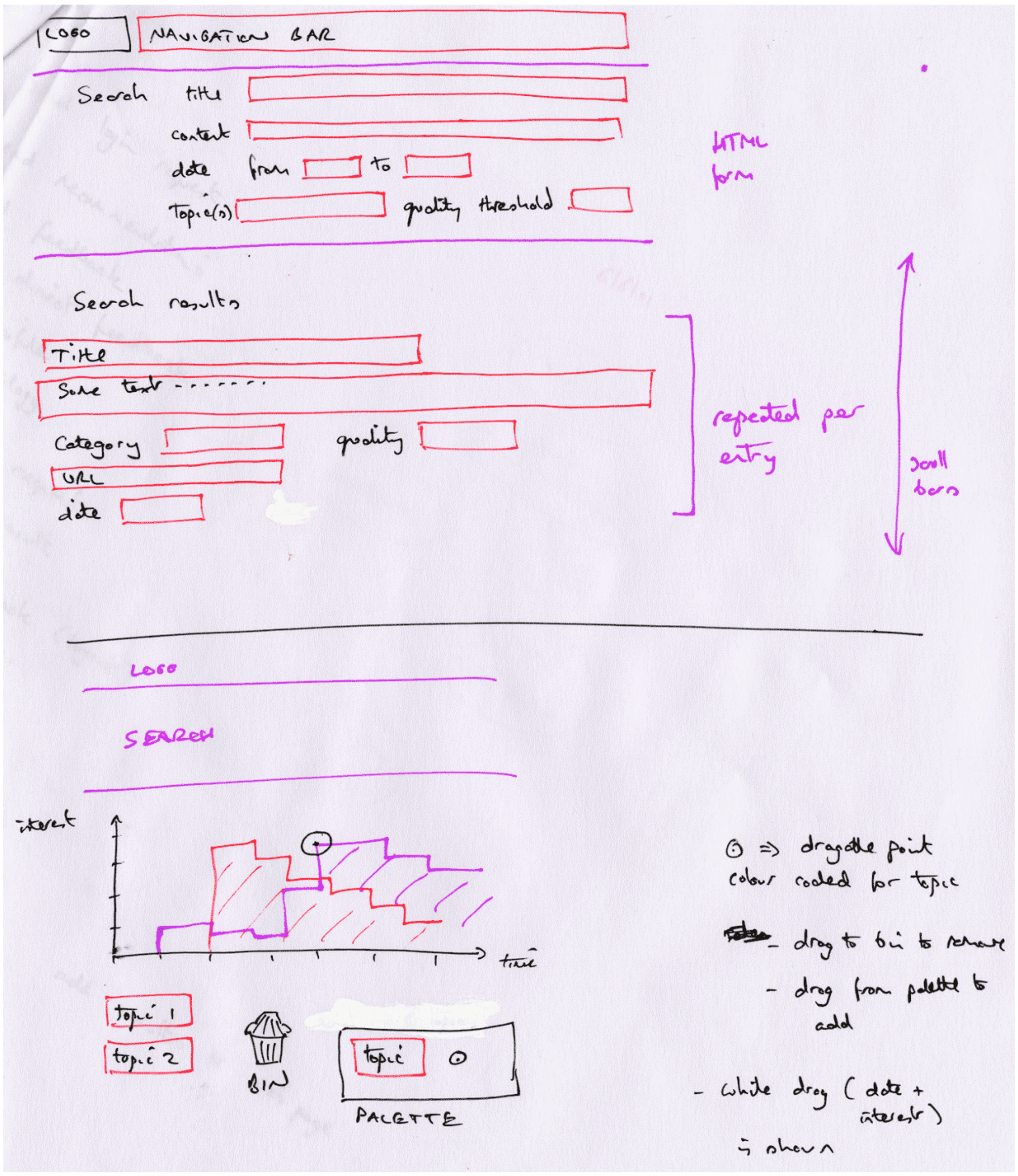
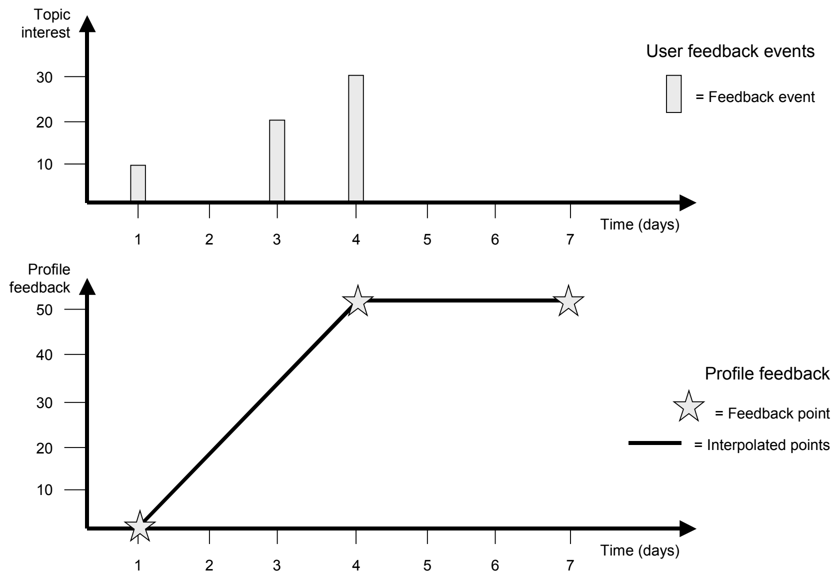


Figure 6.1 : Proposed Foxtrot interface

A proposed design for the Foxtrot interface is shown in figure 6.1. It supports a search-engine like interface, with the initial results being Foxtrot's recommendations. These will be overwritten by any search query. Results include a topic, topic interest and document quality drop-down boxes to facilitate optional user feedback. A proxy warning is displayed if the proxy log date does not correspond to the recommendation date (reminding users to turn the proxy on).

The profile can be accessed via a tab control, where it will be visualized and feedback accepted. It is displayed using a time/topic interest graph, with topics being colour coded. Absolute topic interest points can be added, by dragging them from the palette. If more than one point is added, a straight line connects the two (or more). Points can be removed by dragging them to the bin.

6.3.2 The Foxtrot profiler



Normal profiling algorithm

$$\text{Topic interest} = \sum_{1..\text{no of instances}}^{n} \text{Interest value}(n) / \text{days old}(n)$$

Profiling algorithm with direct profile feedback

```
Loop on days
    Interpolate direct profile feedback
    Compute topic interest
    Compute error adjustments
```

$$\text{Error adjustment}_{day} = \text{Interpolated profile value}_{day} - \text{Topic interest}_{day}$$

$$\text{Topic interest} = \sum_{1..\text{no of instances}}^{n} \text{Interest value}(n) / \text{days old}(n) + \sum_{1..\text{no of days}}^{n} \text{Error adjustment}(n) / \text{days old}(n)$$

Figure 6.2 : Proposed algorithm to process direct profile feedback

The profiler will use an algorithm to process direct profile feedback (as shown in figure 6.2). The points provided by the user are absolute interest points. Straight lines connect these interest points and the interest of day's in-between interpolated. The error between the current profile and the direct feedback profile is then computed, and the difference at each point added as if it were feedback provided by the user in the normal way. This error calculation is iterative, so error corrections before a point in time are factored in. The end result is a set of interest events that compensate the existing profile for its "error". The future profile (i.e. current topics) is then computed in the normal way, using these error corrections.

This algorithm allows the user the opportunity to manually correct the profile. The user can change the absolute interest points at a later date, and the error computations will be re-formulated. By adding the interest values at the time of the error, the time-decay properties of the added interest values will not be as dramatic as they would be if the error compensation was added at the current date.

Potentially a more sophisticated profiling algorithm could be used other than a time-decay function. This would bring the shape of the profile into play more, allowing advanced predictions of future interest. This idea will be explored as a follow on from the trial data, by running various algorithms on the user log data accumulated by the trial.

6.4 Experimental evaluation of the Foxtrot system

To obtain sufficient subjects, 3rd year computer science undergraduates will be targeted and Foxtrot offered as a tool to help their 3rd year projects. The undergraduates will be split into two groups, one using profile visualization and one not (using simple feedback like Quickstep does).

About 100 undergraduates are expected to participate, making 50 for each group. As 25 subjects are needed per group to avoid the need for T-tests in the analysis, about half the undergraduates must actively participate to produce a valid statistical analysis. Active participation involves using the Foxtrot system on at least 3 different occasions, and hence generating enough feedback to allow assessment of the effectiveness of the profiling. The profile visualization group will also need to use the profile visualization facility to allow analysis to be performed. If this level of subject participation is met, it is hoped that the differences between the groups will be bigger than the error margin (derived from the standard deviation), and hence quantitative results concluded.

If subject participation proves to be less than the desired level, T-test analysis can be performed. This is unlikely to produce a low enough standard deviation (and hence error margin) to prove any differences between the groups seen are not down to chance. With this eventuality, qualitative results will have to be drawn.

The normal Quickstep metrics will be measured, and groups compared. In addition to behaviour logging, profiles will be compared to the known projects the students are undertaking. This provides a control for what sort of topics they should be interested in and hence allow profile accuracy to be measured. A post trial questionnaire will also ask for each student to report on what research paper interests they had over the period of the trial.

6.5 Experimentation with the subjects behavioural log data

The Foxtrot trial will produce about 9 months worth of undergraduate URL browsing log data. This will be used to perform further experiments to determine the best way to predict future interests. Machine learning approaches, polynomial curve fitting and time-decay functions will be compared to access the best way to predict interests.

The log data is also expected to become an IAM group resource, available for other researchers to experiment with in the future.

Chapter 7 Conclusions and future work

Chapter summary
Conclusions are drawn from the initial work
The future direction of work is discussed, in particular two planned trials.
A 3-year plan is detailed.

7.1 Conclusions

Capturing knowledge of the users preference is not an easy task. The reasons behind user activity are many and varied, and to attempt to capture all this information is unrealistic. What can be done though, is to observe the users at work and build models of their behaviour. This is the approach taken by recommender systems, and has seen some success both in the academic field and commercial market place.

This thesis examines some of the issues involved with profiling user preferences based on unobtrusive observation. A novel multi-class approach is adopted, allowing domain knowledge to be brought to play in the formulation of a profile. The trade-off between the inherent inaccuracy of multi-class classification (compared to binary-class classification) and usefulness of domain knowledge is examined.

Quickstep assesses the degree to which the structure of domain concepts (an "is-a" hierarchy of research paper topics) can assist in predicting user preferences. The two experiments with the Quickstep system suggest a clear trend to better profiles when using this domain knowledge. In comparison with other reported systems (mostly binary-class classification systems), the overall performance appears similar. This is a promising finding since the experiments involved real subjects over a significant period of time.

Foxtrot will assess the degree to which a dialogue with the user about their preferences can improve the feedback elicited, and hence improve the overall profiling process. It seems intuitive that if a system can successfully ask the users for what they actually want, then that system will have a superior profile. The Foxtrot experiment aims to quantify this intuition, and see if the effort invested by the users in providing direct feedback on profiles is worth the gains seen in the profiling process.

What both these system show is that capturing user preferences is a realistic aspiration for today's technology. In the future, increasingly sophisticated user models will no doubt allow systems to become increasing personalized and offer services people require at the time they need them.

7.2 Future direction of work

The next step for the Quickstep experiments is to run more trials and perform rigorous statistical analysis on the results. As the subjects increase in number, there will be increasing confidence in the power of the effects seen. In October, the IAM research group will have some more 1st year postgraduate students, and these will be targeted for a further Quickstep trial (Quickstep3).

As detailed in chapter 6, the Quickstep system will evolve to become the Foxtrot system. It is hoped that 3rd year undergraduate computer science students will participate in a future trial, using the system to aid their 3rd year projects. This will be a long-term (6-9 month duration) trial, with about 100 students. The results obtained will have statistical validity, so should be sufficient to publish as a journal article.

7.3 Three year plan

A three-year plan, which has been occasionally revised, is shown in figure 7.1. This plan was first created in October 1999, and details the milestones along the way to the completion of a PhD thesis. Figure 7.1 shows the plan in its current configuration. Two trials are planned, one for the Quickstep system to gain further statistical evidence and one for the Foxtrot system to evaluate it properly. Time has been allocated to write two journal articles (one for the Quickstep results assuming they are statistically valid and one for the Foxtrot results) and the final thesis.

