SWAMI: A Multiagent, Active Representation of a User's Browsing Interests

Mark Kilfoil and Ali Ghorbani
Intelligent and Adaptive Systems Group
Faculty of Computer Science
University of New Brunswick
Fredericton, NB, Canada
[mark.kilfoil,ghorbani]@unb.ca

Abstract

The rapid growth of the World Wide Web has complicated the process of web browsing by providing an overwhelming wealth of choices for the end user. To alleviate this burden, intelligent tools can do much of the drudge-work of looking ahead, searching and performing a preliminary evaluation of the end pages on the user's behalf, providing the user with more information with which to make fewer, more informed decisions. In order to accomplish this task, however, the tools need some form of representation of the interests of the user.

This paper describes the SWAMI system; SWAMI stands for "Searching the Web with Agents having Mobility and Intelligence". SWAMI is a prototype that uses a multi-agent system to represent the interests of the user dynamically, and take advantange of the active nature of agents to provide a platform for look-ahead evaluation, page searching, and link swapping. The collection of agents is organized hierarchically according to the apparent interests of the user, which are discovered dynamically through multi-stage clustering. Results from testing show that such a system is able to follow the multiple changing interests of a user accurately, and that it is capable of acting fruitfully on these interests to provide a user with useful navigational suggestions.

1 Introduction

The continual growth and complexity of the World Wide Web has ironically impacted its effectiveness in a negative way. An individual user must sift through a vast number of pages that are of little or no interest to them to discover pages that address their interests.

Tools have been developed to assist in this process, one of the most successful being the keyword-based search engine, such as Google [Google, 2004]. However, keyword-based search engines require a user to carefully craft their query to be an accurate statement of their information desires, which is often difficult to perform.

Another approach is to build websites that are *personalized* or *adaptive*. Personalized websites allow users to describe themselves (specifically including information desires and form factors) and use this information to modify their responses to suit each user (or group of similar users) individually. Adaptive websites extend the idea of personalized websites by using artificial intelligence techniques to automatically discover the apparent interests and information needs of the user.

All of the above approaches attempt to address the problems with server-side solutions. Alternatively, one might approach the problem from the client-side, using the system closest to the user (and thus in the best position to assess their interests).

This paper introduces the SWAMI system. SWAMI stands for "Searching the Web with Agents having Mobility and Intelligence". This system is a client-side, multi-agent-based approach to personalizing the user experience of web browsing. Section 2 describes the domain of the problem, as well as describing other approaches, including other agent-based solutions. Section 3 briefly describes the architecture of the SWAMI system. The implementation details are described in Section 4. In Section 5, preliminary results gathered from experimental data are included. Finally, Section 6 presents a summary of the ben-

efits and drawbacks of the SWAMI approach, and discusses future directions for research.

2 Background and Related Work

The web is a relatively new phenomenon, and has elevated certain problems to a critical level. In this section, the two most prominent problems of web navigation and web personalization are discussed, and a short summary of current solutions is presented.

2.1 Web Navigation

Because of the large size, dynamic nature and inconsistent structure, the web is difficult to navigate. "Traditional", direct navigation approaches depend on an evaluation of the relevance of the currently viewed page as the best indicator of the value of pages pointed to by the current page. This approach relies upon the benevolence of the creator of the link [Kleinberg, 1999], and the hope that by following a series of related links the user will end up at another cluster of useful pages. This "hope" is described as the "small world" phenomenon, which suggests that a highly complex but interacting system will, over time, evolve paths of a limited number of hops between any two related pages.

When the user has discovered a page of lessening interest to them than a previous page, they return backward to an appropriately interesting (although already viewed) page and go forward from a link on that page (if there is one) until all links from that page have been exhausted, retreating back up another level. This navigation strategy is implicitly promoted by the linear nature of web navigation tools, such as the "back" button of a web browser (see Figure 1).

The traditional strategy closely resembles a depth-first graph search, where leaf nodes are represented by pages of less interest. Effectively, however, the user must go "one page too far" in such a scheme, and travel deeper and deeper distances from the original page they were browsing into possibly uninteresting area. In fact, to return all the way to the original page technically requires an examination of *all* the subgraph contained from the single link away they had taken. Given the highly-connective nature of the Web, this suggests that the user will spend more time in distant pages than in pages more closely connected to the original. This, intuitively, is the opposite of the desired result, as pages directly connected to the current page are most likely to be the most relevant pages to it [Lieberman, 1995].

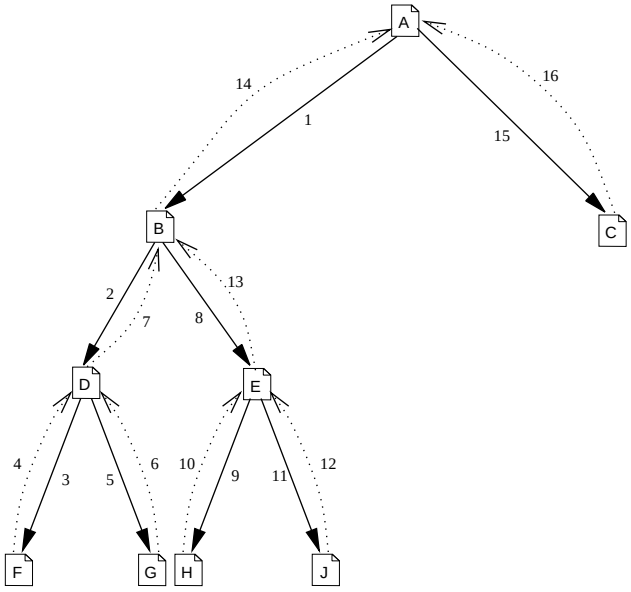

Figure 1: Traditional web navigation works like depth-first search of a graph.

2.1.1 Search Engines

A completely alternative approach is to use a search engine, which, in effect, reconstructs the graph of the Web, reconnecting all the distant pages together into a single layer. In this way, more relevant pages become more likely at an earlier stage of browsing. (See Figure 2.) In the case of Yahoo [YAHOO!, 2004], this rearrangment is done explicitly through a hierarchical, soft categorization of website links. By contrast, Google [Google, 2004] builds a response page (effectively the top-level of a tree or entrace to a graph) dynamically around a set of initial keywords in a query.

Once the user has selected a link from a search engine, however, they are out of the arena of that technology, and browsing returns to the traditional strategy. Thus, this technology produces only a one-shot or one-level navigational benefit, not an ongoing navigational support.

An additional criticism of search engine approaches is that the criteria for the evaluation of results is very specific: the keywords of the request are the only measure of relevance to the user that the system can use, although there are additional measures of the relative importance of a page (some partially dependant

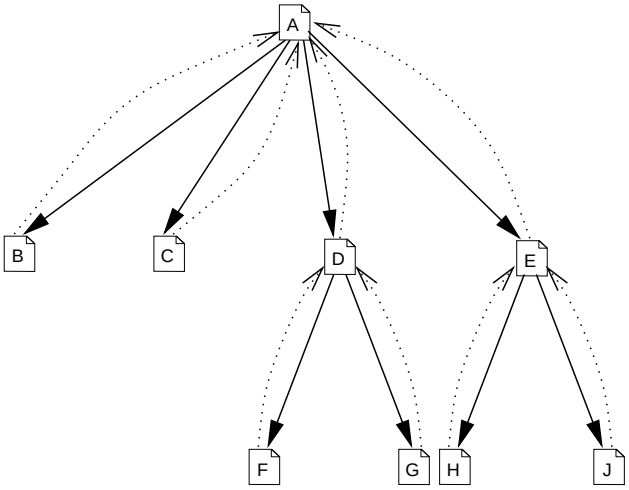

Figure 2: A Search Engine effectively reconfigures the web into a shallow tree.

of the particular request made) [Kleinberg, 1999]. In other words, the result evaluation does not take into account the full nature of the user, such as prior information or additional interests. (Although Google has recently introduced a beta service which seems to allow the user to provide the search engine with a more complex view of the requester.)

2.1.2 Adaptive Websites

Adaptive websites take a highly personalized approach. They use knowledge about the specific user to modify both the presentation [Kobsa et al., 2001] of individual pages and/or the navigation from one page to another [Brusilovsky, 1996]. In this way, they can be seen to either add additional links between pages of relevance to the user (Figure 3) or do a similar rearranging of the graph to the search engine (Figure 4), although beyond just a single level of rearrangement and navigational support. These links may come from a mining of a large set of pages within a scope [Kleinberg, 1999], from an online search and evaluation scheme [Lieberman, 1995] from an external knowledge of the structure of the domain [Freitag et al., 1995, De Bra and Ruiter, 2001], from other, similar users through collaborative recommendation [Mobasher et al., 2002, Lieberman et al., 1999, Cosley et al., 2002] or through some combination of these techniques [Balabanovic and Shoham, 1997].

Prominent examples of adaptive web systems in-

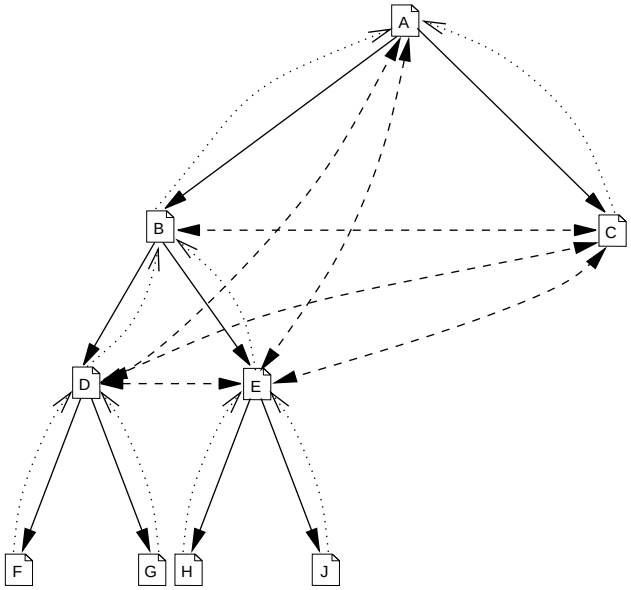

Figure 3: Personalized websites can change the navigational structure of a website by adding links between relevant pages based on user interest.

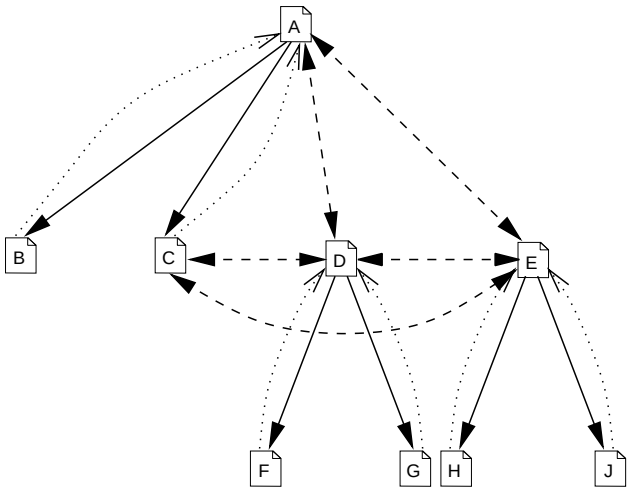

Figure 4: An alternative view of of Figure 3; the website is restructured based on interest.

clude WebWatcher [Joachims et al., 1997], AHAM [Lieberman, 1995], IfWeb [Asnicar and Tasso, 1997] and AVANTI [Fink et al., 1997]. Each of these systems provides server-side adaptive navigation or presentation based on perceived user characteristics.

Server-side solutions for adaptive websites also offer the possibility of collaborative recommendations, where knowledge about groups of users can be used to make suggestions to individuals who are members of a group. Groups might be arbitrarily chosen – such as the group of a person and his friends – or created through observations of common patterns of behaviour or common attributes.

Server-side solutions, however, are generally limited to a single website or set of close websites, something to which the search engine approach is not limited. Client-side personalized approaches, on the other hand, can work across all websites, but do not have the benefit of an internal view of the website (to allow adaptations based on non-disclosed information) or collaborative recommendations (because there is no common place for all users). Client-side solutions include Letizia [Lieberman, 1995] and Personal WebWatcher [Mladenic, 1996], and to a certain degree the proxy-based system PVA [Chen and Chen, 2002].

The SWAMI system is based as a client-side solution, but it allows interaction with peers and with internal sections of websites by allowing parts of its representation to be mobile, and move to a location to interact with the mobile parts of other users' SWAMI, or with representatives of an SWAMI-aware website.