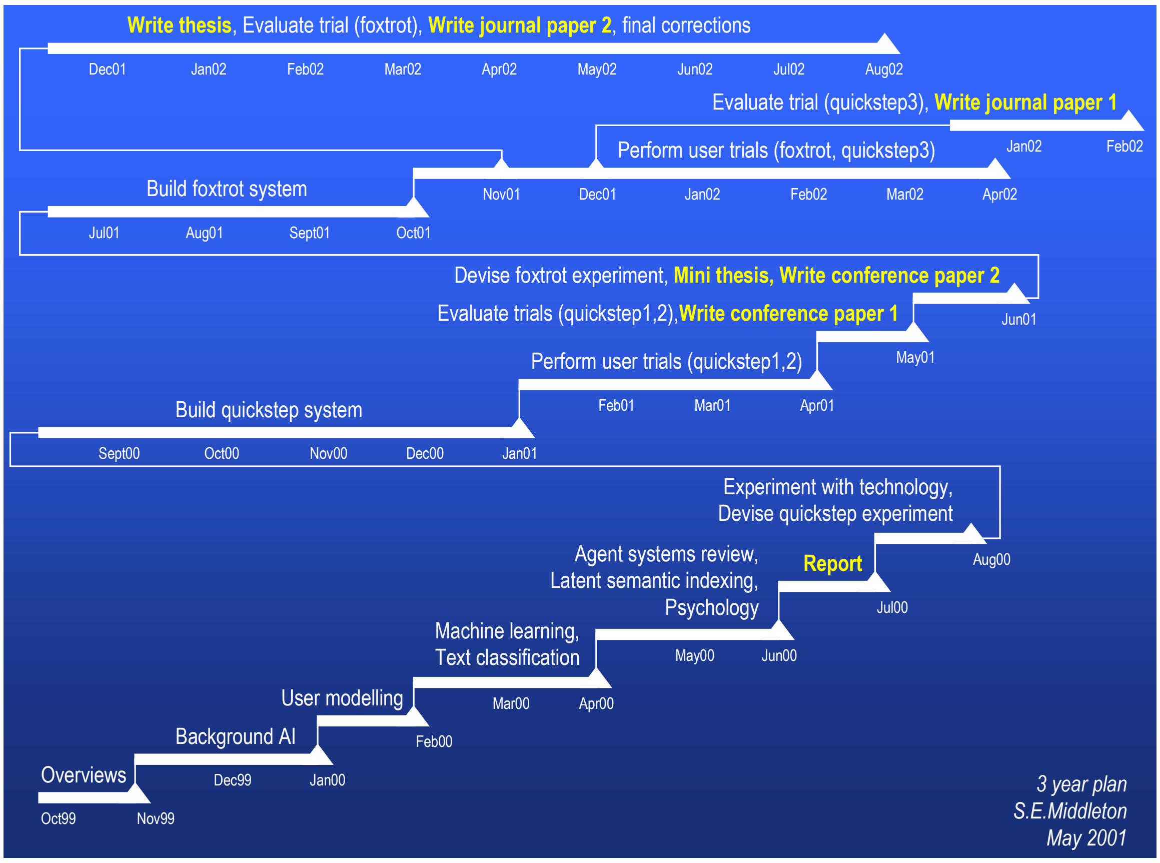

Write thesis, Evaluate trial (foxtrot), Write journal paper 2, final corrections
Dec01 Jan02 Feb02 Mar02 Apr02 May02 Jun02 Jul02 Aug02
Evaluate trial (quickstep3), Write journal paper 1
Jan02 Feb02
Perform user trials (foxtrot, quickstep3)
Nov01 Dec01 Jan02 Feb02 Mar02 Apr02
Build foxtrot system
Jul01 Aug01 Sept01 Oct01
Devise foxtrot experiment, Mini thesis, Write conference paper 2
Jun01
Evaluate trials (quickstep1,2), Write conference paper 1
May01
Perform user trials (quickstep1,2)
Feb01 Mar01 Apr01
Build quickstep system
Sept00 Oct00 Nov00 Dec00 Jan01
Experiment with technology, Devise quickstep experiment
Aug00
Agent systems review, Latent semantic indexing, Psychology
Report
Jul00
Machine learning, Text classification
May00 Jun00
User modelling
Mar00 Apr00
Background AI
Feb00
Overviews
Dec99 Jan00
Oct99 Nov99
3 year plan
S.E.Middleton
May 2001

Figure 7.1 : Three-year plan

References

[1] Aha, D. Kibler, D. Albert, M. Instance-based learning algorithms, Machine Learning, 6:37-66, 1991

[2] Asnicar, F.A. Tasso, C. "ifWeb: a Prototype of User Model-Based Intelligent Agent for Document Filtering and Navigation in the World Wide Web", In Proceedings of the Sixth International Conference on User Modeling, Chia Laguna, Sardinia, June 1997

[3] Balabanović, M. Shoham, Y. "Fab: Content-Based, Collaborative Recommendation", Communications of the ACM 40(3), March 1997, 67-72

[4] Balabanović, M. Shoham, Y. Yun, Y. "An Adaptive Agent for Automated Web Browsing", In Journal of Visual Communication and Image Representation, 6(4), December, 1995

[5] Barrett, R. Maglio, P.P. Kellem, D.C. "WBI: A Confederation of Agents that Personalize the Web", In Autonomous Agents 97, Marina Del Rey, California USA

[6] Bates, J. "The role of emotion in believable agents", Communications of the ACM 37(7), July 1994, 122-125

[7] Billsus, D. Pazzani, M. "Learning Probabilistic User Models", In Workshop Notes of "Machine Learning for User Modeling", Sixth International Conference on User Modeling, Chia Laguna, Sardinia, 1997

[8] Billsus, D. Pazzani, M.J. "A Personal News Agent that Talks, Learns and Explains", In Autonomous Agents 98, Minneapolis MN USA

[9] Bloedorn, E. Wnek, J. "Constructive Induction-based Learning Agents: An Architecture and Preliminary Experiments", In Proceedings of the First International Workshop on Intelligent Adaptive Systems (IAS-95), Melbourne Beach, Florida, 1995, 38-51

[10] Bollacker, K.D. Lawrence, S. Giles, C.L. "CiteSeer: An Autonomous Web Agent for Automatic Retrieval and Identification of Interesting Publications", In Autonomous Agents 98, Minneapolis MN USA

[11] Boone, G. "Concept Features in Re :Agent, an Intelligent Email Agent", In Autonomous Agents 98, Minneapolis MN USA

[12] Brajnik, G. Tasso, C. "A Shell for Developing Non-Monotonic User Modeling Systems", Int J. Human-Computer Studies, 40, pages 31-62, 1994

[13] Brooks, R. A. “Intelligence Without Reason”, In Proceedings of the 1991 International Joint Conference on Artificial Intelligence, 569-695

[14] Brown, S.M. Santos, E., Banks, S.B. “A Dynamic Bayesian Intelligent Interface Agent”, Dept of Electrical and Computer Engineering, Air Force Institute of Technology, Wright-Patterson AFB, OH 45433-7765

[15] Chen, L. Sycara, K. “WebMate: A Personal Agents for Browsing and Searching”, In Autonomous Agents 98, Minneapolis MN USA

[16] Cypher, A. “Eager: Programming repetitive tasks by example”, In Human factors in computing systems conference proceedings on Reaching through technology, April 27 - May 2, 1991, New Orleans, LA USA, 33-39

[17] Davies, J. Weeks, R. Revett, M. McGrath, A. “Using Clustering in a WWW Information Agent”, In 18th BCS IR Colloquium, Manchester, UK, April 1996

[18] De Roure, D. Hall, W. Reich, S. Hill, G. Pikrakis, A. Stairmand, M. MEMOIR – an open framework for enhanced navigation of distributed information, Information Processing and Management, 37, 53-74, 2001

[19] dmoz open directory project, Project home page http://dmoz.org/

[20] Etzioni, O. “A softbot-based interface to the internet”, Communications of the ACM 37(7), July 1994, 72-76

[21] Foner, L.N. “Yenta: A Multi-Agent, Referral-Based Matchmaking System”, In Autonomous Agents 97, Marina Del Rey, California USA

[22] Freund, Y. Schapire, R.E. Experiments with a New Boosting Algorithm, Proceedings of the Thirteenth International Conference on Machine Learning, 1996

[23] Fu, X. Budzik, J. Hammond, K.J. “Mining Navigation History for Recommendation”, In Proceedings of the 2000 Int. Conf. on Intelligent User Interfaces (IUI'00). New Orleans, Louisiana

[24] Glance, N.S. “Community Search Assistant”, In Proceedings of IUI’01, Santa Fe, New Mexico, USA, January 2001

[25] Goldberg, D. Nichols, D. Oki, B.M. Terry, D. “Using Collaborative Filtering to Weave an Information Tapestry”, Communications of the ACM, Vol. 35, No. 12, December 1992

[26] Grasso, A. Koch, M. Rancati, A. “Augmenting Recommender Systems by

Embedding Interfaces into Practices", In Proceedings of GROUP'99, Phoenix, Arizona, November 1999

[27] Green, C.L. Edwards, P. "Using Machine Learning to Enhance Software Tools for Internet Information Management", In AAAI-96 Workshop on Internet-Based Information Systems, WS-96-06, AAAI Press, 1996, 48-55

[28] Han, E. Boley, D. Gini, M. Gross, R. Hastings, K. Karypis, G. Kumar, V. Mobasher, B. Moore, J. "WebACE : A Web Agent for Document Categorization and Exploration", In Autonomous Agents 98, Minneapolis MN USA

[29] Harman, D. An Experimental Study of Factors Important in Document Ranking. Proceedings of 1986 ACM conference on Research and development in information retrieval, September 1986, Pisa Italy

[30] Haynes, T. Sen, S. Arora, N. Nadella, R. "An automated meeting scheduling system that utilizes user preferences", In Autonomous Agents 97, Marina Del Rey, California USA

[31] Hoyle, M.A. Lueg, C. "Open Sesame !: A Look at Personal Assistants", Proceedings of the International Conference on the Practical Application of Intelligent Agents (PAAM97), London, 1997, 51-60

[32] Joachims, T. Freitag, D. Mitchell, T. "WebWatcher: A Tour Guide for the World Wide Web", In Proceedings of IJCAI97, August 1997

[33] Kamba, T. Bharat, K. Albers, M.C. "The Krakatoa Chronicle: An Interactive, Personalized Newspaper on the Web", Proceedings of WWW4, Boston, USA, December 1995

[34] Kautz, H. Selman, B. Shah, M. "Referral Web: Combining Social Networks and Collaborative Filtering", Communications of the ACM 40(3), March 1997, 63-65

[35] Kay, A. "User interface: A personal view", In: Laurel. B. (ed.). The art of Human-Computer Interface Design, Addison-Wesley, 1990, 191-207

[36] Kobsa A. "User Modeling: Recent work, prospects and Hazards", In Adaptive User Interfaces: Principles and Practice Schneider-Hufschmidt, M. Kühme, T. Malinowski, U. (ed) North-Holland 1993

[37] Konstan, J.A. Miller, B.N. Maltz, D. Herlocker, J.L. Gordon, L.R. Riedl, J. "GroupLens: Applying Collaborative Filtering to Usenet News",

Communications of the ACM 40(3), March 1997, 77-87

[38] Lang, K. “NewsWeeder: Learning to Filter NetNews”, In ICML95 Conference Proceedings, 1995, 331-339

[39] Larkey, L.S. “Automatic essay grading using text categorization techniques”, In Proceedings of SIGIR-98, 21st ACM International Conference on Research and Development in Information Retrieval, Melbourne, AU, 1998

[40] Lieberman, H. Letizia: An Agent That Assists Web Browsing, Proceedings of the 1995 International Joint Conference on Artificial Intelligence, Montreal, Canada, August 1995

[41] Lieberman, H. Maes, P. Van Dyke, N.W. “Butterfly: A Conversation-Finding Agent for Internet Relay Chat”, Proceedings of the 1999 International Conference on Intelligent User Interfaces, January 1999

[42] Lieberman, H. Nardi, B.A. Wright, D. “Training Agents to Recognize Text by Example”, In Autonomous Agents 99, Seattle WA USA

[43] Lieberman, H. van Dyke, N. Vivacqua, A. “Let’s Browse : a collaborative browsing agent”, Knowledge-Based Systems, Vol. 12, Dec. 1999, 427-431

[44] Maes, P. “Agents that reduce work and information overload”, Communications of the ACM 37(7) July 1994, 108-114

[45] Maes, P. “Artificial Life meets Entertainment: Lifelike Autonomous Agents”, Communications of the ACM 38(11), November 1995, 108-114

[46] Maes, P. Chavez, A. “Kasbah: An Agent Marketplace for Buying and Selling Goods”, Proceedings of the First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology, London, UK, April 1996

[47] Maes, P. Darrell, T. Blumberg, B. Pentland, A. “The ALIVE System: Wireless, Full-Body Interaction with Autonomous Agents”, ACM Multimedia Systems, Special Issue on Multimedia and Multisensory Virtual Worlds, ACM Press, Spring 1996

[49] Maes, P. Kozierok, R. “A learning interface agent for scheduling meetings”, In Proceedings of ACM SIGCHI International Workshop on Intelligent User Interfaces, ACM Press, NY, 1993, 81-88

[50] Maes, P. Velásquez, J.D. “Cathexis: A Computational Model of Emotions”, In Autonomous Agents 97, 1997

[51] McCarthy, J. “Recursive Functions of Symbolic Expressions”, CACM 3, 1960, 184-195

[52] McCarthy, J. “Some expert systems need common sense”, In Computer Culture: The Scientific, Intellectual and Social Impact of the Computer, vol. 426, Heinz P. (ed.) 1994 Annals of the New York Academy of Sciences, 1983, 129-137

[53] McCarthy, J. Hayes, P. J. “Some Philosophical Problems from the Standpoint of Artificial Intelligence”, Machine Intelligence 4 1969, 463-502

[54] McDonald, D.W. Ackerman, M.S. “Expertise Recommender: A Flexible Recommendation System and Architecture”, In Proceedings of the ACM 2000 Conference on CSCW, Philadelphia, PA USA, December 2000

[55] Menczer, F. Belew, R.K. “Adaptive Information Agents in Distributed Textual Environments”, In Autonomous Agents 98, Minneapolis MN USA

[56] Middleton, S.E. “Interface agents: A review of the field”, Technical Report Number: ECSTR–IAM01-001, ISBN: 0854327320, University of Southampton, 2001

[57] Middleton, S.E. De Roure, D. C. Shadbolt, N.R. “Capturing Knowledge of User Preferences: ontologies on recommender systems”, To appear in K-CAP2001 (pending acceptance), Oct 2001

[58] Minsky, M. “The Society of Mind”, Simon and Schuster, New York, NY 1986

[59] Mitchell T. M. “Machine Learning”, McGraw-Hill, 1997

[60] Mitchell, T.M. Caruana, R. Freitag, D. McDermott, J. Zabowski, D. “Experience with a learning personal assistant”, Communications of the ACM 37(7), July 1994, 81-91

[61] Mladenić, D. “Personal WebWatcher: design and implementation”, Technical Report IJS-DP-7472, Department for Intelligent Systems, J. Stefan Institute

[62] Mladenić, D. Stefan, J. “Text-Learning and Related Intelligent Agents: A Survey”, IEEE Intelligent Systems, 1999, 44-54

[63] Morris, J. Maes, P. “Negotiating Beyond the Bid Price”, In Workshop Proceedings of the Conference on Human Factors in Computing Systems (CHI 2000), The Hague, The Netherlands, April 1-6, 2000

[64] Moukas, A. “User modelling in a Multi-Agent Evolving System”, In

International Conference on User Modelling '97, Machine Learning in User Modelling Workshop Notes, Chia Laguna, Sardinia, 1997

[65] Nawana, H. “Software agents: an overview”, In The Knowledge Engineering Review, Vol 11:3, 1996, 205-244

[66] Negroponte, N. “The Architecture Machine; Towards a more Human Environment”, MIT Press, Cambridge, Mass. 1970

[67] Newell, A. Shaw, J. C. Simon, H. “A General Problem-Solving Program for a Computer”, Computers and Automation 8(7), 1959, 10-16

[68] Nilsson, N. J. “Problem-Solving Methods in Artificial Intelligence”, McGraw-Hill, New York, NY, 1971

[69] Nilsson, N. J. “Shakey the Robot”, SRI A.I. Center Technical Note 323, April 1984

[70] Norman D.A. “How might people interact with agents” Communications of the ACM 37(7) July 1994, 68-71

[71] Odubiyi, J.B. Kocur, D.J. Weinstein, S.M. Wakim, N. Srivastava, S. Gokey, C. Graham, J. “SAIRE – A scalable agent-based information retrieval engine”, In Autonomous Agents 97, Marina Del Rey, California USA

[72] Pannu, A.S. Sycara, K. “A Learning Personal Agent for Text Filtering and Notification”, In Proceedings of the International Conference of Knowledge Based Systems, 1996

[73] Pazzani, M., Muramatsu, J. and Billus, D. “Syskill & Webert: Identifying interesting web sites”, Paper presented at AAAI Spring Symposium on Machine Learning in Information Access, Stanford, California, USA, March 25-27, 1996

[74] Pazzani, M.J. Billsus, D. "Adaptive Web Site Agents", In Proceedings of the Third International Conference on Autonomous Agents (Agents '99), Seattle, Washington, 1999

[75] Porter, M. An algorithm for suffix stripping, Program 14 (3), July 1980, pp. 130-137

[76] Resnick, P. Varian, H. R. “Recommender systems”, Communications of the ACM 40(3) March 1997, 56-58

[77] Rhodes, B.J. “Just-In-Time Information Retrieval”, PhD thesis, June 2000

[78] Rhodes, B.J. “Margin Notes: building a contextually aware associative

memory", In IUI 2000: 2000 International Conference on Intelligent User Interfaces, New Orleans, Louisiana, January 9-12, 2000, ACM.

[79] Rhodes, B.J. Starner, T. "Remembrance Agent: A continuously running automated information retrieval system", In The proceedings of the First International Conference on The Practical Application of Intelligent Agents (PAAM96), 487-495

[80] Rich, C. Sidner, C.L. "COLLAGEN: When Agents Collaborate with People", In Autonomous Agents 97, Marina Del Rey, California USA

[81] Rich, E. "User modelling via Stereotypes", Cognitive Science 3 1979, 329-354

[82] Rucker, J. Polanco, M.J. "Siteseer: Personalized Navigation for the Web", Communications of the ACM 40(3), March 1997, 73-75

[83] Rumelhart, D. E. Hilton, G. E. Williams, R. J. "Learning Internal Representations by Error Propagation", In "Parallel Distributed Processing", Rumelhart D. E. and McClelland J. L., MIT Press, Camberidge, MA, 1986

[84] Sagula, J.E. Puricelli, M.F. Bobeff, G.J. Martin, G.M. Carlos, E.P. "GALOIS: An Expert-Assistant Model", In Autonomous Agents 97, Marina Del Rey, California USA

[85] Sakagami, H. Kamba, T. Koseki, Y. "Learning personal preferences on online newspaper articles from user behaviors", In 6th International World Wide Web Conference, 1997, pp. 291–300

[86] Schafer, J.B. Konstan, J. Riedl, J. "Recommender Systems in E-Commerce", In Proceedings of the ACM E-Commerce 1999 Conference, Denver, Colorado, 1999

[87] Schlimmer, J.C. Hermens, L.A. "Software agents: Completing Patterns and Constructing User Interfaces", Journal of Artificial Intelligence Research 1 (1993), 61-89

[88] Schwab, I. Pohl, W. Koychev, I. Learning to Recommend from Positive Evidence, Proceedings of Intelligent User Interfaces 2000, ACM Press, pp 241-247

[89] Sebastiani, F. "Machine Learning in Automated Text Categorisation", Consiglio Nazionale delle Ricerche, Via S. Maria, 46-56126 Italy

[90] Segal, R.B. Kephart, J.O. "MailCat: An Intelligent Assistant for Organizing

E-Mail", In Autonomous Agents 99, Seattle WA USA

[91] Selker, T. "Coach: A Teaching Agent that Learns", Communications of the ACM 37(7), July 1994, 92-99

[92] Shadbolt, N. O'Hara, K. Crow, L. The experimental evaluation of knowledge acquisition techniques and methods: history, problems and new directions, International Journal of Human-Computer Studies (1999) 51, pp 729-755

[93] Shneiderman, B. Maes, P. "Direct manipulation vs interface agents", Interactions: new visions of human-computer interaction Nov-Dec 1997

[94] SMART Staff, User's Manual for the SMART Information Retrieval System, Technical Report 71-95, Revised April 1974, Cornell University (1974)

[95] Starr, B. Ackerman, M.S. Pazzani, M. "Do-I-Care : A Collaborative Web Agent", ACM CHI'96, April 1996

[96] Svensson, M. Höök, K. Laaksolahti, J. Waern, A. "Social Navigation of Food Recipes", In Proceedings of SIGCHI'01, Seattle, WA, USA, April 2001

[97] Tatemura, J. "Virtual Reviewers for Collaborative Exploration of Movie Reviews", In Proceedings of IUI'2000, New Orleans, LA, USA, 2000

[98] Terveen, L. Hill, W. Amento, B. McDonald, D. Crester, J. "PHOAKS: A System for Sharing Recommendations", Communications of the ACM 40(3), March 1997, 59-62

[99] Turing A. M. "Computing Machinery and Intelligence", Mind 59, Oct 1950, 433-460

[100] Turing, A. M. "On Computable numbers with an application to the Entscheidungsproblem", Proc. London Math. Soc. 42, 1937, 230-65

[101] van Rijsbergen, C.J. Information Retrieval (Second Edition). Butterworths, 1979

[102] Vivacqua, A. "Agents for Expertise Location", In Autonomous Agents 98, Minneapolis MN USA

[103] Voss, A. Kreifelts, T. SOAP: Social Agents Providing People with Useful Information, Proceedings of the international ACM SIGGROUP conference on Supporting group work (GROUP'97), Phoenix AZ, 1997, pp 291-298

[104] Wasfi, A.M.A. "Collecting User Access Patterns for Building User Profiles and Collaborative Filtering", In Proceedings of the 1999 International Conference on Intelligent User Interfaces, pages 57-64, 1999

[105] Waszkiewicz, P. Cunningham, P. Byrne, C. “Case-based User Profiling in a Personal Travel Assistant”, In Autonomous Agents 98, Minneapolis MN USA

[106] Witten, I.H. Frank, E. Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations, 2000, Morgan Kaufmann publishers.

[107] Wooldridge, M. J. Jennings N. R. “Intelligent Agents: Theory and Practice”, The Knowledge Engineering Review 10(2), 1995

Appendix A Glossary of terms

For a more detailed description of machine learning, [89] provides an excellent overview of machine learning techniques, as does [59].

AQ15c – Rule learning algorithm. Rules are added until an example is fully classified, using a general to specific approach.
A-Priori algorithm – An optimisation algorithm for reducing the number of large itemsets. Used in data mining (for example when finding association rules).
ART-2 – Adaptive resonancy theory-2, a neural network approach.
Association rule discovery – Data mining technique to discover rules that associate items within a database. It is related to the induction of classification rules.
Autoclass – Bayesian classifier for unsupervised classification, based on a classical mixture model.
Backpropagation - Neural network algorithm for updating hidden layer weights. A reliable technique, it is the backbone of many neural networks.
Bag of words – Document representation consisting of a list of words and the number of times the words appear (term frequency).
Bayesian network – A probabilistic network storing the believed conditional dependencies between variables. Joint probability distributions specify the probability of a set of values to a set of variables.
C4.5 – ID3 variant, applying rule post-pruning and other additional techniques.
Case retrieval net – Type of case-based reasoning algorithm, storing similarity between connected elements within the network.
Case-based reasoning – A memory-based reasoning algorithm, using a symbolic representation of cases (as opposed to a vector space approach).
Constructive induction-based learning – Inductive logic programming approach, where background knowledge is used to augment the set of predicates used.
Cosine similarity – dot product measure of the distance between two vectors. This is used to measure similarity between two documents when the vector space represents document features.
Decision tree – Algorithm using a tree, with each node of the tree dividing the hypothesis space using an attribute. As the tree is traversed, from top to bottom, the

hypothesis space is increasingly sub-divided until only one hypothesis is left. Decision trees can be easily converted into classification rules.

Entropy – A measure of the "purity" of a collection of examples. It measures the difference between the number of positive and negative examples (zero is a "pure" or perfectly balanced set).

FAQ – Frequently Asked Questions – A document with often asked questions answered aimed at helping novice users.

Finite state machine – Decision trees situated as states within a finite state machine (used in [87]).

Genetic algorithm – Learning algorithm that supports a population of hypotheses, and evolves them by survival of the fittest, cross-over (combining two successful hypotheses) and mutation.

Hierarchical agglomeration clustering – Starts with one document cluster, and agglomerates the most similar clusters until the desired number of clusters exists. TF-IDF is often used to weight document vectors.

IBPL – A memory-based reasoning algorithm, storing situations and classifying by comparing similarity between situations.

ID3 – Classic decision tree learning algorithm. Uses information gain to select node terms.

ID4 – Variant on ID3.

Information gain – Measure of the expected reduction in entropy of a term.

Jaccard coefficient – No. of pages containing two entities / no. of pages with either entities

Keyword vector – A vector of keywords. Vector has length equal to the number of terms in a document set, and values are the frequency of each term (usually applied to a document to give a document vector).

LikeIt – An algorithm to measure distance between two strings, based on the number of character/symbolic transformations to make first string into the second string.

Memory-based reasoning – Example-based classifier, storing all labelled examples in memory, and determining similarity at run-time. The nearest neighbour algorithm is an example of memory-based reasoning, where labelled examples are held as document vectors.

Minimum description length – Principle that favours hypothesis with the smallest number of terms over longer ones. This ensures simple hypotheses dominate.

Multi-variate Bernoulli formulation – A specific type of naïve Bayesian classifier.

Mutual information - Type of information measure, used to weight terms with respect to positive examples.

Naïve Bayes classifier – Probabilistic classifier based around Bayes theorem. Term probabilities are assigned to classes, and for a new document the probability of belonging to any particular class is computed.

Nearest neighbour – Learning algorithm that measures the distance between document vectors within a vector space representation. The distance indicates similarity of documents (the nearest neighbours) – cosine similarity is often used.

Neural network – Network of units, with inputs usually representing terms and outputs classes. Connections between units have weights, which are trained by loading examples (using a training rule such as backpropagation to update weights).

npdm metric – comparative measurement of how documents rank relative to each other.

Pearson correlation – Type of information measure, used to weight terms with respect to positive and negative examples.

Principle component divisive partitioning – Top down clustering method, splitting the training set until small enough clusters have been formed. A binary tree thus holds the clusters.

Reinforcement learning – Learning algorithm where actions produce rewards or penalties, thus the most rewarding sequence of actions is reinforced (hence learnt).

Rocchio classifier – Learning algorithm, often used with TF-IDF weightings. Class term vectors are computed by summing positive example weights and subtracting negative example weights.

Savant – Type of TF-IDF algorithm.

Scatter/gather browsing – A method of browsing clustered documents. New clusters are formulated when the user selects a subset from the clusters presented, hence allowing iterative browsing of a document space.

Selective induction learning – this is the same as Inductive logic programming (see AQ15c algorithm).

SMART – An indexing engine, which converts documents into document vectors. It uses TF-IDF weighting.

Stemming – Removal of suffixes from words. Used to reduce the number of terms that are synonyms in a textual document.

TF – Term frequency. The number of times a term (often a word or phrase) occurs within a document.

TF-IDF – Term frequency – Inverse Document Frequency. Algorithm for assigning weights to terms in a document set, biased to weight the most discriminating terms highest.

UMT – User Modelling Tool – A user modelling shell system, detailed in [12].

Appendix B Summary of reviewed systems

What follows is a summary of the interface agent systems reviewed in this thesis. The agent systems are listed by application domain, so that similar types of interface agents can be compared together. The machine learning terminology used here is described in the glossary.

When reviewing commercial systems the exact algorithms used are often not published, so can only be surmised. A more detailed review of commercial E-commerce recommender systems can be found in [86].

Quantitative results are detailed where seen so that comparisons (however difficult) can be drawn between systems.

Believable/entertainment domain

ACT [40] is an addition to the ALIVE [47] system. It is a creature within the ALIVE world, observing the user and learning chains of actions. It tries to help the user by completing new action chains in the pattern of previous ones.

ALIVE [47] is a "magic mirror" system to a 3D world. Interactive agents (such as a dog) exist for users to play with. Gesture recognition, and competing goal architecture is employed.

Cathexis [50] is a believable agent with modelled emotions, as is the *Oz* project [6].

Email filtering domain

MAGI [27] filters emails, monitoring user behaviour and receiving relevance feedback. CN2 and IBPL are used to classify emails.

MailCat [90] filters email by providing a choice of folders to the user. Term frequency – inverse document frequency (TF-IDF) vectors are created for existing emails, and cosine similarity used to match new emails. The user has the final say, choosing one of the suggested folders or moving message manually.

Results: 0.3 second classification time, 60-80% accuracy giving user one choice, 80-98% accuracy giving user 3 choices of folder.

Maxims [44] Filters email by learning repetitive actions the user performs. It monitors user actions using memory-based reasoning to discover patterns. Agents can share expertise with other agents, and user programming is allowed.

Re:Agent [11] is an email filterer that accepts user provided keywords for its groupings. TF-IDF vectors are created for each email, along with the TF of the user provided keywords. This representation is then clustered using a nearest neighbour and neural network clustering algorithm (for comparison).
Results: classification accuracy – neural network 94.8 ± 4.2%, nearest neighbour 96.9 ± 2.3%; high accuracy due to simple classification task (into "work" or "other" categories)

Tapestry [25] is an email recommendation system. Users annotate documents as they read them, and collaboratively share this information with others. Users email habits are monitored and implicit feedback obtained (such as user x replied to email y). Users can program filter rules into the system, which are regularly executed. Rules are things like "I want all documents read by user x". As the implicit feedback is shared, the had-crafted rules are simple collaborative filters. A SQL like language allows users to enter rules.