2.2 Personalization & User Representation

Personalization on the Web means to modify the contents in or navigation on a web page to reflect the particular user who is viewing it. It is adaptive to the user's characteristics or behaviour, responding in a way to enhance the user's experience.

In order to perform personalization, it is necessary to consider the way the user is represented within the system, and the information which is used to populate that representation. This information may include a profile of user demographics, a history of their experience, or explicitly stated information goals, for example. Generally, it can be said that the primary factor in a decision of whether a user likes a page is that it satisfies one of their *interests*.

Interests can be generally characterized into three kinds: **long-term interests**, which are stable and rarely changing, although at a particular instant may be unexpressed; **short-term interests**, which are sudden and strong, but vanish quickly, never to return; and **periodic interests**, which have the qualities of both long- and short-term interests, in that they are strong for short periods of time and relatively unimportant for the rest of the time. An example of a long-term interest would be cooking; a short-term interest might be real estate (because once the place is bought, you never need to look at it again); a periodic interest might include the Olympics or national elections on a long scale, or holidays or anniversaries on a shorter scale.

Each of these interest types suggests a certain time scale, which brings up another problem: interests change over time. Interests may grow or wane in importance to the user; new interests may be added and old interests removed.

Because specifying interests is difficult (perhaps even impossible) for a user to express, the approach was taken in SWAMI for the interests to be discovered from the browsing behaviour of the user. This approach has proven to be effective in many cases [Pazzani and Billsus, 1997, Chan, 1999, Schwab et al., 2000]. By having the system continually learning about interests from the user's ongoing browsing behaviour, the problem of changing interests is addressed.

[Godoy and Amandi, 2002] presents a general architecture for discovering and maintaining a user profile in agent terms. This architecture suggests that users have multiple interests of varying levels of detail, and organizes these topics in terms of a hierarchy. It is also recognized in this architecture that user interests are not static, but tend to both change and recur over time. In their architecture, they suggest an explicit "temporal context" might be used to modify the strength of suggestions about particular topics at a given time.

2.2.1 SWAMI User Modelling

SWAMI develops a model of the users apparent interests in order to make forward evaluations, user-centric web searches and navigation suggestions about pages to visit.

A similar approach to [Godoy and Amandi, 2002] has been taken in SWAMI, but with a significant difference: where in [Godoy and Amandi, 2002] an externally organized hierarchy was created, pages placed in that hierarchy, and the user's interests be-

ing taken from a subset of that hierarchy, in SWAMI the hierarchy is developed entirely from scratch, allowing it to be a customized size to reflect the user's interests.

3 Design & Architecture

SWAMI aims to provide web users with a personalized representative of their interests who can browse on their behalf and provide recommendations about what pages would be most interesting to be viewed. There are several implications and additional considerations in order to build an architecture with that goal in mind. This section discusses first these design considerations, and then the architecture created for this application. Implementation details can be found in the next chapter.

3.1 Design Considerations

The design process of the SWAMI system is guided by several considerations. These include:

1. **The system should not require the user to explicitly state their interests.** This implies that the system must be capable of learning the user's interests from observation, rather than from an explicit query.

2. **A user may have multiple different interests.** Many systems consider the user interests to be all related to some degree. In SWAMI, it was felt that this was an unrealistic assumption, so a model that allows for multiple competing interests was considered instead.

3. **A user may have several related interests ("sub-interests") within the context of a general interest.** Topic hierarchies and ontologies have been proposed and used in adaptive web agent systems before [Chen and Chen, 2002, Godoy and Amandi, 2000]. When combined with the previous design goal this naturally leads to multiple, independent hierarchies.

4. **A user's interests change over time.** In particular, interests may be short term, long term or recurring. SWAMI allows for this by introducing an element that increases with recent use and decreases with inactivity (the "age" of an agent) in the representation of a user's interests.

5. **Those interests which receive more recent attention (i.e. that are more "active") should be considered more important than those that receive less attention or those which have not received recent attention.** In SWAMI, an agent's longevity and ability to take action are tied to the quantity of page visits that it represents and the recency of those page visits. The better the agent's longevity and ability to take action are, the more likely that they are to search for new pages for the user.

6. **The system should be capable of evaluating a page based on the user's interests.** Each agent within the system carries with it a weighted vector of keywords that it can use to evaluate a potential page. An agent within a hierarchy representing an interest can also refine its evaluation based on what it's parent agent considers important.

7. **The system should be able to search for pages that might be of interest to the user.** Agents that accumulate a sufficient level of "wealth" can create search agents that search and evaluate pages for the user in parallel with their own browsing.

8. **The knowledge of evaluated pages should be available not only to a particular user, but to all users within a community.** The introduction of the "Rendezvous Server" allows page recommendations to be shared with others.

9. **Since the controllers of a particular website know their own material best, it should be possible to consult with them (or their representatives) for page recommendations.** The SWAMI architecture describes local expert agents that can act as expert representatives of a website. These representatives can be consulted for recommendations, taking advantage of any hidden context they might have.

3.2 SWAMI Architecture

The SWAMI system consists of a front-end interface, a user representation, and components which perform page searching. It is implemented using a (custom) multi-agent system (see Figure 5). This section describes each of these three components in more detail.

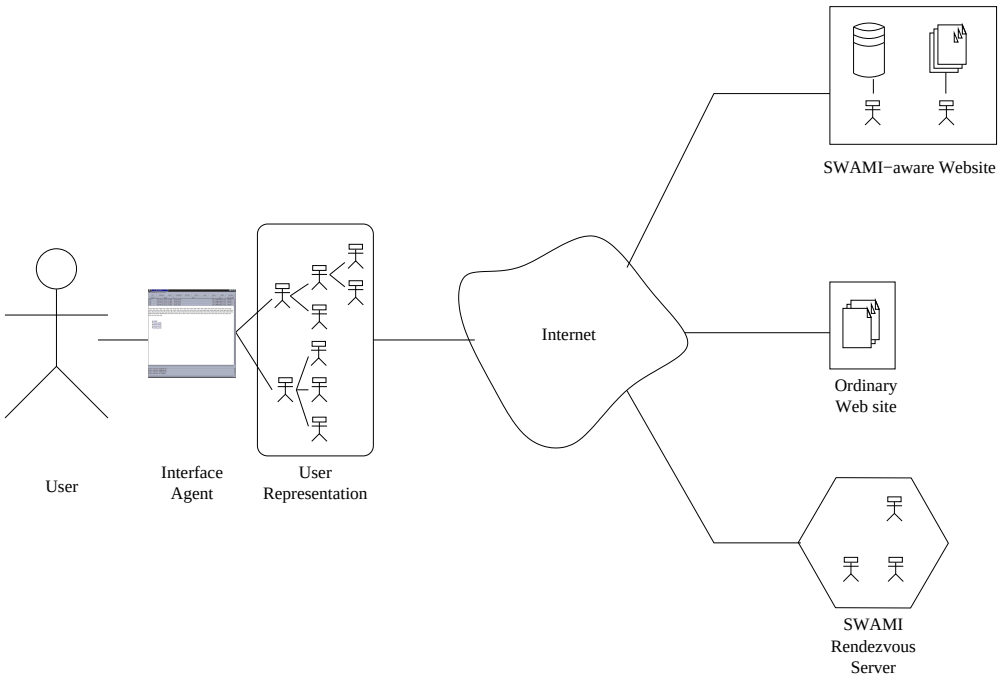

Figure 5: A high level view of the SWAMI system.

3.2.1 The Interface

The user interacts with the system using the SWAMI interface agent. The interface agent is currently integrated into a simple browser, allowing the agent to observe user activity easily and report the search and evaluation results of the user representation to the user. The browser also allows the user to display the agents currently representing them.

3.2.2 The User Representation

The user is represented by a hierarchically-arranged collection of agents. Each representation agent represents a cluster of pages the user has viewed, with the hierarchies representing the relationships between clusters.

The hierarchies of agents is created using an online, dynamic clustering technique. An agent collects pages as they are viewed by the user that are similar to the pages it has already gathered. The agent continually checks the tightness of its cluster, and if it is too loose (beyond a threshold), it will attempt to split the collection of pages up into tighter subgroups. If it is successful, it creates (or "hires") new agents to represent the subgroups. These agents are positioned below the original agent, so that incoming pages are first examined by the original agent, and then may be passed down to the more specialized sub-agents, and so on, until the best match has been made.

Initially, the interface agent collects all pages until a distinct group is discovered, forming the first representation agent. If no current agent is representative of a given page, the interface agent holds on to it until a new group manifests itself. Each of these top level groups is referred to as a "corporation", and represents a major interest of the user.

Each agent has a measurement of "wealth", which reflects the importance and relevance to the user of the cluster the agent represents. Equation 2 shows the formula used to calculate wealth. This combines the agent's size (`sizeActivity`), the success the agent has had in finding new pages for the user (`search`), the success the agent has had in having found pages accepted by the user (`acceptance`) and a history momentum which allows an agent to rest on its laurels briefly (`wealth(t-1)`).

When an agent's wealth is reduced below a threshold, the agent is removed from the hierarchy and moved into a holding area. In this way, agents which are not producing useful assistance are prunes from the hierarchy. However, to represent periodic interests, these agents are not immediately deleted.

$$\begin{aligned} \text{wealth}(t) = & \quad \alpha \times \texttt{sizeActivity} \\ & + \quad \beta \times \texttt{search} \\ & + \quad \phi \times \texttt{acceptance} \\ & + \quad \gamma \times \texttt{wealth}(t-1) \end{aligned} \tag{1}$$

Rather, they remain in the holding area, continuing to decay, until one of two conditions is satisfied: either they are the best representative for a new page the user views, or they represent a newly discovered subcluster better than a blank agent. In the first case, they become the head of a new corporation; in the second case they are simply added into the hierarchy at the appropriate point. This also allows subclusters to migrate to the most appropriate place; for example, a "Mexican cooking" agent might be retired from beneath the general "cooking" agent, but later be rehired under a "Mexican culture" agent. Note that agents are not labelled in this way; this is merely for illustration purposes.

3.2.3 The Search Components

When a representation agent reaches a sufficient level of wealth and experience, it can create search agents to work for it. Search agents take criteria from the representation agent (the set of word features the representation agent has used to form its cluster, for example) and attempts to find and evaluate pages on its behalf.

Four types of search agents have been considered for the system: agents that search the links from existing pages, agents that leverage search engines as a source of potential recommendations, agents that consult with local topic experts for recommendations, and agents that consult with other search agents.

Link-following Search Agent

The link-following search agent follows links from pages the user has already viewed and evaluates them based on its criteria. The agent, in a way, acts like a user in its pattern of browsing, following a similar pattern as discussed in Section 2.1.

Search-engine based Search Agents

The search-engine based search agent can submit different combinations of word features to a search engine and evaluate the results. In this way, it can take advantage of the massive database of knowledge available to a search engine, but provide the personalization that the search engine lacks.

Topic expert consulting Search Agents

The topic expert consulting search agents are mobile agents which can travel to SWAMI-aware websites and interact with topic expert agents representing the web page owner. These topic expert agents may have access to information that cannot be gathered from simply browsing the pages, and may be in a better position to provide recommendations. For example, the topic expert agents may know about arbitrary groupings of pages that do not have labels on the pages themselves.

Collaborative Search Agents

The collaborative search agent seeks to take advantage of the browsing behaviour of people with similar interests. It travels to a host (referred to as the "rendezvous server") where it can interact with agents representing other people. There, they can swap recommendations based on how similar the agents are to each other. Another type of agent, the rendezvous hosts, remain in the rendezvous server at all times, interacting with all the visiting search agents and collecting all recommendations that they have. The rendezvous host becomes a "memory" for the rendezvous server, so that not all interactions between agents need to be synchronized.

4 Implementation

SWAMI was realized by a custom multi-agent, mobile agent system (MAS) [Wooldridge, 2001] written in Java. This section discusses some of the implementation details.

4.1 Web Page Representation

The SWAMI system is an exploration of user modelling, hierarchical and incremental clustering, multi-agent systems and search methods. The task all this is focused on is the browsing of web pages, so it is necessary to discuss the composition of a web page, how web pages are viewed within the SWAMI system, and how pages can be compared.

4.1.1 Composition

A web page consists of a document written in the HTML language, referred to by an address or URL. This document contains a mixture of text data and images organized both structurally and visually.