Expert assistance domain

Coach [91] is a LISP help system, that monitors user mistakes and offers unsolicited advise. A knowledge-based user model is supported, with the concept of user experience stereotypically represented. Heuristics adjust model based on user mistakes.
Results: Student performance improved, knowledge of functions improved by a factor of 5

Eager [16] automates observed repetitive HyperCard actions. It monitors the user looking for behaviour patterns, and creates helpful macros from them.
Results: Users felt a loss of control; macros for some irrelevant small patterns were created

GALOIS [84] monitors users using an application, and offers expert advise when they are lost or being inefficient. An initial knowledge-based user profile is constructed (from personal information), then a behavioural model is build by observing user actions. Stereotypes are used to classify users, thus allowing customized help.

GESIA [14] helps expert system developers by suggesting predicted actions. A Bayesian network models user behaviour, allowing predictions (with the help of hard-coded domain knowledge).

Open Sesame! [31] observes user actions and offers to automate repetitive tasks. The ART-2 learning algorithm is used.
Results: only 2/129 suggestions were followed – system deemed to have failed; action patterns do not generalize across situations well

Matchmaking domain

ExpertFinder [102] monitors users' Java code and finds people who use the same classes. TF-IDF vectors represent code file, and cosine similarity is used to find similar people.

Expertise Recommender [54] assists technical support help desk staff in finding the right expert for a task. Recommendations are based on prior requests, so people who helped successfully with a problem before can be selected by the user to do so again. The user can choose which pre-programmed heuristics are used to recommend suitable people (e.g. select by minimum current workload).
Hand crafted heuristics mine an eight-year change-control database to extract initial profile records of who can handle what sort of problem. Profile records are stored with a vector space model, with features identified using a set of thesauri (a stop list is used but no stemming applied).
The recommender only filters the set of possible people to handle a task. An interactive interface allows the help desk staff to pick the particular person they need.

Referral Web [34] models a social network by monitoring social communication sources (email, net news, home pages etc.) and extracting a network model. Heuristics extract names of other people from an individual's communications, which are then refined by computing the Jaccard coefficient between the individual and other names. Once built, the social network can be browsed, and questions asked of it. Recommendations of related people to talk to about an area can thus be extracted (e.g. list documents close to Tom Mitchell).

Yenta [21] allows user agents to "find" each other, and determine commonality of interests. The SMART algorithm initially classifies user emails, newsgroups and created files in order to build an interest profile. Agents then find each other in the Yenta system, compare profiles for similarity, and suggest other agents to try.
Results: Halves the worst-case search space, robust to removal of agents

Meeting schedulers

CAP [60] is a calendar manager, monitoring email and scheduling software to detect meeting patterns. Decision trees (ID3) using information gain to select deciding features were used, being converted to production rules.
Results: 31-60% accuracy (average of 47%) not sufficient for automation, rules were human readable which improved user understanding

Meeting scheduling agent [49] schedules meetings by learning repetitive actions the user performs. Memory-based reasoning and reinforcements learning are used. Users can give explicit feedback.
Results: Confidence for correct predictions settles to 0.8 to 1.0. Confidence for incorrect predictions settles to 0 to 0.2. Some rouge confidence values remain after settling time.

Haynes' [30] meeting scheduler assigns an agent to each user, who then programs it with their preferences. The agents then negotiate meeting times with each other.

News filtering domain

ANATAGONOMY [85] is based on the Krakatoa Chronicle, providing a personalized newspaper. Implicit feedback from user activity has been added.

Results: 1-10% error after 3 days settling time

Butterfly [41] finds interesting conversations within Usenet newsgroups. The user initially provides keywords, and term frequency similarity between newsgroups and the user's profile is computed.

GroupLens [37] collaboratively recommends Usenet newsgroup articles. A ratings database containing user ratings for each message, and a correlations database containing pairs of similar users, is maintained. A Pearson correlation algorithm was used to find similar users (applied within single newsgroups since ratings were too sparse to work for all newsgroups).
Results: Various Usenet use figures are presented.

IAN [27] filters Usenet news, taking relevance feedback from the user. C4.5 rule induction with TF keyword selection (low entropy words being removed) is compared to IBPL (as used in MAGI).
Results: accuracy – C4.5 broad topics 70%, narrow topics 25-30% IBPL broad topics 59-65%, narrow topics 40-45%

The *Krakatoa Chronicle* [33] is a personalized newspaper which adapts to its users' preferences. User reading is monitored and relevance feedback accepted. The SMART algorithm is used, with TF-IDF, to represent articles and compute similarity.

NewsDude [8] reads interesting news articles via a speech interface. The news source is Yahoo! News, with an initial training set of interesting news articles provided by the user. Length of listening time provides implicit user feedback on articles read out. A short-term user model is based on TF-IDF (cosine similarity), and long-term model based on a naïve bayes classifier (multi-variate Bernoulli formulation).
Results: Accuracy 60-76% (using hybrid of long and short term models), F1 measure 25-60%

NewsWeeder [38] filters Usenet newsgroups, taking user relevance feedback. It uses a "bag of words" approach, with stemming. TD-IDF and minimum description length are compared, using cosine similarity.

Results: TF-IDF precision was 37-45%, MDL precision was 44-59% (best)

NewT [44] filters articles from Usenet Netnews. NewT a vector space model to represent news articles. An initial training set and user relevance feedback trains the filter and user programming is allowed.
Results: Users liked the system and found it useful; the simple keyword representation was a limitation

PHOAKS [98] recommends web references found in Usenet news articles. A hand-crafted set of filter rules is used to classify web resources into categories. Web references are then given a rating based on the number of authors that recommend the reference (the idea being frequently referenced web pages are good ones). Each newsgroup thus has a set of ranked recommendations to web pages.
Results: The filter rules have a precision of 88% with a recall of 87%. When compared to the newsgroups FAQ, the 1st place URL had a 30% probability of appearing in the FAQ.

Pannu's [72] learning personal agent finds relevant information from Usenet news (e.g. conference proceedings). The user, along with feedback on examples suggested, provides an initial training set. TF-IDF and a neural network (3 layer backpropagation) were compared.
Results: neural network precision 94% recall 60%, TD-IDF precision 93% recall 100% (best)

E-commerce domain

Amazon.com is a commercial book shop/recommendation service. Customers can rate books they have read using a five star rating, and attach a textual review of the book. This feedback information is shared and used to collaboratively recommend books to other users. Recommendations are based on either the most frequently purchased books or books purchased by similar people to the current user (based on a match between the current users previously purchased books and other users previously purchased books).

In addition to the recommendation service, a conventional search engine can be used to find specific books.

CDNOW is a commercial music CD shop/recommender system. Customers provide feedback as to which artists they prefer and own. Likes and dislikes can be indicated and a set of 6 albums recommended upon request. Feedback on these recommendations is also elicited.
A standard search facility is provided, and 10 other related albums to any single album recommended. A list of albums the customer owns is maintained, and new purchases are added to this list.

eBay is a commercial system which allows buyers and sellers to contribute to profiles of other customers with whom they have done business. A satisfaction rating is elicited from users, which is shared and used when recommending potential sellers.

EFOL [96] is a shopping program in which recipes are selected and the ingredients ordered on-line. Collections of recipes are created by users (using an editor) and made available to others. Discussion about individual choices of recipe is facilitated using a chat area. Collaborative recommendation is supported where clusters of similar users are formulated (using a system editor) and made available for others to take suggestions from.
Results: 12 people (all researchers) used the system on two separate occasions. Half the users reported other people's recipes influenced them, and the pictures of food made them feel hungry.

ELFI [88] recommends research funding program information to users. Users are monitored as they use the system, and positive examples obtained from observations of the type of thing they are interested in. This training set is applied to both a simple Bayes classifier and k-nearest neighbour (kNN) classifier. Funding information is held as feature vectors, and univariate significance analysis used to reduce vector dimensionality. The classifiers are used to measure the similarity of unseen database entries to the interesting training set. The closest matching pages are recommended to the user.

Results: 220 users, divided into 5 groups. The user activity logs were used as training/test data using a cross validation method. simple Bayes classifier 91-97% accuracy, kNN 94-97% accuracy

Kasbah [46] is a market system where each user has an agent. The user programs the agent with a buying behaviour profile, and the agent negotiates to buy and sell items for the user.
Results: Users wanted more "human like" negotiation from the agents, otherwise well received

Levis is a commercial clothing recommender system. Feedback on three categories (music, looks and fun) is elicited from the user using a 7-point scale. 6 items of clothing are then recommended from the Levis range and feedback elicited on the recommendations. Recommendations are thus based on what similar people preferred.

MovieFinder.com is a commercial system that recommends movies. Previous interests are recorded using a 5-point feedback scale, and new recommendations based on the average customer rating. Individual movies also contain a textual prediction when they are browsed.

Reel.com is a commercial system that recommends movies based on customer reviews. The customers enter their movie requirements (genre, viewing format, price etc.) and a set of recommendations is computed based on the habits of other customers.

Ringo [44] is a music recommender system. It uses collaborative filtering based on users ratings of music albums. Virtual users are created to bootstrap the system, providing some initial stereotypical ratings (e.g. a virtual Madonna fan). Pearson r correlation coefficients are used to determine similarity.
Results: A real/predicted scatter plot is presented

Sardine [63] is an auction agent that tries to purchase an airline ticket for the user, based on some specified preferences. The user's agent negotiates with travel agents to secure the best deal.

Tatemura's [97] system uses virtual reviewers for recommendation of movies. Users can explicitly rate movies they have seen. Users can collate movies together to form a new viewpoint (a virtual reviewer), and ask for recommendations from this viewpoint. A vector space model is used to compute the similarity between a particular viewpoint and known movies, and a scatter/gather method used to navigate this space.

Web domain

AARON [27] is the same as LAW, but the AutoClass learning algorithm is used.

Amalthaea [64] observes user browsing behaviour and assists the user in finding interesting WWW information. Browser history, bookmarks and other agent profiles initialise the system. Relevance feedback is recorded and document representation is by stemmed, keyword vectors. A genetic algorithm approach is used to learn.
Results: After 5000 user feedback instances, error averaged 7% with a large scatter (0 – 30%). Sudden changes in user interest were tracked after about 20 generations.

ARACHNID [55] is a spider that crawls the web (or any library), starting from the users' bookmarks, searching for user provided keywords. The user provides feedback on pages found. A genetic algorithm is used in addition to reinforcement learning.
Results: Average search length (number of irrelevant docs before a relevant one is found) shorter than breath-first search by a factor of 10. More sophisticated techniques not compared.

CiteSeer [10] helps users perform keyword searches for publications by making use of citations in documents. Heuristics extract citations, titles and abstracts, then algorithms (TF-IDF, LikeIt) classify publications based on stemmed words. Citation links lead to new search keywords.

Community Search Assistant [24] is a meta-search engine, which maintains a database of previous search queries from which to recommend. No user feedback is required, as similar search queries are identified using heuristics that find similarity within the search graphs. Each users search queries are shared. Similarity is based on query keyword correlation.

Do-I-Care [95] is an agent that monitors known sites, and reports any interesting changes. Mutual information selects terms, and a Bayesian classifier determines similarity of changes to known examples of relevant changes. The user gives feedback on relevance of notifications (by email) and other people agents can share examples of relevant changes.

Fab [3] recommends web pages based on relevance feedback from users. Users rate recommended web pages on a 7-point scale. A set of collection agents dynamically formulates useful groups of pages, with successful agents duplicated and unsuccessful agents removed (success is a measure of feedback scores). Several agent heuristics are used to create page groups, including using commercial search engine results, random picks and human selected "cool" sites.
Selection agents pick pages from the collection agent topics for recommendation (thus sharing topics between users). A profile is built from the terms of the pages (selection agents have user profiles, collection agents have topic profiles).
Results: ndpm measure (distance of user rankings from profile rankings) 0.2 - 0.4 using Fab system, 0.75 – 0.5 using random selection

ifWeb [2] assists web navigation and recommends pages similar to example pages provided by the user. A vector-based user profile is maintained using features of web pages (host, size etc.) and a semantic network for term page co-occurance relationships. The users provide explicit feedback on page relevance (positive and negative). A temporal decay function weights features within the user model, which is represented using UMT. A tree interface can be displayed showing where the current web page is located in relation to its links, and how interesting the links are from it (based on correlation between the crawled pages and the user profile). A search for similar documents is also available, using the same mechanism.
Results: Results on tests using 4 subjects on a limited set of documents (4-6). 9 sessions were conducted, with learning from feedback occurring between each session. Precision 65%, ndpm 0.2

Jasper [17] finds relevant information within a limited WWW library. The user provides interest groups and keywords and gives relevance feedback. User behaviour (reading and authoring) is also monitored. Documents and keywords are clustered using a hierarchical agglomerative process, with features extracted from keywords and metadata (e.g. URL, title etc.).

LAW [27] is an agent that finds interesting web pages for its user. The agent monitors user browsing and asks for relevance feedback. TF, TF-IDF and term relevance are compared for feature selection algorithms, along with IBPL and C4.5 for learning algorithms.
Results: accuracy TF 60-80%(best), TF-IDF 55-70%, term relevance 60-65% and accuracy IBPL 65-83%(best), C4.5 55-65%

Letizia [40] suggests interesting web links, based on monitored user browsing. TF similarity between documents is used, and heuristics infer user preferences (e.g. short view time implies irrelevant page).

Let's browse [43] helps users of TV systems collaboratively browse the web. Based on Letizia, TF-IDF scans of users' home pages provide an initial profile, and group browsing creates trials of web pages. Browsing behaviour is used to infer page relevance.
Results: 50 (as opposed to 10 in Letizia) keywords needs, reflecting a groups wider interests; system well received by users (no controlled experiments however)

LIRA [4] finds interesting web pages (via a heuristic search), and presents them in a daily newspaper, upon which the users provide relevancy feedback. TF-IDF, on stemmed words, with cosine similarity is used.
Results: LIRA matched human performance; pages were very similar to each other

Margin Notes [78] adds a suggestions list to the side of the web browser. The user provides an initial list of interesting documents, which are converted to vectors by the Savant algorithm (similar to TF-IDF). The current web page provides the context for suggestions, with the top suggestion being displayed (summary and a link).

Results: only 6% of suggestions were followed; users found suggestion summaries useful (even without following links)

MEMOIR [18] records user web navigation trials and provides a framework for recommendation of web pages. Users must manually enter URL trials into the system, which are then shared with all users. MEMOIR monitors user web browsing and tries to correlate the current web position with a trial. Web pages (based on trial end points) are recommended when a correlation is made. In addition to web pages, similar users can be recommended based on keyword profiles derived from their trials.