HTML is a markup language, and is only a partially semantically-structured language. That is, the

tags used in the language sometimes indicate meaningful semantics (e.g. the HEAD, BODY and TITLE tags), sometimes indicate visual layout or decoration (e.g. the TABLE tag), and sometimes are intended to suggest semantic importance but often are used for visual reasons instead (e.g. the EM, STRONG and the various header tags such as H1 and H2). It also allows the embedding of programming languages (e.g. Javascript) and visual extensions (such as the cascading stylesheet (CSS) extension). HTML also allows inline references to images and other binary components (e.g. through the IMG and OBJECT tags), as well as references (links) to other pages.

HTML is really the client-side language, and HTML files may be composed or generated on the server. These dynamic pages only exist for the length of the viewing, and do not have any independent existence. Thus, the HTML page generated from a particular URL may be different between viewings (or even different every time it is viewed). In this initial version, only static (unchanging) web pages that are the same each time the same URL is used are considered.

4.1.2 Representation

The most important part of a web page, from the perspective of the system, is the textual data it contains; specifically, the words with which the document is written with.

While tags sometimes indicate increased importance of a particular word within a document D, initial investigation indicated that this importance is unpredictable, given the lack of strict adherence or interpretation of the use of tags to mark up specific words. Therefore, the frequency of a word occurring within a document regardless of markup is used as the basis for page description.

The interpretation of a word within a sentence is a complex problem requiring a sophisticated language-specific model. To increase the throughput of the system's analysis of web pages and allow the system to be (largely) language-agnostic, sentence structure and punctuation is ignored for the processing of pages.

Thus, the basic representation is the well-known *bag-of-words* model. This representation collects all the unique words from a document and notes the frequency of each. The procedure for collecting these words is called *text cleaning*, because it requires that the text be transformed. The steps for transformation are as follows (see also Figure 6 for an example of this process):

1. **Retrieve the raw text:** The system retrieves the raw text of an HTML document for processing.

2. **Extract words:** The document is scanned with an HTML parser, and HTML markup is removed; at the same time, punctuation is removed and all letters are transformed to lowercase, leaving behind only what appear to be words, i.e. consecutive sequences of letters separated by spaces.

3. **Word stemming:** It has been widely accepted that words that share the same root but have different suffixes are more alike than not; thus, words with the same root should be considered the same word. For example, the words "company" and "companies" both refer to the same root, "compani". The process of discovering these roots is, however, a language-specific issue. An implementation [Java Stemmer, 2005] of the well-known Porter stemming algorithm [Porter, 1980] was used to stem words for the English language.

4. **Stoplist elimination:** Since word frequency is the basis for understanding the importance of a word to a document, there is a problem with a significant set of words used as lingual structural glue but which have very little interesting value. These words include "the", "and", "but", and "in", for example. Thus, a manually-created list of "uninteresting words" called a *stoplist* was applied to eliminate those less-than-significant but frequent words from the text data.

5. **Unique word counting:** All of the word forms are combined into a single group (referred to as a *term*) and counted, getting the word frequencies. The (unnormalized) frequencies for the example can be seen in Table 1. Terms that appeared only once in this document were omitted for brevity.

4.1.3 Feature extraction

Once the words have been extracted from a document, it is necessary to determine which words are significant descriptors of the page, that is to extract the features from the words.

Features are terms t to which a weight has been associated, indicating its importance. This weight is determined in part by the frequency f_t of the

(a) The key to the technology is a computer-guided nozzle that deposits a line of wet concrete, like toothpaste being squeezed onto a table. Two trowels attached to the nozzle then move to shape the deposit. The robot repeats its journey many times to raise the height and builds hollow walls before returning to fill them.

(b) the key to the technology is a computer guided nozzle that deposits a line of wet concrete like toothpaste being squeezed onto a table two trowels attached to the nozzle then move to shape the deposit the robot repeats its journey many times to raise the height and builds hollow walls before returning to fill them

(c) the kei to the technolog is a comput guid nozzl that deposit a line of wet concret like toothpast be squeez onto a tabl two trowel attach to the nozzl then move to shape the deposit the robot repeat it journei mani time to rais the height and build hollow wall befor return to fill them

(d) kei technolog comput guid nozzl deposit line wet concret toothpast squeez onto tabl trowel attach nozzl move shape deposit robot repeat journei time rais height build hollow wall return fill

Figure 6: Various stages of text cleaning: (a) the raw text; (b) the text turned into simple terms, with punctuation removed; (c) the text after stemming has been applied; (d) the text after a stoplist has been applied.

term considered. In the context of just the page itself, the weight w_t is a normalized value of the frequency. Within the context of a group of documents, the weight is frequently based on of a measure called the *term-frequency/inverse document frequency* or *TFIDF* value of the term. TFIDF has the following formula:

$$w_t = TFIDF(t) = f_t \times \log\left(\frac{N}{DF(t)}\right) \qquad (2)$$

$DF(t)$ is the *document frequency* of term t; that is the number of documents contained within the set that contains that term.

TFIDF penalizes terms that occur in many documents; a term is more valuable to a particular page, in this context, if it occurs rarely.

In the SWAMI system, the context is more complex. Later, the variation of TFIDF that considers this more complicated context will be discussed.

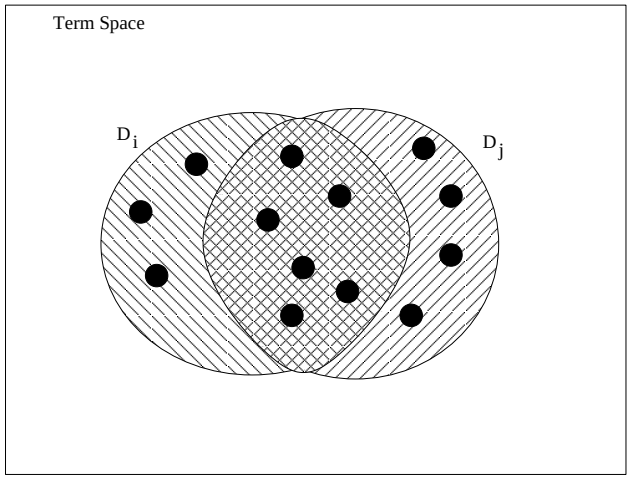

Figure 7: A graphical depiction of the Jaccard similarity measure.

4.1.4 Comparison

Now that documents have been represented as a collection of weighted features, it is easier to compare documents against each other for similarity. A number of similarity measures were considered for this role; in particular, variations of the Jaccard similarity measure were closely examined.

In the Jaccard measure, each document is represented as a set of terms. The larger the overlap between the two documents, the more similar document D_i and D_j are to each other. (See Figure 7.) The standard weighted Jaccard similarity measure formula is shown in Equation 3.

However, most comparison methods are computation-intensive, and do not significantly improve upon the basic cosine similarity algorithm. The cosine method treats the features of documents as vectors in a multidimensional space, and calcu-

print	8		digit	8		camera	8		robot	7
stori	6		materi	6		build	6		subscrib	5
scientist	5		khoshnevi	5		edit	5		technolog	4
hous	4		degussa	4		wall	3		search	3
scienc	3		review	3		construct	3		california	3
world	2		week	2		web	2		univers	2
time	2		test	2		success	2		straw	2
southern	2		servic	2		round	2		return	2
relat	2		prototyp	2		printer	2		precis	2
power	2		nozzl	2		newscientist	2		mud	2
lynn	2		live	2		job	2		inform	2
industri	2		guid	2		greg	2		financ	2
engin	2		drug	2		deposit	2		creat	2
contour	2		contact	2		concret	2		chang	2
canon	2		camcord	2		built	2		break	2
believ	2		behrokh	2		archiv	2		architect	2
abl	2									

Table 1: Terms collected from the article from which the sample was taken.

$$Jaccard_{weighted} = \frac{\sum_{\forall t \in D_i, D_j} w_{D_i,t} \cdot w_{D_j,t}}{\sqrt{\sum_{\forall t \in D_i} w_{D_i,t}^2} + \sqrt{\sum_{\forall t \in D_j} w_{D_j,t}^2} - \sqrt{\sum_{\forall t \in D_i, D_j} w_{D_i,t} \cdot w_{D_j,t}}} \quad (3)$$

lates the angle between the vectors. As the angle decreases, the similarity between the two documents increases.

As seen in Figure 8, each document is represented as a vector in the similarity space. The smaller angle a is, the more similar document D_i and D_j are to each other. The formula for this calculation is

$$\begin{aligned} \cos(D_i, D_j) &= \frac{D_i^T \cdot D_j}{\|D_i\| \times \|D_j\|} \quad (4) \\ &= \frac{\sum_{\forall t \in D_i, D_j} w_{D_i,t} \cdot w_{D_j,t}}{\sqrt{\sum_{\forall t \in D_i} w_{D_i,t}^2} \times \sqrt{\sum_{\forall t \in D_j} w_{D_j,t}^2}} \end{aligned}$$

where $|D|$ is the size of the document (number of terms on the page).

Note that in order for two documents to be properly compared, the feature weights need to be recalculated within the same context.

4.2 User representation

The representation of a user within the SWAMI system is as a collection of interests of varying degree and depth. These interests are organized into hierarchies, each hierarchy representing a broad topic of interest, with the depth of the hierarchy representing specific levels of detail (see Figure 9).

This is similar to the approach taken in Personal View Agents (PVA) [Chen and Chen, 2002] and PersonalSearcher [Godoy and Amandi, 2000], which both use a hierarchy of topics to describe a user's interests. Unlike PVA, however, this hierarchy is not statically created *a priori* in SWAMI but generated based on observed evidence, which makes the profile more specific to the individual user.

Unlike both systems, SWAMI does not force all topics into the same hierarchy, but creates multiple hierarchies, allowing more divergence between topics. An alternative view is that SWAMI may be considered to omit several levels from an absolute, complete hierarchy, levels that are introduced artificially to coax widely separated topics within the same eventual ancestor topic.

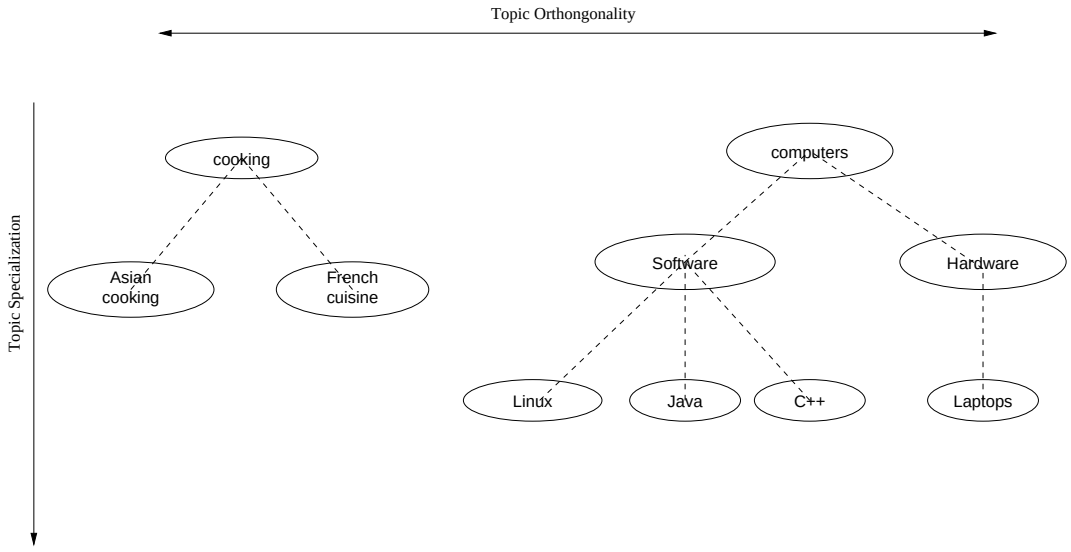

Figure 9: User interests can be characterized as belonging to orthogonally separate groups with differing levels of specialized interests within each group.

These hierarchies are created through a process of incremental, online clustering, which uses local decisions and optimizations. Such an approach is not ideal, as there may be connections between non-contiguous events that are difficult (even perhaps impossible) to observe linearly. However, by allowing the importance of a given cluster to diminish over time or improve with recent activity, as well as allowing a limited set of reconfigurations to occur over time, it is hoped that long-term exposure of the system will show a steady progress in the direction of the ideal.

4.3 Clustering

There are two kinds of clustering decisions that are to be made in this system:

- **Placement.** Given a set of existing clusters and a new page, the system must decide which (if any) of those clusters best represents the page.
- **Specialization.** Given a set of pages within a cluster, the system must attempt to discover if there are one or more subclusters of pages. Such a subcluster better represents a subset of pages than the whole cluster does. Since this decision may produce one or more clusters that divide up the pages of an existing cluster, it is also referred to as *splitting* a cluster.