Personal WebWatcher [61] observes user browsing, and suggests interesting web pages. A “bag of words” representation is used, selecting features based on mutual information. Naive Bayes and nearest neighbour algorithms are compared.
Results: classification accuracy Bayes 83-95%, nearest neighbour 90-95% (best)

ProfBuilder [104] monitors web site use and recommends pages from that site to new visitors. A vector space user profile is constructed from the pages a visitor has seen so far. The content of the pages make up the vectors using TF-IDF weightings. Stemming and stop words are used to reduce vector dimensionality, and a vector cosine measure used to measure vector similarity. For each page in the web site, the probability of previous users moving down a link is computed from historical navigation patterns. Both similar pages and pages historically likely to be navigated from the current page are selected for recommendation.

SAIRE [71] is a large multi-agent system helping users search NASA’s web resources. Knowledge-based, stereotypical user modelling is employed along with semantic networks, relevancy rankings and similarity based (keyword and topic) classification.

Siteseer [82] recommends web pages collaboratively. Bookmarks are used to find similar users (by computing the overlap of a users bookmarks with the other users bookmarks). Recommended URL’s thus derive from the bookmark lists of similar users.
Results: 1000 users, 18% confidence recommending 88% of the time.

SOAP [103] is a multi-agent system that recommends web sites. User, search and recommender agents communicate to achieve recommendation for multiple users. Users can submit queries to an agent, which calls a search engine. The search results are associated with the query (taken as the "topic" of the resulting URL's). Users can explicitly rate pages using a 5-point scale and can provide free-text annotations. User bookmarks are used to infer interest in URL's too. Recommender agents use the topic (query) and rating to filter known URL's and hence provide recommendations. Since annotations and ratings are shared, any user can inspect them. Page content is represented using keyword vectors.

SurfLen [23] monitors user browsing and recommends web pages. User browser history logs are mined for association rules using the A-Priori algorithm. These rules associate URL's with other URL's. When a user opens a known URL, the associated URL's are immediately recommended.
Results: Some quantitative figures for 100 simulated users (based on Yahoo log data)

Syskill and Webert [73] rates web links for relevance based on a user profile. An initial training set and explicit relevance feedback is provided. Simple naïve Bayes, nearest neighbour, decision tree (ID3) and TF-IDF algorithms are compared.
Results: Average precision ratings are TF-IDF 85%, Bayes 80%, nearest neighbour 80%, ID3 73%. Nearest neighbour is thought to be best overall (being more consistent than TF-IDF), especially if many examples are available.

WebACE [28] monitors the user's browsing, and suggests interesting new pages. Browsed pages are classified (via clustering), and search queries generated. New pages found that are similar to the users' browsed pages are suggested to the user. Principle component divisive partitioning (PCDP -document vectors split by TF into a binary tree) and an association rule discovery method are compared to Autoclass and hierarchical agglomeration clustering.
Results: Speed to find new low entropy pages, Autoclass 38 mins, HAC 100 mins, PCDP and association rule < 2 mins (best)

WebMate [15] monitors web browsing and compiles a newspaper of interesting pages. User provides positive feedback while browsing and relevance feedback on presented pages. TF-IDF vectors used with cosine similarity measures.
Results: average accuracy 52% for top 10 recommendations, 30.4% for all recommendations; Accuracy lowered by web advertisements and irrelevant text surrounding articles.

WebWatcher [32] is a tour guide for a web site. Pages are represented using TF-IDF term vectors. User browsing behaviour is monitored, and reinforcement learning used to find the best path to pages related to user supplied keywords.
Results: TF-IDF accuracy 43%, Human accuracy 48%

WBI [5] monitors user browsing, offering simple but helpful services. Keyword analysis classifies web pages, and interest predictions are offered on links to pages. It is commercially available from IBM.

Pazzani's [74] adaptive web site agent suggests similar documents to read for web browsers of a particular site. It uses publication references, download frequency and TF-IDF keyword vectors (with cosine similarity measure) to suggest other documents of interest.
Results: 68% increase in publications downloaded (tech papers domain), 16% increase (goat domain)

Other domains

Campiello [26] is a recommender system for leisure activities in a local community. Campiello elicits feedback on leisure events and places using postcard type forms. Freeform textual comments and scaled ratings are recorded. Recommendations can be requested on particular events and places, and both contend-based and collaborative recommendation is used. The Pearson algorithm is used to find similar users.
An internal database is maintained for a particular city, with a newscard reader installed at the leisure facilities (such as within museums) to accept feedback and provide recommendations.

CILA [9] is an agent tested in an artificial, abstract domain. It tests constructive induction-based learning against AQ15c and selective induction. User monitoring, relevance feedback, initial training sets and social collaboration with other agents are supported (in its abstract world).
Results: constructive induction was most accurate (only artificial results however)

CIMA [40] is a text prediction agent, which suggests completions of sentences in a text editor. Heuristics learn from observed examples, hints and partial specifications.

COLLAGEN [80] is an agent whose interaction style is modelled on human collaboration. Agents and users share a goal and plan, and communicate actions, results etc via a dialogue.

Grammex [42] learns grammar (e.g. email structure) from user examples and performs actions when it spots new occurrences of this grammar. The user programs by example, using a direct manipulation interface to teach the agent.

Mondrian [40] learns graphical operations which are explicitly programmed by users. Novices can then user these operations.

Remembrance agent [79] suggests documents related to the user's current context. Emails and on-line documents are monitored, and the SMART algorithm used to match context and documents.
Results: Email most useful for up-to-date contextual information, RA preferred over a search engine or Margin notes [77].

Softbot [20] plans internet-based actions from incomplete user goal specifications (e.g. "send mail to Mitchell"). A planning library of schemata is used, written by hand in Prolog.

UCI GrantLearner [7] is a system to identify interesting grant funding opportunities. It uses the same user model learning techniques as Syskill & Webert (see previous).

Waszkiewicz [105] describes a personal travel assistant aimed at meeting the FIPA 1997 travel scenario. The user specifies preferences and a travel agent suggests a flight. Case-based reasoning (case retrieval net) is used to build a user profile from past requests. User confirmation is sought before booking.

Schlimmer [87] describes a text-completion agent, using finite state machines with embedded decision trees to predict user's textual input and offer a shortcut to completion.
Results: FSM compares well with ID4 and Bayes, with a hybrid of FSM and ID4 working best. Accuracy of 12-82% was seen, depending on the topic of the notes being taken.

Appendix C The Quickstep system design

This section details some of the design documents generated as part of creating the Quickstep system.
Nine independent processes make up the Quickstep system, all interacting to provide the recommendation service to its users. Figure C.1 shows the process map, detailing how these 9 processes (the ellipses) interact. Basically, each process reads information from a data store, processes it and stores the results to another data store. File locking prevents multiple processes corrupting a data store. Processes are timed to run at periods during the day and night, and sequenced so that the results of one process (e.g. classification process) will be ready for the next process (e.g. profile builder).

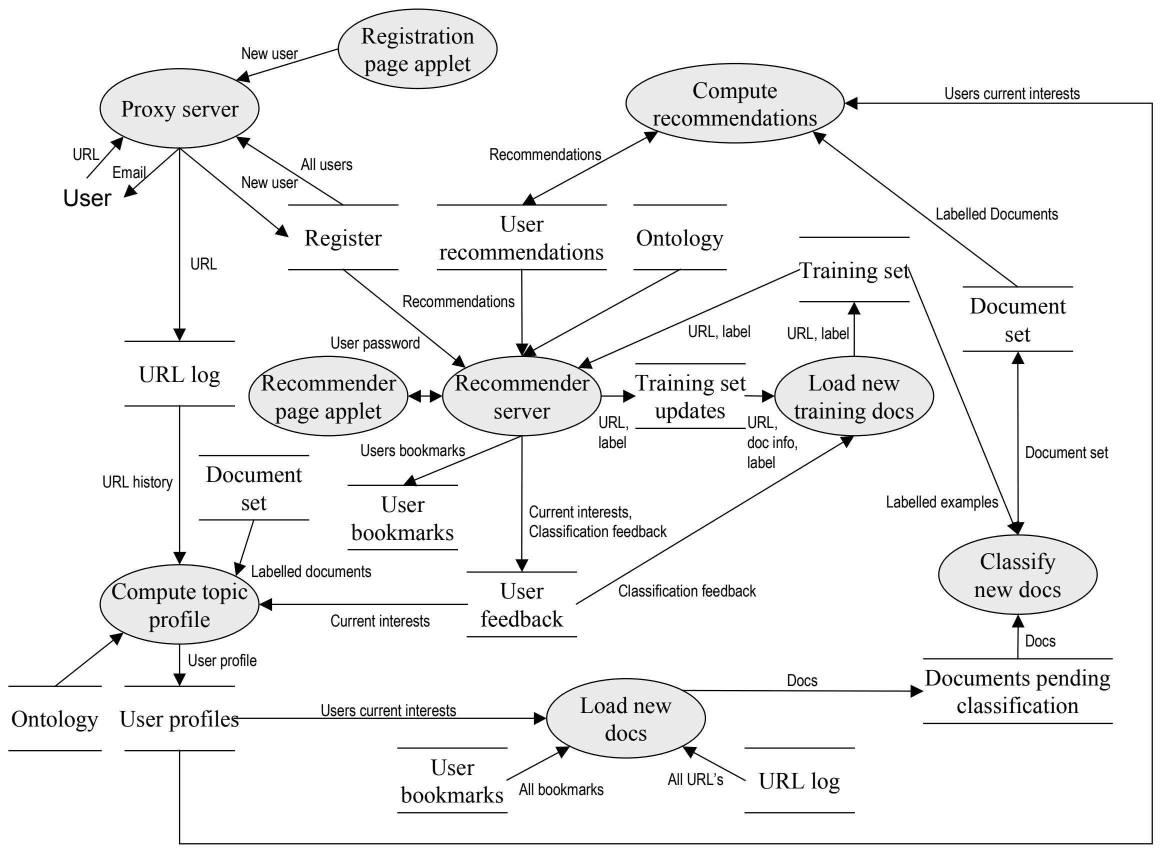


Figure C.1 : Quickstep process map

Each process is functionally decomposed in the detailed design.

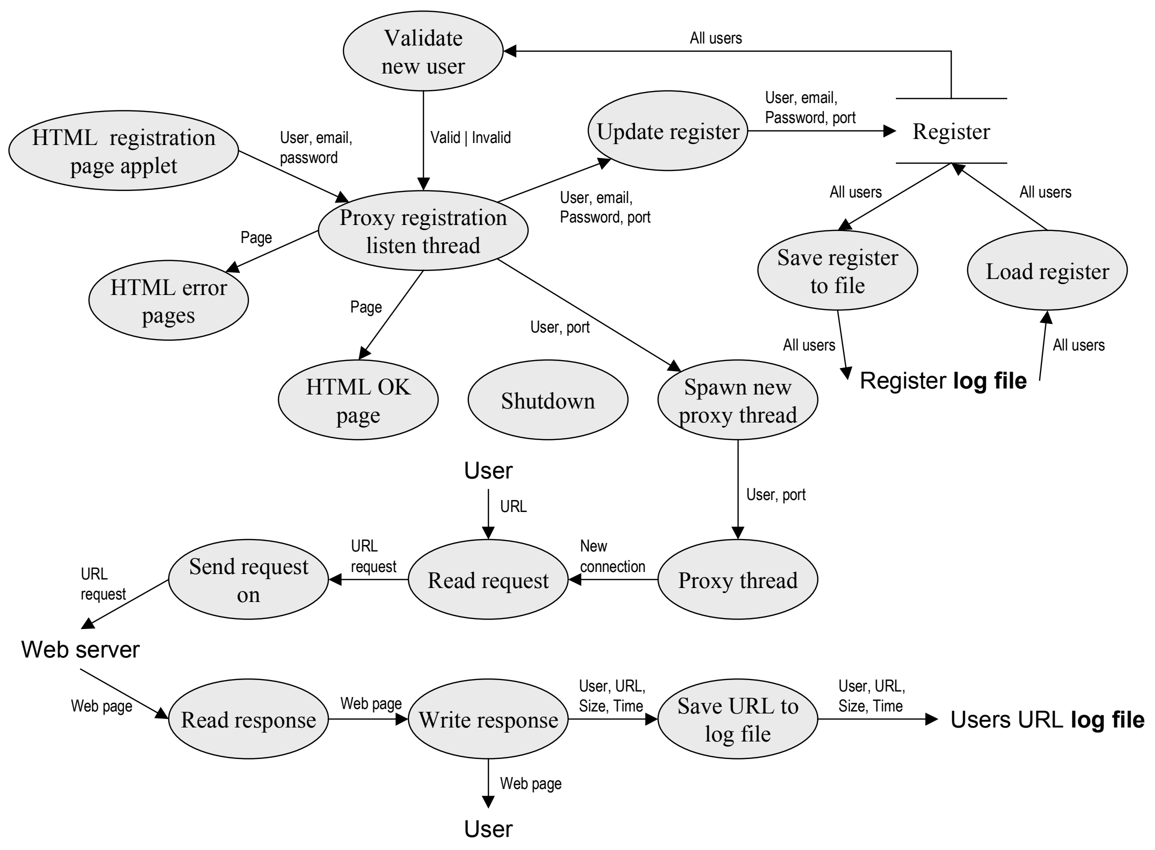


Figure C.2 : Proxy server design

The proxy server process handles requests by users for web pages. The web pages are retrieved from the web without any modification, and the URL request logged (with a time stamp). An on-line registration service is also supported, so users can register with the Quickstep system and allocate a port number. The system emails users their individual proxy port number, allowing them to enter the correct proxy server configuration details into their browsers. Once allocated, a thread is spawned to handle the new port connections. Figure C.2 shows the proxy server design.

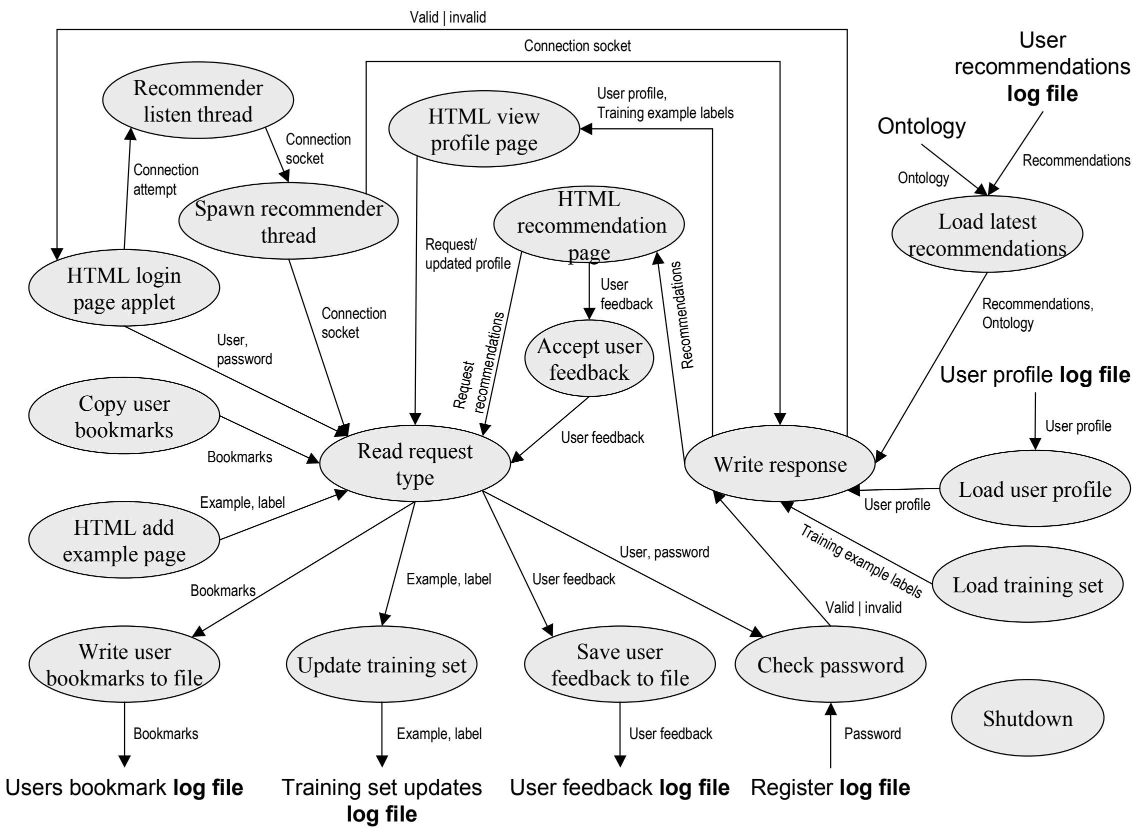


Figure C.3 : Recommendation server/applet design

The recommender server process (and its associated recommender applet process) allows the user to access their pre-computed recommendations. The users will load the applet via a web page, and it will attempt to connect to the recommendation server. A client-server set-up is required to overcome Java security restrictions, since file access is needed on the host machine to read the recommendation log file. Once the user has logged on (and the recommendation server authorised the logon), the recommendations for that user will be sent to the recommendation applet for display. These recommendations can be examined via the interface and feedback provided. When the user logs out (or closes the applet by closing the browser) the feedback is sent to the recommendation server and saved in the feedback log file. Figure C.3 shows the recommendation server/applet design.

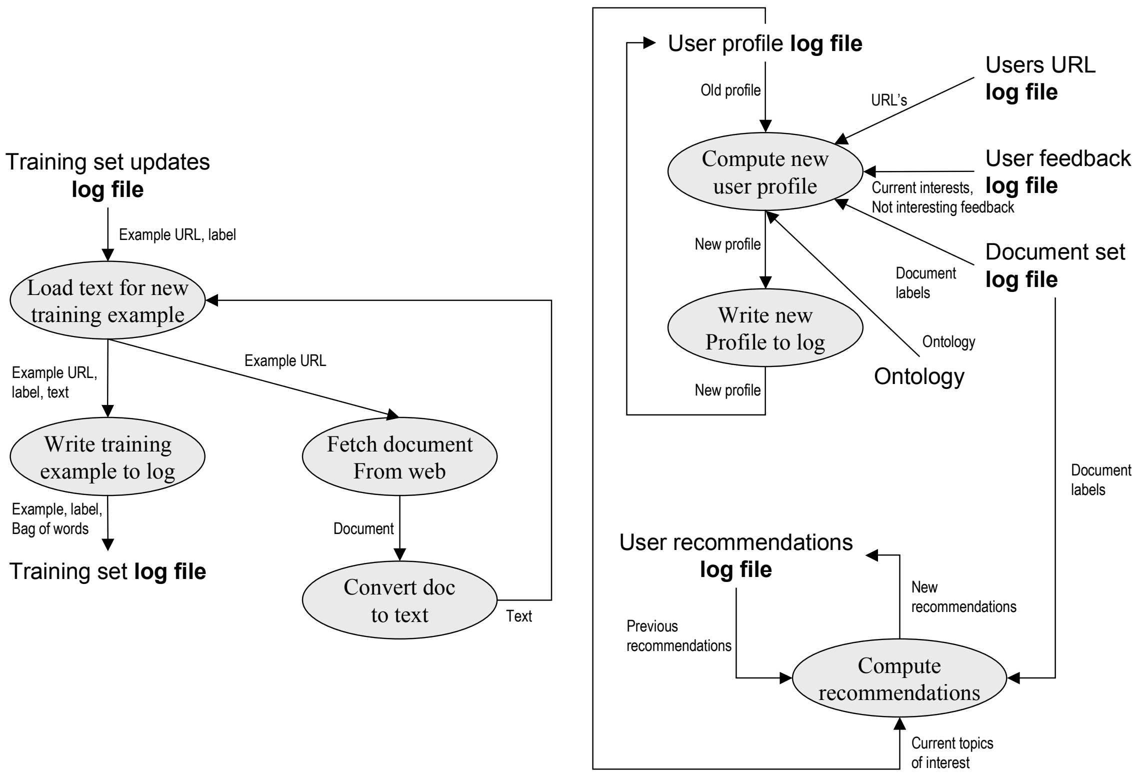


Figure C.4 : Training set compiler/profiler/recommender design

The training set compiler process loads the training set update log (created as part of the feedback saved by the recommendation server) and loads each document. The loaded documents are then saved into the training set log. The profile compiler loads the classified document store and feedback logs, and computes a profile for each user. The URL logs are correlated with the classified documents to build a time log of topics browsed. Interest values can then be calculated from this topic history and the feedback logs. The recommendation compiler loads the user's profiles and correlates them with the classified documents to build a ranked list of potential recommendations. The top 10 recommendations are stored in the user's recommendation log. Figure C.4 details the design of these processes.

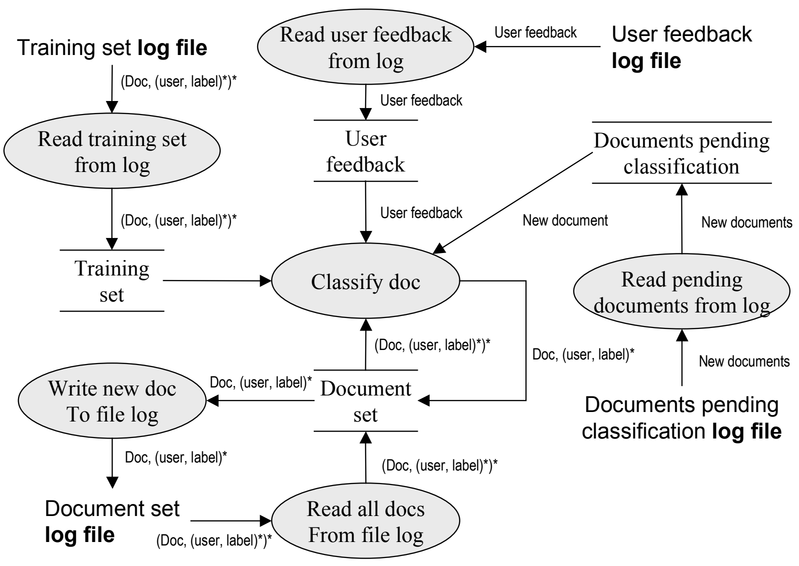


Figure C.5 : Classifier design

The classifier process loads the current training set and builds a new classifier each time it is run. It then iterates over the pending document store and classifies each one. These classified documents are then moved to the classified document store. This is a slow process, so is run overnight. Figure C.5 details the classifier process.

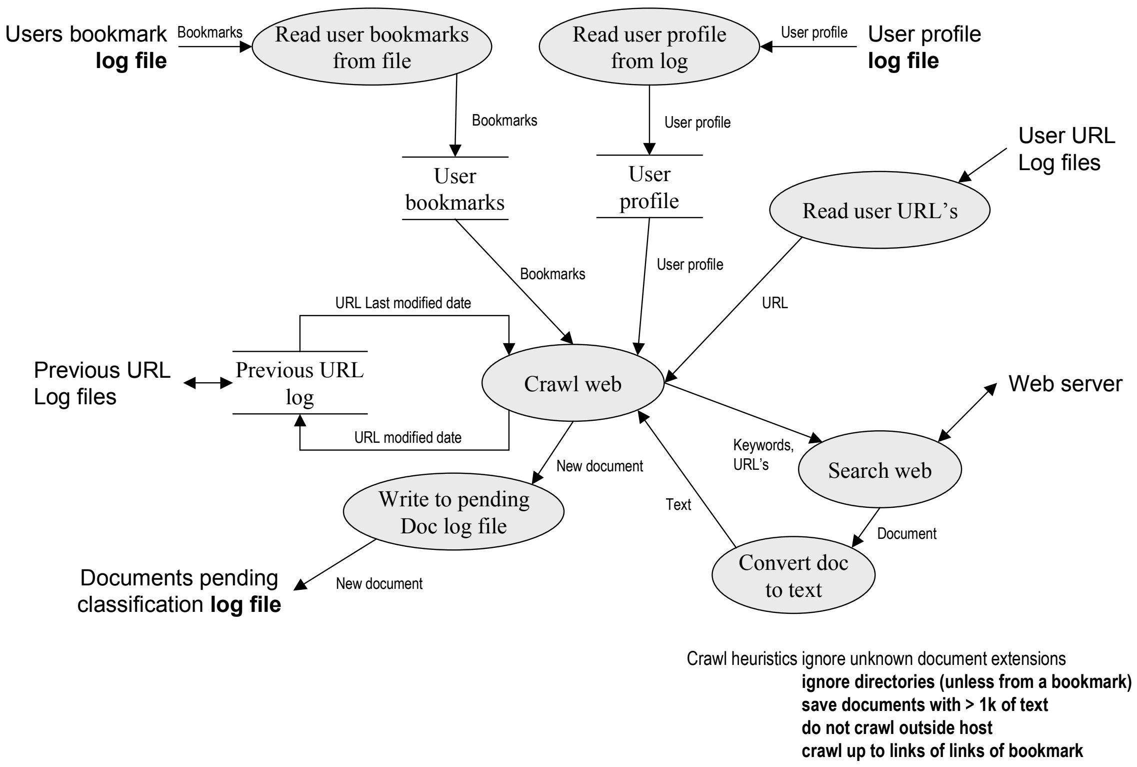

Users bookmark **log file**
Bookmarks
Read user bookmarks from file
Read user profile from log
User profile
User profile **log file**
Bookmarks
User profile
User bookmarks
User profile
User URL Log files
Read user URL's
Bookmarks
User profile
URL Last modified date
URL
Previous URL Log files
Previous URL log
Crawl web
Web server
URL modified date
Keywords, URL's
Search web
New document
Write to pending Doc log file
Text
Document
Convert doc to text
Documents pending classification **log file**
New document
Crawl heuristics ignore unknown document extensions
ignore directories (unless from a bookmark)
save documents with > 1k of text
do not crawl outside host
crawl up to links of links of bookmark

Figure C.6 : Web crawler design

The last process is the web crawler. This loads new documents from the web by crawling bookmarks (loaded into the system at the start) and loading each URL found in the users URL log files. Only PS or PDF documents are kept (compressed versions are decompressed and used too). The documents kept are saved in the pending document store, ready for the classifier to handle them. Figure C.6 describes the web crawler process.

URL log	(date <tab> user <tab> size <tab> URL <newline>)*
(1 file per user)	/URLlogs/<user>_url.log
Register	(user <tab> password <tab> email <tab> port <tab> start date <newline>)*
	/Register.log
	/RegisterBackup.log
User recommendations	(user <tab> date <tab> <type> <tab> title <tab> URL <tab> <topic list> <tab> confidence <newline>)*
(1 file per user)	<type> = “recommendation” | “topic list”
	<topic list> = “(“ (topic <tab>)* topic “)”
	/Recommendations/<user>_recommend.log
User feedback	(user <tab> date <tab> “classification info” <tab> URL <tab> (topic”,”)* topic <newline>)* |
(1 file per user)	(user <tab> date <tab> “interest rating” <tab> URL <tab> interest rating <newline>)* |
	(user <tab> date <tab> “URL jump” <tab> URL <newline>)* |
	(user <tab> date <tab> “current interests” <tab> (topic <tab>)* topic <newline>)*
	/Feedback/<user>_feedback.log
User profiles	(user, (current interests)*, (interest history)*, system data)*
	/Profiles.log
User bookmarks	(bookmark <newline>)*
(1 file per user)	/Bookmarks/<user>_bookmarks.log
Training set	URL, date, (user, topic)*, bag of words
(1 file per URL)	/TrainingSet/<doc id>.arff
Training set update list	(URL, date, (user, topic)*)*
	/TrainingSet/UpdateList.log
Document set	URL, date, last modification date, (topic)*, bag of words
(1 file per URL)	/DocumentSet/<doc id>.arff
Documents pending classification	(URL, date, bag of words)*
(1 file per URL)	/PendingDocuments/<doc id>.arff
Previous URL’s	(URL <tab> date_last_checked <newline>)*
	/PreviousURLs.log
Users URL log checklist	(user <tab> date_checked_up_to <newline>)*
	/WebSearchChecklist.log

Figure C.7 : Quickstep log files

The log file formats are also detailed here. This gives some idea of what sort of data is being stored in each of the log files. Figure C.7 shows this.

Appendix D The Foxtrot system design

This section details some of the design documents generated as part of creating the Foxtrot system.