  Similarly, since subclusters can change, this also refers to deciding whether pages should move from the general parent cluster to a more specific subcluster. This process is called the *redistribution* of pages within a cluster hierarchy.

4.3.1 Definitions

In order to describe the algorithm more formally, a few definitions are needed.

(a) Document Cluster

A *document cluster* C is a collection of documents that all share an acceptable level of similarity:

$$C = D_1, D_2, \ldots, D_n \tag{5}$$

$$\forall D_i, D_j \in C, sim(D_i, D_j) >= \sigma \tag{6}$$

or at least the average similarity is above a threshold :

$$\frac{\sum_{\forall D_i, D_j \in C} sim(D_i, D_j)}{|C|} >= \sigma \tag{7}$$

Ideally, that similarity is maximized for each document in this cluster:

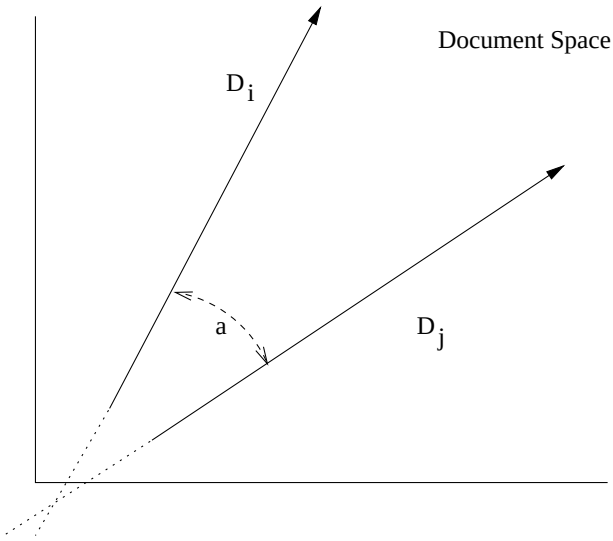

Figure 8: A graphical depiction of the cosine similarity measure.

$$\forall D_i \in C_x, C_x = \underset{\forall C_j \in \mathbb{C}}{\operatorname{argmax}}\, avgsim(D_i, C_j) \quad (8)$$

where $\mathbb{C}$ is the set of all clusters.

(b) Cluster Centroid

To save time, a cluster may be represented by a set of features in much the same way as a document. These creates a "virtual" document (referred to as a *centroid*), which is the point with the most similarity (or least distance) from all other pages within the cluster:

$$\forall D \in C, D_i = \underset{\forall D_j \in C, D_j \neq D_i}{\operatorname{argmax}}\, sim(D_i, D_j) \quad (9)$$

Since each page is represented as a collection of terms, the ideal centroid may be described as the average of all pages, $\overline{D}$:

$$\overline{D} = \left\{ \frac{\sum_{\forall D_i \in C} D_i.w_1}{|C|}, \ldots, \frac{\sum_{\forall D_i \in C} D_i.w_n}{|C|} \right\} \quad (10)$$

The set of all clusters being considered at a particular time is denoted by $\mathbb{C}$.

4.3.2 Calculating Term Weights

To properly compare the incoming document with each cluster, it is necessary to recalculate the weights of the terms in the document within the context of the potentially owning cluster.

To denote that a document D is being considered in the context of a cluster C, the notation D_C is used.

Simple Cluster Context

In the simplest case, the term weights are calculated using a discrimination function such as TFIDF which promotes features that uniquely identify that document. That is:

$$\forall t \in terms(D): \quad (11)$$
$$w_t = \text{TFIDF}_C(t) \quad = \quad f_{D,t} \times \log\left(\frac{N_C}{\text{DF}_C(t)}\right)$$

Hierarchical Cluster Context

In a hierarchical cluster, the terms of the parent cluster are already significant and have already been taken into account to form the parent cluster. To distinguish a child cluster and properly focus and specialize its terms, it is necessary to change the weighting scheme of the terms to take into account the fact that words from the parent are going to be present in all of the child cluster's pages.

The modified TFIDF-based calculation is as follows:

$$w_t = f_t \times \left(\frac{\text{DF}_{C_C}(t)}{N_{C_C}}\right) \times \log\left(\frac{N_{C_P}}{\text{DF}_{C_P}(t)}\right) \quad (12)$$

where $DF_{C_C}(t)$ is the document frequency of the term in the child cluster, C_P refers to the parent of cluster C, $DF_{C_P}(t)$ is the document frequency of the term in the parent cluster, N_{C_C} is the number of pages in the child cluster (including the candidate page), and N_{C_P} is the number of pages in the parent cluster.

If the cluster has no parent, the last term would result in a zero error, so the following truncated formula is used:

$$w_t = f_t \times \left(\frac{DF_{C_C}(t)}{N_{C_C}}\right) \quad (13)$$

Note that this formula is nearly the opposite of the TFIDF formula. In TFIDF, a term is considered less important if it occurs in more documents of a group, because the intent is to find terms that uniquely describe the page. In this case, terms that are common

to more pages better describe the cluster, and are thus more important to the cluster. Those terms that are common to the cluster which are shared with the parent, however, are not as important to distinguish this cluster, because, by definition of being a subcluster of the parent, all of the parent's terms are already found throughout the cluster; in other words, no new information is learned about a cluster by those words it inherits from its parent.

4.3.3 Placement

The placement of new pages is directly based on the ability to compare two pages discussed above.

To determine whether a document D should be part of a cluster C, we can calculate the similarity between the document in the context of the cluster (D_C) and the cluster's centroid, $\overline{D}_C$:

$$sim(D_C, C) = sim(D_C, \overline{D}_C) \tag{14}$$

The best cluster $\widehat{C}$ for a given document D is therefore the cluster that maximizes the similarity:

$$\widehat{C} = \underset{C \in \mathbb{C}}{\operatorname{argmax}}\, sim(D_C, C) \tag{15}$$

Formally:

1. Given a set of existing clusters $\mathbb{C}$ and a new document D, for each cluster C_i recalculate the term weights of document D, producing a candidate document D_{C_i}.

2. For each candidate document D_{C_i}, calculate the similarity of the candidate to its cluster C_i:

$$score(D_{C_i}) = sim(D_{C_i}, \overline{D}_{C_i})$$

3. A clear winning cluster $\widehat{C}$ is discovered by finding a cluster for which the document score is better than every other cluster, and for which the score is greater than a given minimal similarity threshold (ϵ=0.6) :

$$C_i, \widehat{C} \in \mathbb{C}, \forall C_i \neq \widehat{C}, \quad \text{score}(D_C) > \text{score}(D_{C_i}), \text{score}(D_C) > \epsilon \tag{16}$$

4. If there is a single winning cluster $\widehat{C}$, the page is given to that cluster.

5. If there is a tie for the winning cluster or there is no clear winner, the page is held back in the "general area" until such time as it can be awarded to a clear winner.

Hierarchical Placement

When hierarchical clusters are involved, a distant descendant of a cluster may be the winning cluster. The simple placement algorithm can be extended by searching through all of the children for the best possible match. This can be accomplished by changing the score function of the cluster to return the best score between the cluster and its children:

$$\text{score}(D_C) = \max(\text{sim}(D_C, \overline{D}_C), \max_{\forall C' \in C_C} \text{score}(D_{C'})) \tag{17}$$

4.3.4 Redistribution

As subclusters can change periodically, pages should be moved from a more general cluster to a more specific one whenever possible. The process of redistribution is the same as placement; the only difference being that the winning cluster must not only surpass all other clusters and the minimal similarity threshold, but must also be a better fit than the current cluster containing the page.

4.3.5 Splitting

Since pages are added incrementally and are held back if there is no clear winner, it is possible that a number of pages in a given cluster actually form a subcluster. The clustering program (in this case, an agent) must periodically examine the cluster it represents in order to discover these subclusters.

In order to find these subclusters, the following definition is used: given a set of documents $\mathbb{D}$, a subcluster is defined as a group of documents within that set for which the following holds:

1. the documents that best match/have the best similarity to a given document D' is denoted as the set $B \subseteq \mathbb{D}$;

2. all best match sets for each document within a given subcluster C are also contained within the subcluster;

3. all best match similarity scores are above a minimum similarity score.

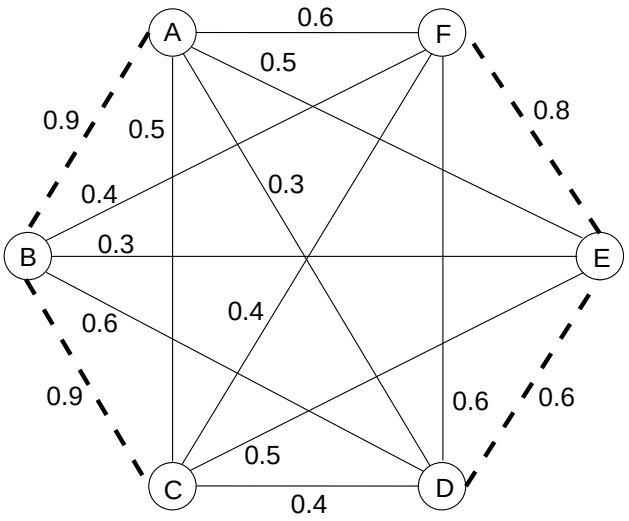

Figure 10: A graph representation of the largest disjoint set algorithm.

If the set of documents $\mathbb{D}$ is viewed as a weighted graph, such that each vertex represents a document within that set, and each edge is weighted with the similarity score between the documents on its vertices, then the clusters are those distinct, non-overlapping subgraphs.

Given the pages $\{A, B, C, D, E, F\}$ in the example in Figure 10, two subsets, $\{A, B, C\}$ and $\{D, E, F\}$ can be found.

Formally:

1. For all documents $D_i, D_j \in \mathbb{D}$, $D_i \neq D_j$, calculate the similarity score to be placed in the half-filled matrix M. That is, each entry $M[i][j]$ contains $\text{sim}(D_i, D_j)$.
2. Initiate a set of clusters $\mathbb{C}$.
3. For each document D_i:
   (a) If $\exists C \in \mathbb{C}$ such that $D_i \in C$, skip to the next document.
   (b) Otherwise, find the set of best matching pages, B; i.e. those pages that have the best similarity scores with the given document. Together, this forms a potential cluster C'.
   (c) If any document in B is contained in any cluster of $\mathbb{C}$, this cluster is invalid.
   (d) For each document D' in B, repeat from step 3(b) until no new pages are discovered.
4. If the cluster is not of a sufficient size, the cluster is invalid.
5. If the cluster has not been invalidated, add the new cluster to the set of clusters.
6. Repeat from step 3.

The code shown in Listing 1 illustrates this processing.

4.4 Agents

Clusters by themselves do not entirely make up the SWAMI user profile; they are data representations, a reflection of the user's interests, but inactive. To act on behalf of the user requires something more; it requires an actor, in this case, an agent, to take up the cause of that interest and perform its duties of search and evaluation.

For SWAMI, a simple, custom agent solution has been created. This section describes the stages of development of these agents and their particular roles and responsibilities.

4.4.1 Processing Overview

The agents within the multiagent system representing a user are organized into a hierarchy of responsibilities, with the most general of roles being held by the agents at the top and the most active of roles being held by agents near the bottom (see Figure 11).

At the top of the structure is the browser itself, sometimes referred to as the *web browsing user agent*. This is really not an agent in the strict sense, but is really just the wrapper program. It is normally just responsible for answering user requests and displaying the resulting pages, but in SWAMI it has the additional responsibilities of forwarding the browsing information to the system, displaying recommendations received by the system, and allowing the user to choose from those recommendations for places to browse. The prototype implementation of SWAMI includes a basic Java browser with the SWAMI system integrated into it; in a more general implementation, this would be accomplished by a combination of a normal third-party web browser and either a plugin, web proxy or similar device.

Below the browser comes the real beginning of the SWAMI system, the interface agent. This agent is responsible for coordinating between the various internal agents and the browser, including creating new

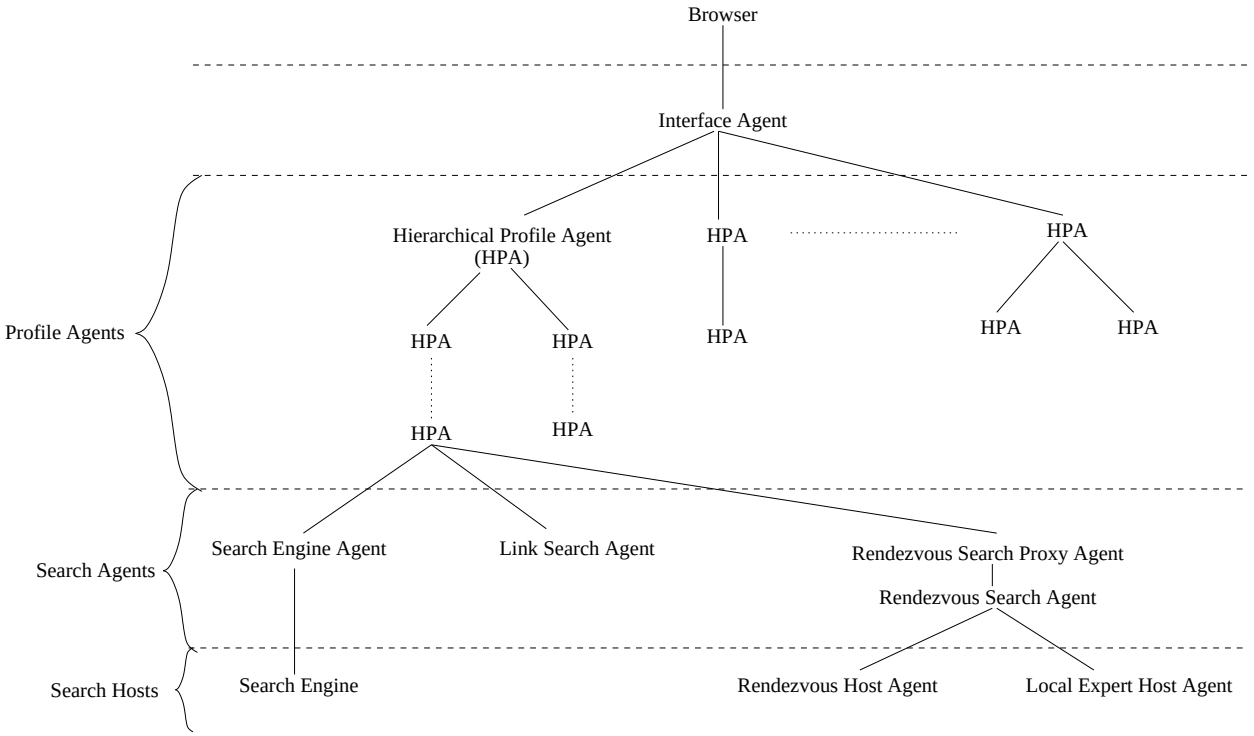

Figure 11: The hierarchy of agent interactions.

top-level agents to handle newly discovered topics, requesting bids on new pages the user has browsed, and keeping general track of recommendations.

Next on the hierarchy are the true representatives of the user's interests, the hierarchical profile agents. Each tree of agents is referred to as a *corporation* of agents, working together to provide representation and support to a particular high-level topic.

Below the representative agents in the system are the search agents. These agents are specialized to particular search methods, and are created to search and evaluate a particular topic as directed by a profile agent.

Finally, the last class of agents are the hosts to visiting search agents. There are two kinds, varying only in where they get their information for trade: local expert host agents and rendezvous host agents.

The following sections describe these agents in more detail. Rather than the system order in which they have been presented here, the agents are described in order of complexity, in keeping with the object-oriented design methodology used. In this way, agent knowledge, responsibilities and actions are described incrementally, so that the relationship might be more easily understood.

4.4.2 The Basic Agent

The basic agent is primitive and unmotivated; it has very little capacity for knowledge, both external and internal, and has very little activity on its own. The two basic things it knows about itself are its age and its wealth.

The age of an agent is a difficult concept to measure. Since activities in the agent world happen at a vastly different speed than the real world that we inhabit, and since the activity modelled here (web browsing) is not a continuous activity but rather is marked by intense, effectively random periods of activity with unpredictable intervals of time between, real world time measures are insufficient and altogether non-indicative of an agent's real age.

Age is important, however, as one of the goals of SWAMI is to allow behaviour over time to change the model (e.g. the interests of a user shift or change with time). Thus, a measure of age relative to the central focus of the agent's activity is used: the number of significant events since the agent's creation. In this

case, the significant events of an agent's lifetime are the web pages browsed by the user.

Since an agent may also be called to account for its current status, given its history of activity and the knowledge it has, it is created with the capability of calculating its own instantaneous wealth. The wealth calculation must take into account several factors, including activity over time, success, strength and depth of knowledge, and recency of success. In the case of a basic agent, however, the calculation is relatively simple:

$$wealth_{simple}(t) = \gamma \times wealth_{simple}(t-1) \qquad (18)$$

Where $wealth_{simple}(0) = \omega$, some starting wealth, and γ is an age retention factor, allowing an agent to rest on its laurels from previous success.

Since an agent should be capable of notifying those agents it is communicating with of any important events, it has a mechanism whereby agents who wish to be informed of significant changes can register themselves with the agent.

Finally, to give an agent the ability of some independence, it is capable of running as its own execution thread. However, for a basic agent that has no motivations of its own, this capability is unused; it becomes significant with more sophisticated agents.

4.4.3 The Basic Agent Environment

An agent operates within an environment. In the SWAMI system, the `AgentEnvironment` class provides this. This environment tracks all of the agents within the system, and provides a way for an agent to discover and communicate with another agent in the system. In the basic environment, this is done directly by the id of the agent.

The basic SWAMI agent environment also holds general knowledge that all agents should be aware of, in particular the stoplist, which is used to eliminate uninteresting words when evaluating a web page, is retrieved from this basic environment.

The agent environment also operates independently within a thread. Housekeeping operations can be performed in this thread, although the basic environment has none.

4.4.4 The Profile Agent

This agent is capable of representing a single cluster of web pages and arranging for searches to be performed. It is capable of evaluating a given page for inclusion within its cluster and providing a numerical score of the page. This score is a similarity measure based on the similarity between the page and the pages already contained within the cluster. As it accepts pages, the `ProfileAgent` recalculates its centroid and notes the best, worst and average similarity of the pages it contains.

This agent also keeps track of those links already searched and any pending recommendations that searches have returned. To prevent searches from happening too often, a "grace period" has been introduced. This grace period is actually measured both in terms of age and real time. In addition, in order to start a search the agent must be of sufficient size, have minimum average similarity, have no pending searches, and have no recent recommendations pending or have had excellent success with their recommendations. If all these conditions are met, the profile agent can create a search agent to do a search.

Life-cycle

A profile agent goes through the following steps in its life-cycle:

1. **Birth**: When a profile agent is first created, it is being created to take over the control of one or more pages. From this initial page dump it calculates its features to be used for page comparisons. The wealth of an agent after it has been initially created is the minimum wealth, 0.0.

2. **Page bidding.** Each time a user views a page, a profile agent is called to bid on that page. If the page is already owned by the agent or the page matches one of the agent's recommendations, the value of the bid is 1.0 (perfect match). Otherwise, the value of a bid is simply the comparison between the page in the context of the agent's pages and the agent's features; in this case, the agent's age increases by one, and its wealth is recalculated.

3. **Page acceptance.** If a profile agent is the winner of a bid for a page, it will be given the page to own. If the page is one of the agent's recommendations, that recommendation is marked as "accepted". When a page is added, the agent is marked as increasingly "dirty". When the dirtiness surpasses a threshold δ (currently in the system, this is 2), the agent's features are recalculated to match the possibly shifting centroid of its cluster.

4. **Searching.** After a page has been accepted and the wealth recalculated, the agent may initiate a search if its wealth exceeds the threshold. The profile agent creates a new search agent of the appropriate type and starts that search agent in a separate thread. When that thread completes, the search agent will notify the profile agent that it has finished searching, and the profile agent will retrieve the list of recommendations from the search agent (if there are any) and destroy the search agent.

5. **Recommendation gathering.** At any time, the recommendations found for a particular profile agent may be requested. If the agent has not yet searched or is in the middle of searching, it may not have any recommendations.

6. **Retirement.** After an agent's wealth has been recalculated, the agent may be deemed no longer relevant to the profile if its wealth falls below a particular threshold, η. If this is discovered, the agent is removed from the profile and placed in a special location called the holding area (see Section 4.4.8 for a more detailed description of the holding area).

   Note that there is a "grace period" of 5 age units during which time an agent is not considered for retirement. This allows an agent to establish itself.

   This stage allows interests that are no longer significant to fade away.

7. **Reinstatement from reserve.** If an agent that has been retired is successful on a new page bid, it may be returned to active service by the interface agent. In this way, interests that recur are represented.

8. **Death.** If an agent falls below a lower threshold τ of wealth, the agent is removed from the system.

4.4.5 Wealth calculation

Finally, a profile agent uses the wealth formula described in Equation 2 to account for its changed knowledge and responsibility.

In this formula, α is the relative importance of the agent's size, β is the relative importance of the search success and ϕ is the relative importance of the success in having recommendations followed.

Useful values of each of these factors was discovered experimentally. A value of 0.25 was initially used, and was modified as the relative rates of change of each of the components were observed. The goal in this was to achieve an acceptable balance between the sustaining of active agents and the decay of inactive agents, while remaining sensitive enough to allow changes in the system to happen in a reasonable time frame. That is, the functioning of the system for each of the main points (agents increasing and decreasing in importance, agents splitting, agents retiring, agents rehiring and agents being removed) could be observed within a short number of page views.

After several iterations, the values were assigned as follows: $\alpha = 0.50$, $\beta = 0.25$, $\phi = 0.24$ and $\gamma = 0.01$. While these values were useful for demonstrating the system, in the future use of the system it is expected that they will have to be less sensitive.

Note that all of the factors add up to 1.0, and each of the component values has a range from between 0.0 and 1.0; this yields a weight between 0.0 and 1.0.

The agent's *sizeActivity* is a time-diminishing measure of the size of the agent balanced with how recently the agent was updated.

First, a few definitions: the *size* of an agent is calculated as the number of pages the agent holds:

$$\text{size}(A) = |\text{pages}(A)| \tag{19}$$

The *age* of an agent is the number of pages it has bid on since its birth. The *recency* of activity is the difference between the current age and the age when a page was last added:

$$\text{recency}(A) = \text{age}(A) - \max_{\forall p \in \text{pages}(A)} \text{addAge}(p) \tag{20}$$

The general size activity equation is:

$$\text{sizeActivity}(A) = \frac{f1 + f2}{2} \tag{21}$$

where $f1$ and $f2$ are defined in Equations 22 and 23:

$$f1 = \begin{cases} 1.0 & \text{size}(A) > \text{age(A)} \\ \frac{\text{size}(A)}{\text{age}(A)} & \text{otherwise} \end{cases} \tag{22}$$

$$f2 = \begin{cases} 1.0 & \text{size}(A) = \text{recency}(A) = 0 \\ 0.0 & \text{recency}(A) \geq \text{size}(A) \\ 1.0 - \frac{\text{recency}(A)}{\text{size}(A)} & \text{otherwise} \end{cases} \tag{23}$$

Two other special conditions apply: when the age of the agent is 0, the *sizeActivity* is 0, and when the

size of the agent is zero but the recency is non-zero, the *sizeActivity* is 0.

The search success is a measure of how good recent searches have been, calculated as the proportion of good searches to all searches:

$$\text{search}(A) = \frac{\sum_{\forall r \in \mathbb{R}_{\text{good}}} \left(\frac{\text{score}(r)}{\text{age}(A) - \text{birthAge}(r) + 2} \right)}{\sum_{\forall r \in \mathbb{R}} \left(\frac{\text{score}(r)}{\text{age}(A) - \text{birthAge}(r) + 2} \right)} \tag{24}$$

where $\mathbb{R}$ is the set of all pending recommendations, r is an individual recommendation, birthAge(r) is the age of the agent when the recommendation was created, and $\mathbb{R}_{\text{good}}$ is the the set of all good searches:

$$\mathbb{R}_{\text{good}} \subseteq \mathbb{R} \| \forall \text{r} \in \mathbb{R}_{\text{good}}, \text{score(r)} \geq \rho$$

The threshold ρ is arbitrarily set to 0.5 in the prototype.

Acceptance is a measure of how many recommendations were followed by the user, calculated as the proportion of recommendations followed relative to the number of recommendations made:

$$\text{acceptance}(A) = \frac{|\mathbb{R}_A|}{|\mathbb{R}|} \tag{25}$$

where $\mathbb{R}_{\mathbb{A}} \subseteq \mathbb{R}$, and all recommendations in $\mathbb{R}_{\mathbb{A}}$ were followed by the user at some point .