Nine independent processes make up the Foxtrot system, all interacting to provide the recommendation service to its users. Figure D.1 shows the process map, detailing how these 9 processes (the circles) interact. Basically, each process reads information from a data store, processes it and stores the results to another data store. File locking prevents multiple processes corrupting a data store. Processes are timed to run at periods during the day and night, and sequenced so that the results of one process (e.g. classification process) will be ready for the next process (e.g. profiler).

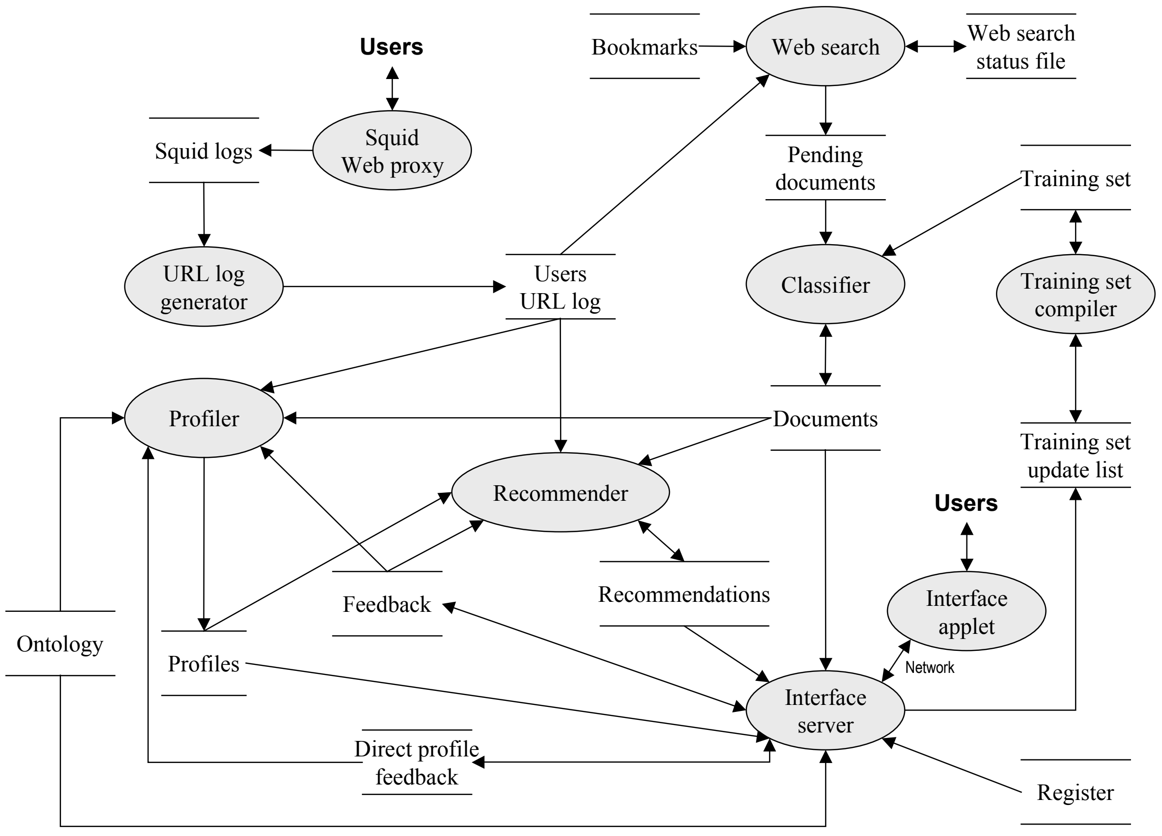


Figure D.1 : Foxtrot process map

Each process is functionally decomposed in the detailed design.

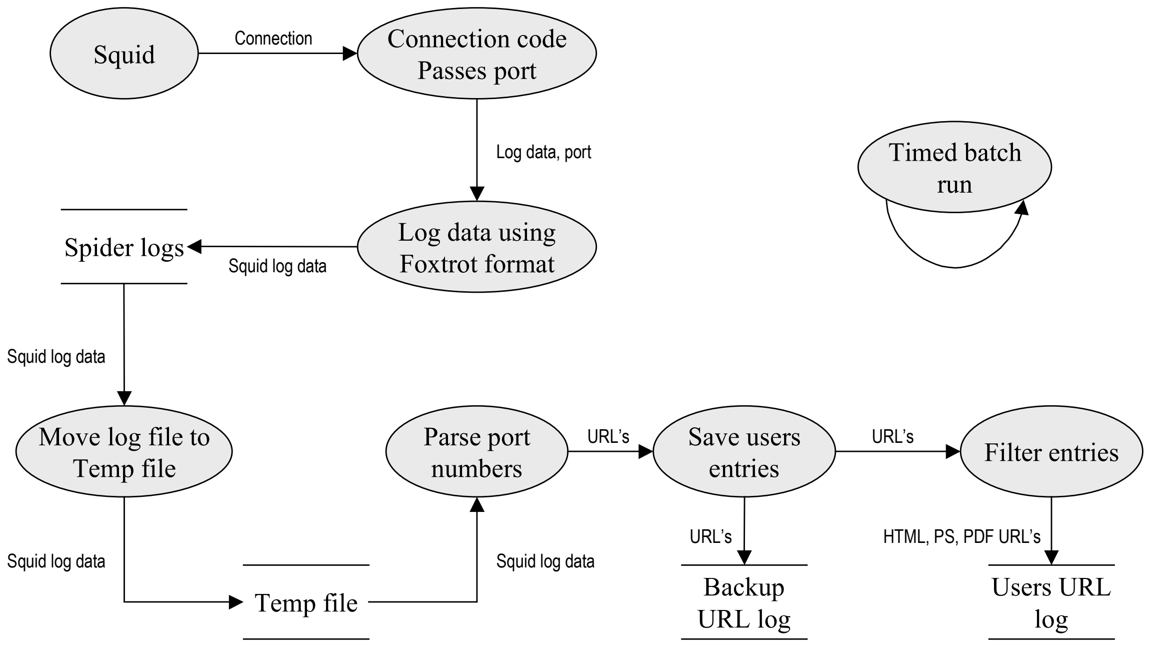


Figure D.2 : Squid web proxy and the URL log generator design

The Squid web proxy process handles requests by users for web pages. Squid is a well-respected third party web proxy, written in C. It is open source and supports the latest HTTP protocols. Foxtrot will use a modified version, logging port numbers with the users normal URL logging information. As Squid rotates its log files a second URL log generator process will regularly copy the log files, parse them and save each URL log entry to a separate user log file. Figure D.2 shows these two processes.

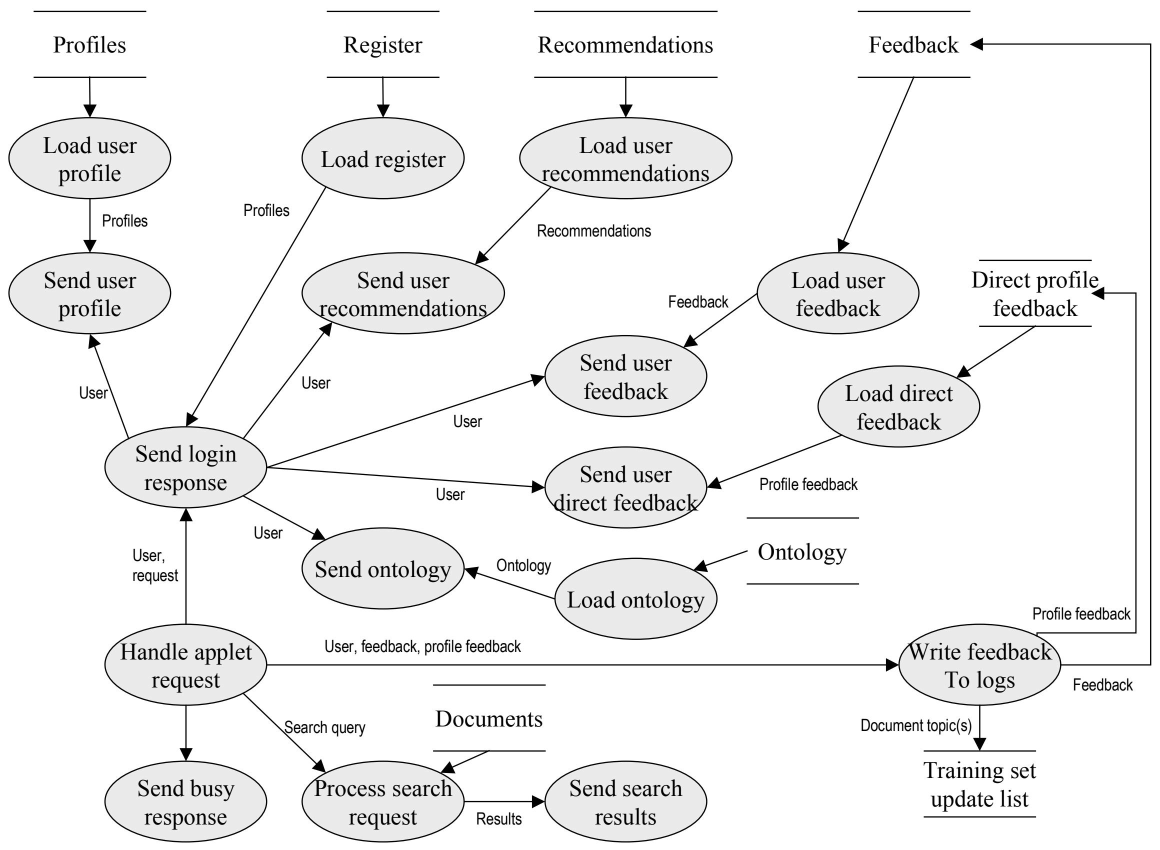


Figure D.3 : Interface server design

The interface server handles requests from the interface applet for recommendations. The client/server set-up is needed to overcome security restrictions applied by most browsers on applets. Once a login request is received the users recommendations, profile and the current topic ontology are sent to the applet. Upon logout, any user feedback is appended to the feedback log (and any topic labels added to the training set update list). The server handles search requests by reading a search query and spawning a thread to handle it. The applet will regularly check on the status of the search query and will be sent a busy signal until it is complete. When ready, the search result (a set of URL's) is returned. Figure D.3 shows the interface server process.

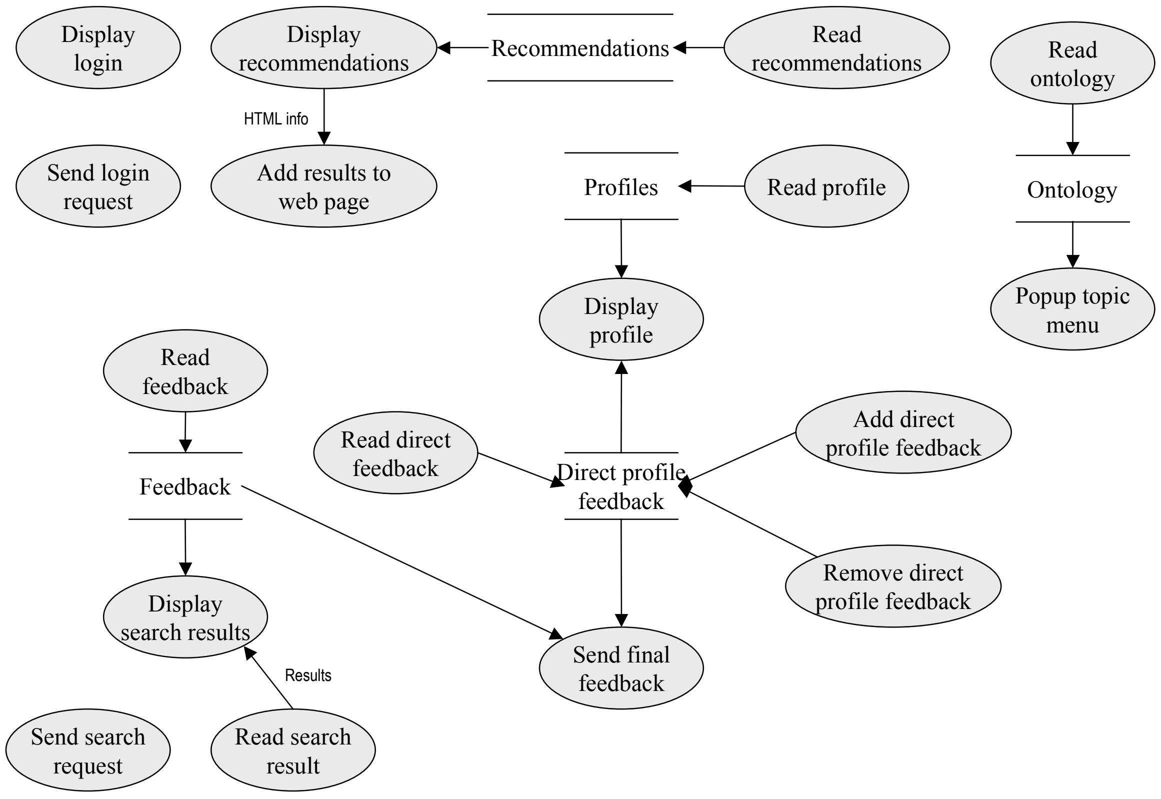


Figure D.4 : Interface applet design

The interface applet is run in a web page, and is Foxtrot's interface to the users. Users must first logon, where they are presented with a search interface and a set of recommendations. These recommendations are sent by the interface server, and are the URL's Foxtrot thinks will be most interesting to the user. These recommendations can be examined, and feedback provided (paper quality, paper topic interest, corrections to paper topic). A popup menu allows topic corrections to be effected. The user can visually see their profile (if they are in the subject group with this feature enabled) and provide absolute reference points for the interest graph; this is the direct profile feedback.

The major functionality of Foxtrot to the users is the search system. Users can enter keywords for title search, topics for category search etc. The final search query is sent to the interface server and processes. The applet regularly checks with the interface server to see if the search query has been handled. A busy indication (such as the

hourglass and a “searching…” message) is displayed until it has, when the full search results are shown (a list of ranked URL’s).
Upon logout the users feedback is sent to the interface server. Figure D.4 shows the interface applet design.