Each of these formulae was created to arbitrarily represent elements believed to be of importance to an agent. They are all designed to diminish with a lack of activity over time.

4.4.6 The Hierarchical Profile Agent

The hierarchical profile agent extends the profile agent by adding the knowledge and ability to find subclusters, modifying the procedure used to evaluate a page for ownership, and the ability to distribute pages to children that have changed to more accurately represent them. This agent is a node in a tree, having at most one parent agent and zero or more children agents.

When a hierarchical profile agent is requested to evaluate a page, it also requests that each of its children agents also evaluate the page. The best evaluation between itself and those its children present is then returned to the request. In this way, a page will move down the tree to the most specialized agent that it resembles.

Similarly, when recommendations are requested from such an agent, all the recommendations from the agent's children are gathered and returned as well.

Each time a page is received by a hierarchical profile agent, it is "dirtied". When it becomes sufficiently dirty, it updates its representative centroid. This can cause some pages to be better represented by a child agent, so this agent will attempt to distribute its pages among its children.

This kind of agent will also look for subclusters within its pages in order to create child agents. Obviously, such a subcluster cannot contain all the pages of the parent, because then the child is nothing more than a copy of the parent. Also, a subcluster check is not made after each page is added, to avoid the *cascading subcluster problem*. This problem is caused when a single page is added to a subcluster that, while more similar to the subcluster than the supercluster, is less similar to the other members than they are to each other. Thus, the single page is added, and if a subcluster were to be looked for immediately, the pages that had previously made up the subcluster would be found, and a new subcluster, identical to the original, would be found. This leads to a large number of agents in a single branch which each have a single page and a subcluster of the same, with a tight subcluster moving further away.

(a) Life-cycle

The life-cycle of a hierarchical profile agent differs from a regular profile agent in a few ways:

1. **Birth.** Same as a profile agent.
2. **Page bidding.** Bids from all children agents $\mathbb{C}_C$ are collected and returned.
3. **Page acceptance.** Pages may not be accepted directly by an agent, but must be delivered to the appropriate winner of the bid.
4. **Page distribution.** If a page has been accepted by a child agent, that agent may have recalculated its features and pages owned by the parent agent might more appropriately fit with the child agent. Thus, the parent agent asks all of its child agents for a comparison score. Note that this is different from a bid, in that it does not increase the age of any agents. If there is a clear winner for a page, it is redistributed down the branch that won to the appropriate child agent.

This allows a limited reconfiguration of the hierarchy, so that pages flow downward to the most specific agent for the page topics.

5. **Wealth recalculation.**
   If pages have been accepted or redistributed there will possibly be changes in the wealth of agents along the way.
6. **Subcluster search, or "splitting".** At the core of the hierarchical structure is the ability of a cluster to contain one or more subclusters. An agent will search through its pages to attempt to find one or more maximal subclusters. For each of these subclusters a new child agent is created.
7. **Searching.** Same as for the simple profile agent, the hierarchical agent may schedule a search to be performed.
8. **Recommendation gathering.** When the recommendations are requested of a hierarchical profile agent, it in turn requests all the recommendations from all of its children and collects them all in a list. If more than one agent has a recommendation for a particular page, the best recommendation is used.
9. **Retirement.** When a hierarchical profile agent is retired, all of the children agents become children of the agent's parent agent. If the agent has no parent, i.e., it is the head of a corporation, all of the child agents become heads of new corporations under the interface agent.
   While retired, a hierarchical profile agent may not create any new children.
10. **Reinstatement from reserve.** With hierarchical profile agents, new agents may be added beneath existing agents (as opposed to the simple profile agents, where new agents can only be created as new corporations). Whenever a hierarchical profile agent discovers that it has need of a new agent, it may request from the interface agent that an agent be retrieved from the holding area and used instead of creating a new agent from scratch. This allows the older agent a new lease on life, and the parent benefits from the existing knowledge of the older agent.
11. **Death.** Since the only way an agent can face death is to be placed in the holding area and an agent in the holding area cannot have any children, death comes under the same circumstances and conditions as for a simple profile agent.

b Term Weight Context

This agent requires modifications to the function used to calculate the term weights of a page. Two things need to change, based on the addition of child agents: the determination of the document frequency of a term, and the calculation of the size of the agent. For the purposes of document frequency, a child agent contains the term if one of its features contains the term. A hierarchical profile agent's size is calculated based on the sum of all of its own pages and the number of children agents it has. In effect, the presence of a child agent is treated much like the presence of another page within the agent.

c Wealth Calculation

The only change to the wealth calculation is a change in the way that an agent's size is determined. An agent's size is the total number of pages and the total number of child agents combined:

$$size(A) = |pages(A)| + \sum_{\forall c \in children(A)} size(c) \tag{26}$$

4.4.7 Search Agents

Search agents are responsible for finding pages that might be of interest to the user and evaluating them. They are created as needed for a specific search by a profile agent, and do not carry any history. The specific information used to search may be specialized by the type of search agent, but generally, they act upon a vector of features and collect recommendations.

Multiple types of search agents are used in the system:

1. **Google search agents** are a type of search-engine leveraging search agent, and contain the Google API to submit queries and evaluate their results on behalf of the user.

2. **Link search agents** use the proximity-priority search algorithm described in the previous chapter.

3. **Expert consultation agents** travel to the environment of a set of local expert agents representing a website.

4. **Rendezvous search agents** travel to community environments (rendezvous servers) to interact with other search agents and host agents for recommendations.

In this initial implementation, the type of search agent used is fixed for the particular compilation. That is, a user using a particularly compiled version of the system will be using one and only one type of search agent. (It is planned in the future that the choice of search agent to use can be influenced both by the user's preferences and by the observations of the situation and performance of the search agent types by members of the agent corporations; in this way, the system will be able to adapt the use of search agents to the material searched as well as observed successful searching.)

(a) Search Hosts

The search-engine-leveraging search agent uses an external search engine host to accomplish its search, but otherwise has no special requirements of the server.

The link search agent only relies upon the web servers that contain the pages for its search process. It does not require any special search host features.

The expert consultation agent and the rendezvous search agent travel to the remote search host in order to interact with agents there locally. This requires a specialized host that implements a SWAMI agent environment. This environment is capable of receiving agents from remote environments and providing them with a way to navigate once they have arrived. In both cases, the actual search mechanisms and environments are effectively identical, with the real difference coming in the kinds of host agents presented.

The expert consultation agent travels to remote SWAMI environment to consult with experts. This environment contains a number of local SWAMI expert agents that each contain some hidden knowledge and set of pages over which they are considered experts. Upon arriving in the environment, the expert consultation agent requests a list of expert agents that are similar to itself, and seeks out each agent mentioned on the list until it has reached its search limitations. The local expert agents give recommendations to the visiting agents, but only change their knowledge based on internal discoveries; that is, they do not change based from their interactions with visiting agents. The search limitations include the number of pages considered, the number of agents spoken with and the number of (good) recommendations discovered.

The rendezvous search requires a Rendezvous Environment. Similar to the expert environment, this environment is capable of delivering a visiting search agent a list of locally hosted agents that are similar to it that the search agent can consult. There are two important differences that make this environment different, however: first, the list of agents returned contains not only agents that are locally created, but also contains any other search agents that happen to be visiting at the time; second, if the environment detects that no internal rendezvous host agents were among the list of agents recommended to be consulted with, it will create such an agent, allowing the environment to grow and reflect those things searched within it. The locally-created rendezvous host agents trade recommendations with visiting agents, and thus are changed by those agents that visit.

In both cases, the score of a recommendation is modified by how similar the two agents are.

(b) Search Proxy Agents

In the case of both of the expert consultation and rendezvous agents, a local controlling agent is needed to handle the creation of the connection to the remote environment and the agent sending and retrieving. This agent serves as a controlling proxy for the communication. This agent is created by a profile agent to create a search, and controls the search agent in turn; see Figure 12 for a illustration of the proxy's role.

4.4.8 The Interface Agent

The top-most agent involved in the system is the `InterfaceAgent`. It is a specialized `HierarchicalProfileAgent` which observes the pages viewed by the user, distributes them to the appropriate agent corporation, and collects recommendations from all corporations for the user. It cannot

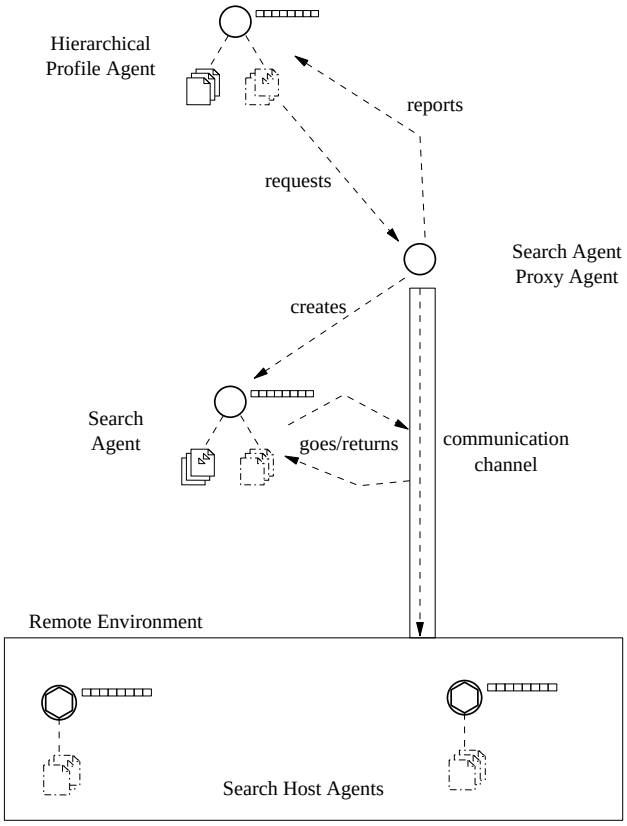

Figure 12: The role of the search proxy is to coordinate search agents' travel to remote search environments.

perform searches, cannot be retired or deleted, and cannot be placed in the holding area.

The interface agent accepts *all* pages, unlike other hierarchical profile agents; like other hierarchical profile agents, it will attempt to discover subclusters to create new corporations to handle them.

The Holding Area

The interface agent is responsible for the special area in which agents that have been retired are placed. An agent is not directly deleted from the system when it ceases to be immediately relevant. Instead, the agent will be removed from its current place and put into the *holding area* when its wealth falls below a *retirement threshold*, η. This allows agents that performed well to exist longer, in case the topics they represent are repeated in the future.

While in the holding area, an agent may not create any child agents nor create any search agents. It continues to make bids on incoming pages and therefore, to age. If it should win a bid from an incoming page, it will be reinstated from retirement and added the head of a new corporation. This reflects the situation of a recurring interest, where the user had drifted away from an interest for a period of time, but has now returned to it.

If the agent fails to win any bids, its wealth will gradually decrease over time. Once its wealth has fallen below a second threshold, τ, it is removed permanently from the system.

4.4.9 The Browser

The whole system is put together in the browser (see Figure 13). This browser has basic features of a typical browser:

- it has a URL input field, so that a user may enter a URL and jump immediately to any page;
- it has history, and the user can navigate backward/forward through history;
- a link on any displayed page can be followed by clicking on it.

In addition, several specialized features have been added:

- there is a table of current page recommendations for the user to select from, if desired
- the current analysis of any page can be displayed
- each of the agents working for the user can be inspected using the agent browser
- the state of all agents can be saved and loaded, allowing the user to take a snapshot of the system
- a message window at the bottom tells the user the current activity, including which agent won the bid for the current page, if a new agent has been created, retired or removed, and when a search has been conducted. (This was primarily used for monitoring activity during testing.)

4.5 Summary

The SWAMI system contains a custom agent implementation written in Java. Many of the agents map to clusters in an incremental clustering scheme, and each agent is capable of independent action.