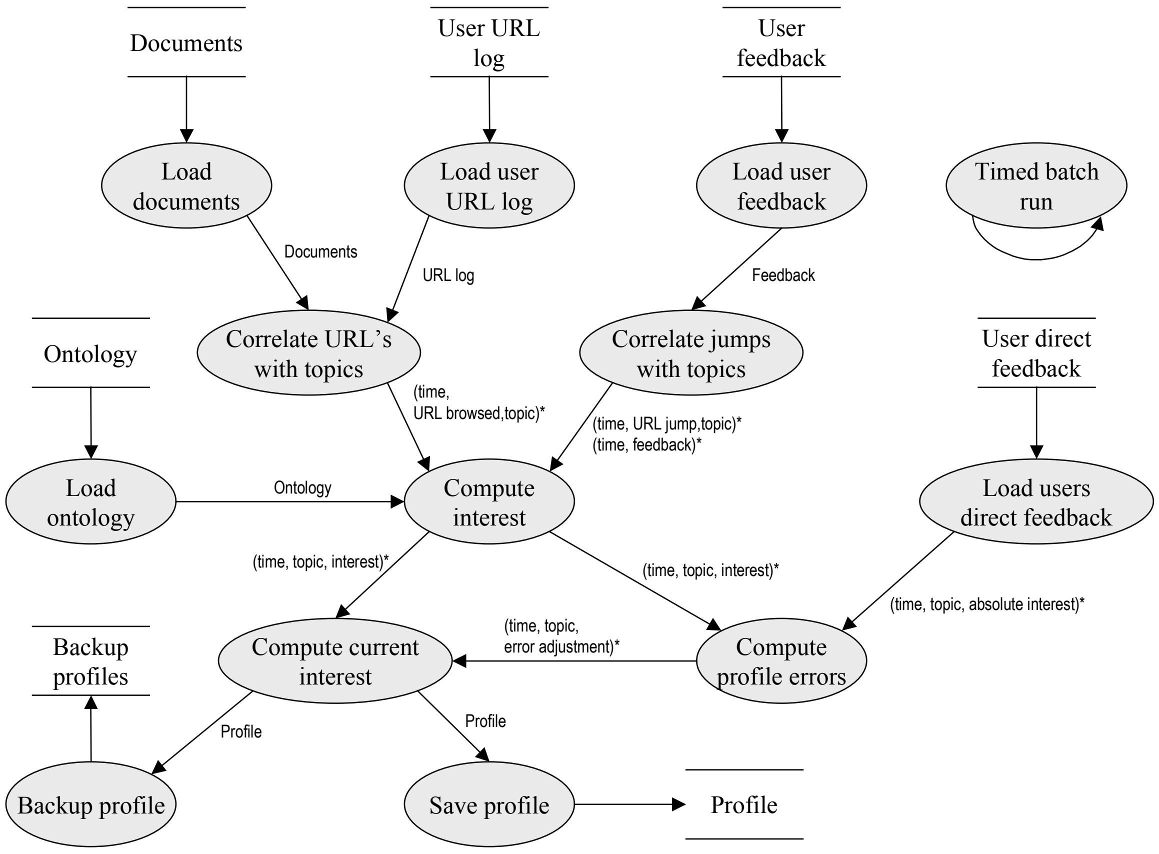


Figure D.5 : Profiler design

The profiler reads the URL logs generated by the web proxy, and correlates them with the classified documents to formulate an interest event time-line. Feedback events are also added as interest events. The interest events are then used to formulate a profile via a time-decay function. If any direct profile feedback exists, this is also used to improve the profile. Newly computed profiles overwrite the existing ones with a backup made for later analysis work. Figure D.5 shows the profiler design.

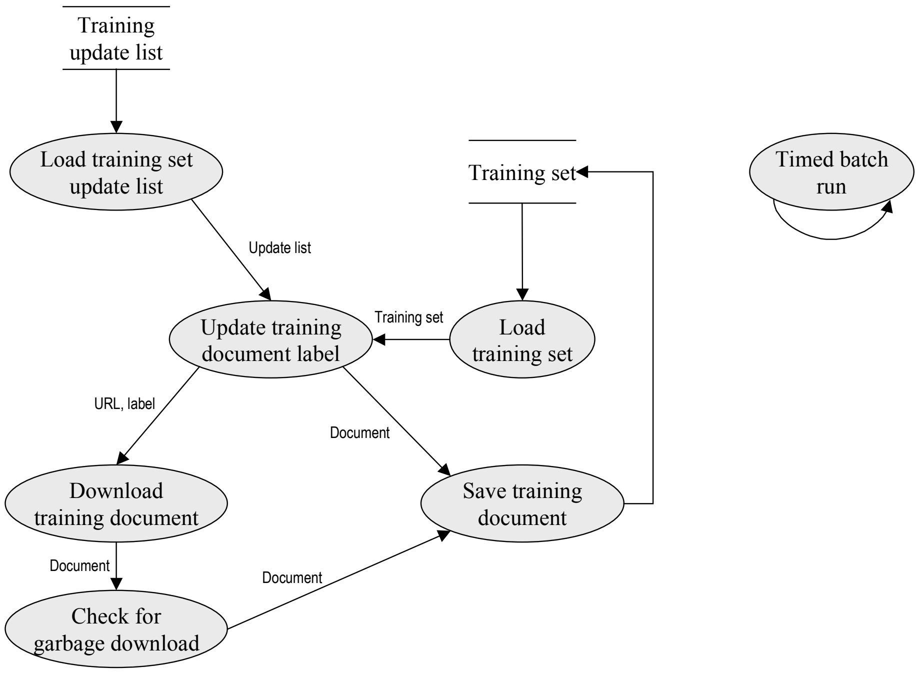


Figure D.6 : Training set compiler design

The training set compiler reads from the training set update list and downloads any URL’s found. They are then given a label and saved in the training set. If the document exists already in the training set its label is updated. Figure D.6 shows the training set compiler design.

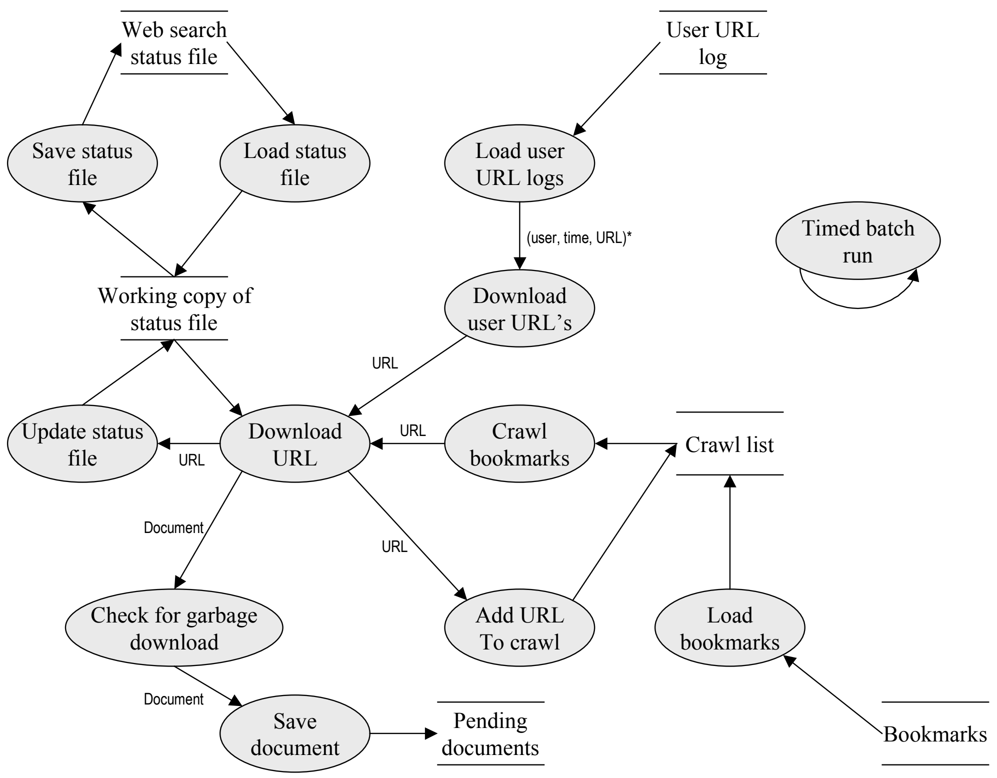


Figure D.7 : Web search design

The web search process loads all URL's browsed by the user and adds them to the pending set. They will be loaded later and classified by the classifier process. A set of bookmarks are also crawled and downloaded if changed. A status file (which includes document links and the last modified date) is kept to avoid re-loading documents that have not changed. A garbage check is made before keeping downloaded documents as the text conversion utility can sometimes fail and produce garbled words. Figure D.7 shows the web search design.

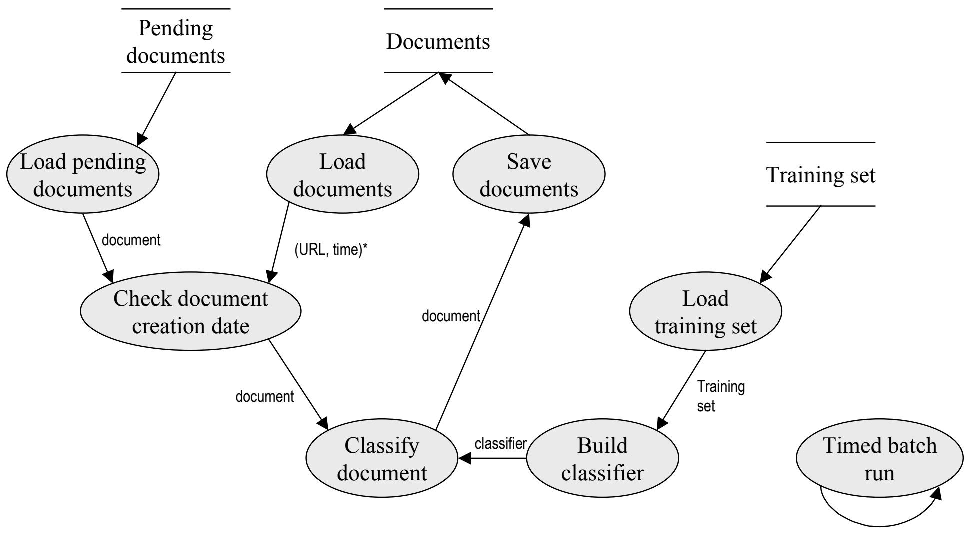


Figure D.8 : Classifier design

The classifier process loads all documents in the pending set and attempts to classify them. The current training set is used to build the classifier. Once built the classifier is applied to all pending documents and labels generated. The labelled documents are added to the document set. Figure D.8 shows the classifier design.

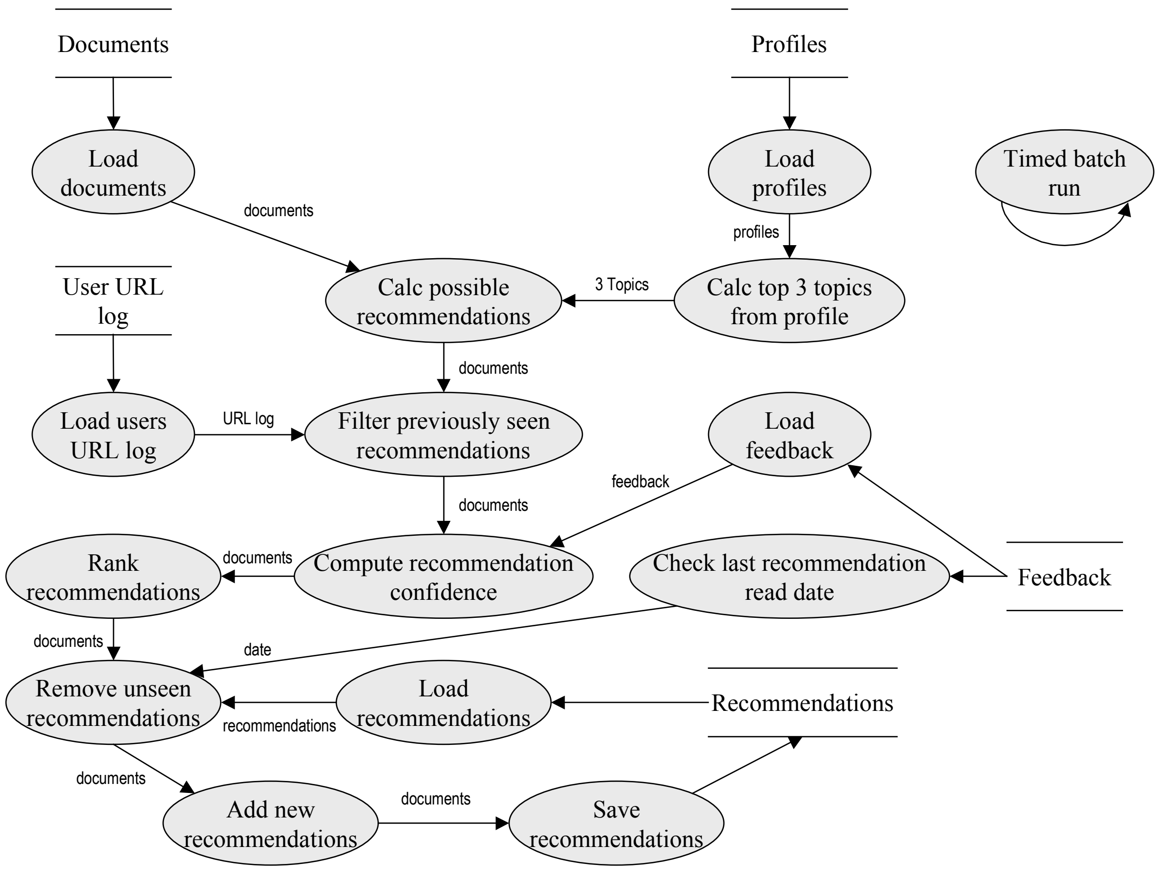


Figure D.9 : Recommender design

Lastly we have the recommender process, which takes the user profiles and formulates a set of recommendations for each user. The top three topics are extracted from the user's profile and correlated with the classified documents. Previously seen documents are removed so that the same thing is not recommended twice. The remaining recommendation set is ranked by recommendation confidence (based on quality feedback and classification confidence). The existing recommendation set is then pruned for any unread recommendations and the new set appended. Thus, the next time the user opens the interface applet this recommendation set will be read. Figure D.9 shows the recommender design.