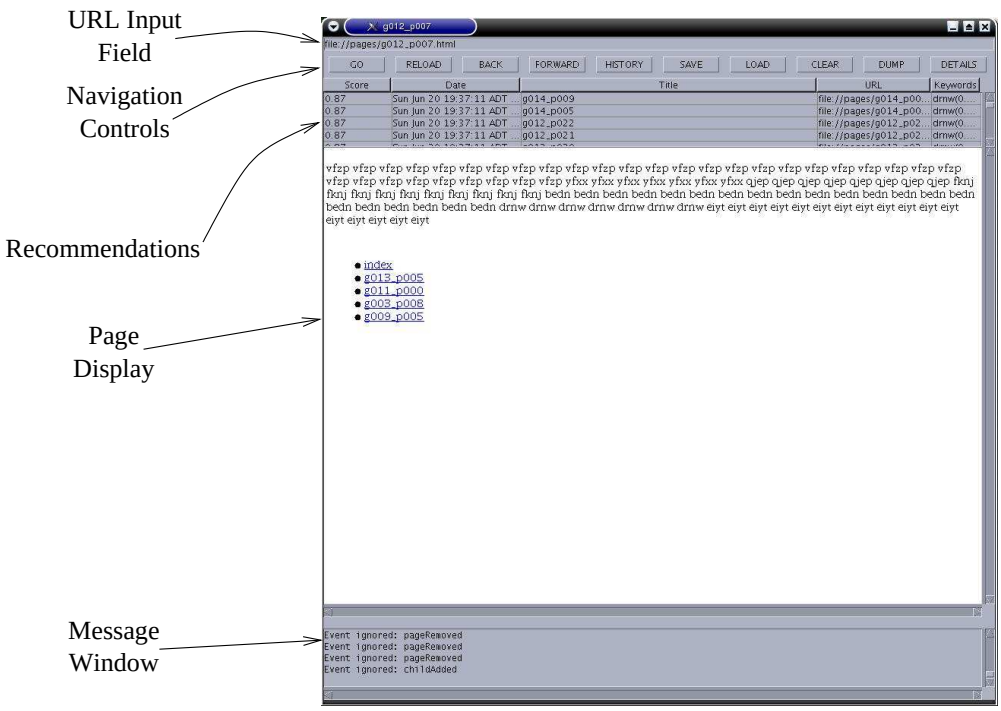

Figure 13: A screenshot of the browser with integrated SWAMI.

A number of parameters control the agent's life cycle, including the balance of importance between agent size, agent searching, agent recommendation success and momentum. Other parameters include the retirement threshold and the death threshold of wealth. Parameters governing the clustering include the minimum number of pages an agent must see in order to search for clusters (the minimum dirtiness threshold).

5 Evaluation

The evaluation of the SWAMI system consists of verifying that it can detect the growth and shift of user interests, provide a usable model of the user, and act upon that model on behalf of the user.

To perform this evaluation, test data was generated that represents a web of pages that are interconnected and that have localized coherence. From this test data, numerous trial runs were conducted in order to demonstrate that all of the key events expected of the system were observed, and that the system was behaving as expected. Four typical example runs are highlighted here.

Because the test data was generated offline and never made available to a search engine, no examination of the search-engine-based search method could be attempted, without implementing a specialized search engine, which was beyond the scope of this initial research.

5.1 Test Methodology

The SWAMI system was tested by the author manually by selecting a random start page and following links according to one of five typical user motivations, derived from the earlier observations about browsing behaviour:

1. **Continued interest.** Links might be chosen to represent a continued interest in an existing topic. The links followed for this motivation were links to pages within groups from which other pages had been visited in this session.

2. **Changing interest.** Links might be chosen to represent a break in the interests of the user. These links were chosen at random from the page being viewed or their URLs were entered directly into the input area.

3. **Returning interest.** Links might be chosen that represent a return to a past interest. Often, these links were chosen to be relevant to an agent that had been retired but not yet deleted from the system, to demonstrate that such an agent would be returned to active service.

4. **Intense and specialized interest.** Links might be chosen to represent a deepening and possibly specialized interest within an existing topic. These links were chosen from the best recommendations provided by the system.

5. **Shallow interest.** Links might be chosen to represent the seeking behaviour of a user, wherein links are essentially chosen at random.

Test runs were continued until it had demonstrated a particular feature. In total, 25 test runs were complete.

For each test run, the following events were observed:

- when an agent is created;
- when an agent splits; this is distinct from the previous event in that it concentrates on the effect of the change on the parent, rather than the details of the new child agent;
- the bids for each page from each agent in the system;
- the wealth of each agent after each bid
- when a search was conducted, and the results it found;
- when an agent is retired;
- when an agent is reinstated;
- when an agent is deleted permanently;

To summarize the results, two charts were devised: the page group activity chart and the agent activity chart. The time scale on both of these charts is in terms of page views by the user.

The page group activity chart shows the activity of the user in terms of the pre-generated page group to which each page the user chose belonged. Note that pages within the same pre-generated page group share a high similarity, while pages between groups tend to have a much lower similarity, so a steady line indicates a consistent interest that the system should recognize.

The agent activity chart shows the wealth of all of the agents in the system over the same period of time. As agents accumulate pages and become more sure about the topics they represent, their wealth should increase; as the topics they represent fall out of favour, the wealth should decrease. The identification of subtopics can also be seen in the creation of child agents, which is shown as a sudden start of a weight track on the chart. When agents are removed, their weight track disappears.

These two charts, when taken together, show the inputs and outputs of the system, and the better they correspond, the better the system is tracking the user and acting on their behalf.

5.2 Results

The following sections describe the results from two particular trial runs in detail. These trial runs were chosen to clearly illustrate the performance of the system, but are otherwise typical.

It should be noted that the system uses a particular naming scheme for all agents within the system. All agents begin with the prefix "Charlie" and have a suffix indicating their parentage. In the case of the topmost agent, the Interface Agent, no suffix is used. The first child of the Interface Agent is called "Charlie_0"; the second child of this agent would be called "Charlie_0_1", and so on.

To describe the life-cycle of an agent, a chart showing the agents' wealth over time is used. This chart is calibrated in absolute terms, meaning that while an individual's age is calculated relative to when they were born, it has been adjusted to the appropriate real outside age relative to the age of the Interface Agent. The age also describes the number of unique pages viewed. Where a line begins on the graph indicates when an agent was born; if the line ends prematurely, that agent was removed from the system.

The lower threshold for an active agent's wealth before being retired is 0.2; only the Interface Agent cannot be retired. If their wealth continues to drop, an agent will be removed when it falls below 0.15.

Pages within a particular group are known to be similar to each other, and thus represent a topic. This is used both to train the system and to interpret its results.

Also, as the agents search, they discover pages in other page groups that are relevant to the topic, thus forming a virtual topic group based on the user's demonstrated interests.

5.3 Example 1: Interest Shifts

This example demonstrates SWAMI's ability to follow a user's changing interests and react accordingly. The page groups that the user visited can be seen in Figure 14. On the weight track in Figure 15, five agents (in addition to the Interface Agent) are shown.

Each agent was created when the system detected a cluster of similar pages. The set of pages initially chosen all came from pre-generated group 40, followed by a number of pages selected from group 38. Charlie_0 was created when the subset of pages from group 40 were detected as distinct, at age 4. At age 11, a second agent (Charlie_1) was created to take control of the second subcluster discovered (for group 38). Note that while the pages were chosen from the pre-generated group, the system itself has no knowledge of these groups.

Between ages 14-36, links were followed semi-randomly from existing pages, but not corresponding to any previous page. These pages were similar enough to existing agents that Charlie_0 rose in wealth during this time period, and Charlie_1 maintained a high wealth. Concentration by the user on a single pre-generated page group again from age 36-43 resulted in the creation of a new agent, Charlie_2, to handle a newly-discovered cluster formed out of those pages. Another agent, Charlie_3 was created at the same time, as the new pages highlighted some previous cluster in the previous pages.

Between ages 53 and 77, recommendations made by Charlie_0 were followed, resulting in that agent's consistent wealth, while other agents diminished. At age 77, a new topic was focused on, and a new agent, Charlie_4 was created in response.

Note that when the user concentrated on a particular topic, the system responded by creating a new agent to handle this new topic when it detected it. As the user drifted away from that topic (by not visiting again), the agents that had been responsible for it waned in wealth.

The longevity of both Charlie_0 and Charlie_1 indicate long-term interests. Charlie_0, in particular, has received a lot of attention from having suggestions followed.

Charlie_2 and Charlie_3 accurately map to short-term interests. In the case of Charlie_3, no recommended pages from that agent were viewed, leading it to degrade in wealth very quickly and disappear within about 5 page views. Charlie_2 was a short-term interest which the user paid a little attention to.

Finally, Charlie_4 is a new interest to which the user is paying attention and good recommendations have been found. The system responds quickly to the newly discovered cluster, and it becomes the most influential among them.

This example has shown that the system creates new agents to handle new user interests, and the wealth of those agents reflects the ongoing interest in the topic they represent.

5.4 Example 2: Interest Specialization

In stark contrast to the previous example, this example demonstrates the creation of specialized agents for sub-topics discovered within the context of a larger topic. While the page group activity shown in Figure 16 seems to be chaotic (particularly after age 57), the corresponding location on Figure 17 shows relatively stable behaviour.

Charlie_0 represents a long-term interest (page group 42) which was concentrated on for a considerable period of time. Two sub-topics were detected from within this one, represented by Charlie_0_0 and Charlie_0_1. The second of these was pursued momentary, but was forgotten for a period of time. Note that Charlie_0_1 was retired but brought back instantaneously when the user returned to that topic. At that point, it actually triggered a split, creating the very short term topic represented by Charlie_0_1_0.

At approximately age 45, the system has detected that the user has decided to view another topic intensely for which good suggestions could be found. This is represented by Charlie_1, whose continued strength is due to its suggestions being followed. The return of a peak in Charlie_0 at approximately age 72 was due to following a link on a page suggested by Charlie_1 which led off to an older topic.

The relative stability of the wealth track of Charlie_1 after age 45 despite the apparent randomness of the page group activity for the same time period is due to the agent having found pages within multiple groups which are similar to the topic at hand. In this way, it has created a virtual group of pages centred around the user's interests.

6 Conclusions

This paper describes a framework for a multi-agent system for providing personalized web page recommendations to users. The SWAMI framework fea-

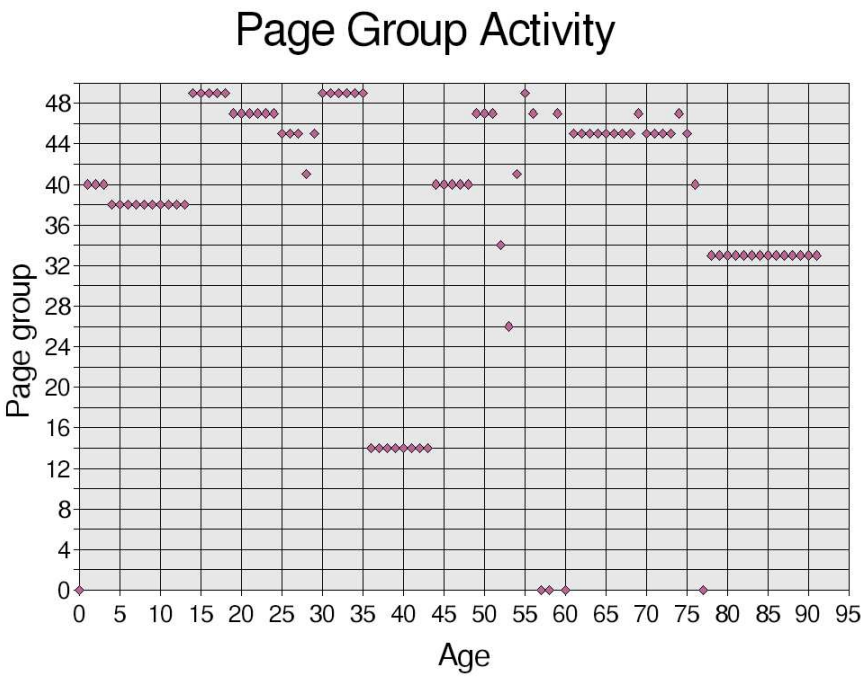

Figure 14: The page groups visited by the user on the first example test run.

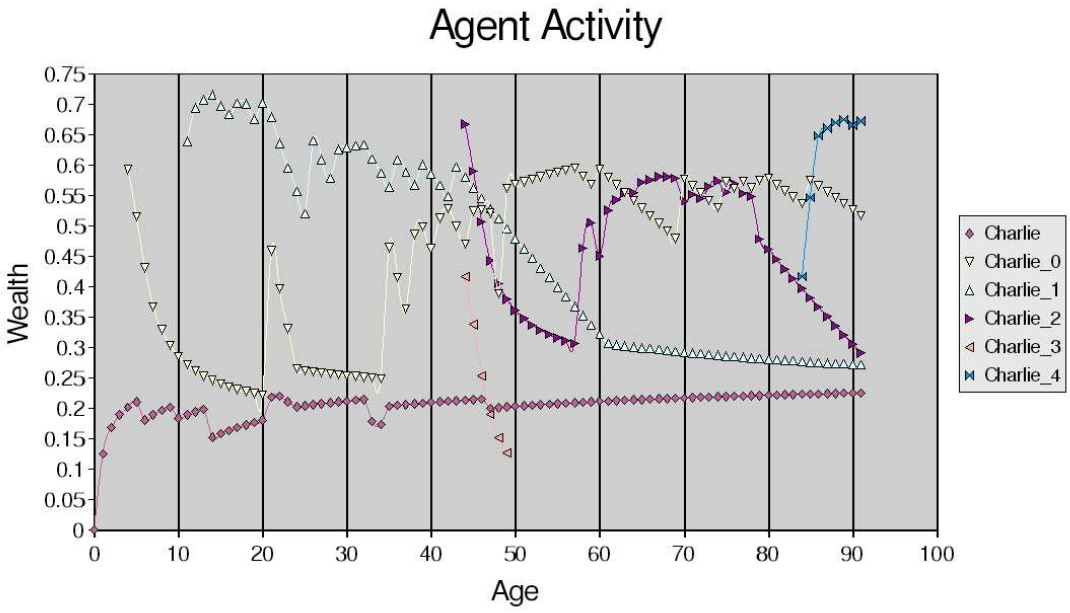

Figure 15: Agent activity from the first example test run.

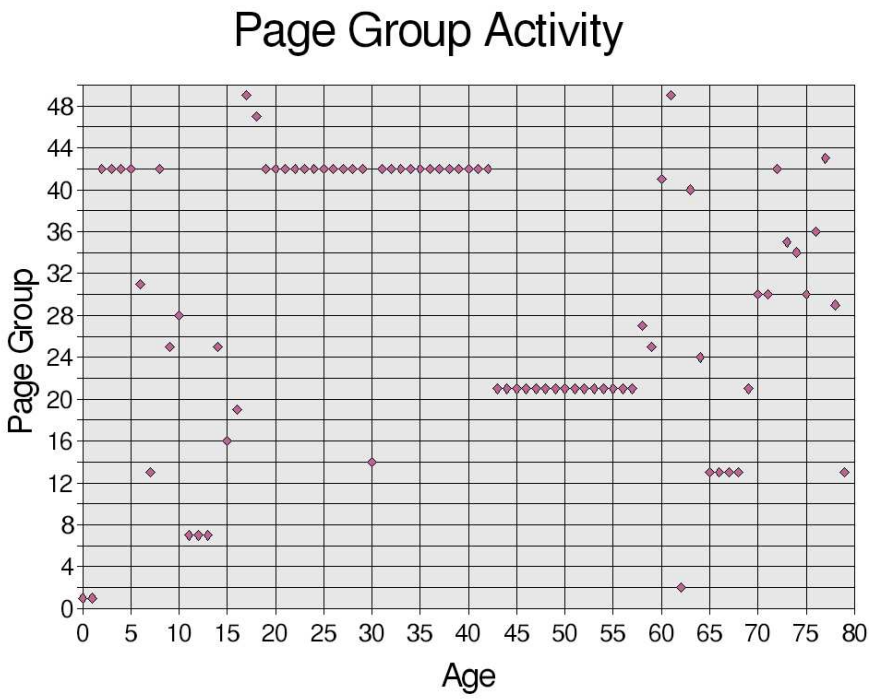

Figure 16: The page groups visited by the user on the second example test run.

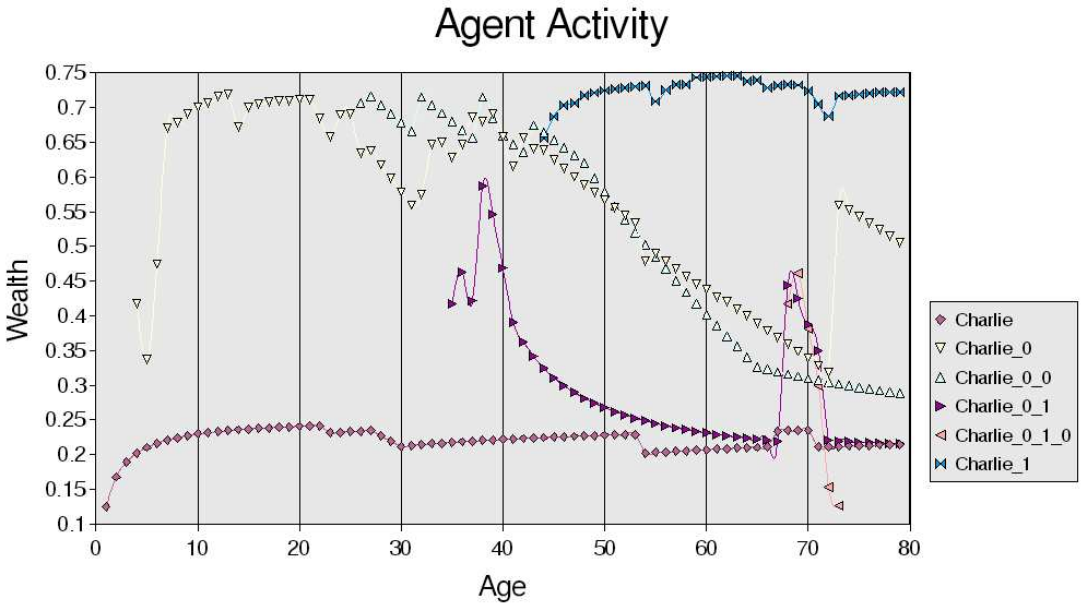

Figure 17: Results from the second example test run.

tures a sophisticated user model using a social multi-agent system with a cost-driven and time-variable interaction model organized into hierarchies of related topics. Agents representing particular topical interests in this system can search for recommendations for the user with one of multiple strategies. Among those search strategies is the ability of the search agents to become mobile. Mobile search agents can travel to particular, SWAMI-aware websites and interact with local topic experts, or they can travel to SWAMI "rendezvous servers", where they can interact with user-independent collaborative recommendation agents and with other search agents representing users.

Key features of this framework include local representation of a user's interests (allowing the system to "learn once, apply everywhere"), the integration of local, site-based and collaborative recommendations, and an active user profile representation which takes into account short-term, long-term and recurring interests, as well as the specialization of interests.

This holistic approach to web search represents a more realistic solution to the problem of web search than site-specific or user-agnostic approaches.

Several trial runs were performed, from which typical examples were chosen to examine in detail. These trial runs demonstrate that the agents do grow to mirror the user activities and change over time to reflect changes in the user intentions. Short-term, long-term and recurring interests have been detected by the system, as well as specialization to accommodate a particularly important interest. Recommendations could be gathered successfully by using a link-search algorithm, by consulting with site experts or through interacting with a community. Recommendations in the community were successfully distributed between members of that community.

6.1 Future Work

This research has proved promising, but there are several additional questions raised throughout the work that would make interesting follow-up research. These questions include:

- **Parameter setting.** There are a large number of parameters within this system, including: minimum similarity thresholds; minimum agent wealth before retirement; minimum agent wealth before deletion; the relative weights of each component in the agent wealth calculation; the minimum number of pages required before an agent considers splitting; the minimum number of pages that must remain in a cluster after all others have been allocated to sub-clusters; and so on. These parameters are currently established through observation and arbitrary decision. However, in some cases, these parameters do not always work correctly. It seems natural that these parameters might either be adjusted by evidence or even determined entirely by evidence within the system. In addition, it may also be beneficial to allow parameters to be tuned according to user tastes.

- **Agent hierarchy/structure reorganization.** As agents are retired and rehired, the hierarchical structure of the agents is reorganized, allowing multiple independent but similar specializations the possibility of converging in one branch. This specialization will only occur, however, when interest in a particular sub-agent has waned significantly. There seems to be a natural role for a "headhunter" agent or something similar which can help facilitate reorganizations of the agent structure without the need of a diminished interest.

- **Open agent structure.** The agents are currently arranged in a strictly hierarchical manner, with each agent having at most one parent and any number of children. This structure, while convenient, is artificial; information often does not follow a strictly hierarchical structure, instead having more of an open graph structure. One possible modification of the SWAMI architecture is to modify the concept of "parent" and "child" agents into the more general "ancestor" and "descendant" roles, or even into the most general "sibling" role. Such a structural change would be capable of modelling much more subtle interactions within the data, but each agent would have a larger web of information from which to discover patterns.

- **Page comparison.** Several page comparison mechanisms were examined before selecting the cosine similarity measure. In particular, several variations on the Jaccard measure were strong contenders. Only a few of the measures examined take into account structural or positional information about the term on a page. Alternative methods of page comparison might improve the identification of page clusters and the ability for pages to reach the appropriate agent. Pages

are also currently viewed only as the collection of terms that physically occur within the body of the document (in their stemmed forms). The system might be significantly improved if a facility such as WordNet could be included to search for other words based on common relationships such as antonym or synonym, although at a cost of complication and processing time.

- **Incremental calculation.** When a page is added to a new cluster, it affects the centroid of that cluster, which may shift the set of features enough that some of the pages that had been part of the cluster may no longer fit properly within it, or might fit more appropriately within a sub-cluster. Thus, every time a page is added to a cluster, a significant amount of recalculation is potentially performed, significantly impacting performance. If it could be possible to calculate only the effect itself on the cluster instead of calculating everything, or to calculate a predictor that can indicate whether a full recalculation should be done, this would significantly improve the speed of the system.

- **Local experts agents.** Only one form of the local expert agent was examined in the process of this thesis, but several are possible. In particular, it might be possible for the local experts to reorganize themselves to reflect the kind of requests that are being made, making the local experts adaptive to usage.

- **Rendezvous agents environment.** The rendezvous server implemented here is basically functional, but it does not have any sophisticated mechanisms for creating new rendezvous host agents. Currently, if no host agents are discovered to serve a particular incoming search agent, a new host agent is created. Also, there is little attempt to guarantee that two host agents do not end up covering the same material; in fact, in the evaluation of the rendezvous server, there were in fact two host agents that had a difference of only two pages.

- **Remote search environment locating.** The current system looks for a remote search environment (whether rendezvous server or local expert environment) on the local machine, at a known port. To make the system more generally useful, a mechanism for identifying these remote environments is necessary.

- **Choosing search methods.** In the current system, the method of searching is hard-wired into the particular incarnation of the executable. It is desirable that multiple search methods be available simultaneously to the system. It is also desirable to allow the agents to choose which method is appropriate, perhaps under the direction or suggestion of the user. This includes preferences, perhaps, for different remote search environments for different topics. It is expected that the search methods would complement each other well: local expert agents allow specialized exploration in a particular local environment; link search agents allow easy exploration between local page environments; rendezvous servers link individuals together into communities, allow transfer of recommendations between users; and the search-engine-leveraging search agents allow disconnected local environments to be connected.

- **User control and manipulation of agents.** It has been speculated that if users were able to tag or name the agents working for them, they would be able to further judge the recommendations provided by those agents. Other controls might also be useful, such as the ability to freeze an agent from changing (thus always providing the same kind of judgement without fear of being retired), arbitrarily remove an agent (to prune the system), or arbitrarily reward an agent for a particularly useful suggestion.

- **Page re-occurrence.** The system currently takes a simplified view of pages: they are unchanging entities, so when a page has been viewed once, it need never be viewed again. The interface tracks the list of pages that have been viewed so far this session, and simply does not process those pages that have already been seen. This simplification works for a large number of pages, but a significant portion of the web changes constantly. If these changing pages could be identified, the system could automatically scan the pages to see if they have changed enough to be revisited by the user.

- **User profile persistence.** Currently, the system is only designed to work for the duration of a single session. The ability to save the state of the system was experimented with early in the development, but it was vulnerable to incompatibilities introduced by code changes. For such a

system to be generally useful, however, it must have a way to store a user's profile between sessions. In a similar way, the rendezvous server should have the ability to persistently store the community it represents.

- **Wealth as accumulated value.** The system calculates the wealth of an agent as an instantaneous value based on the performance and other history of the agent. An alternative view is to treat wealth more like the real economic concept, in that it is something acquired for success and paid out to perform actions or to simply "live". The instantaneous system was implemented to give some measure of control and confidence that the system would have a continuing downward trend if no beneficial activity took place. A more open economy also requires additional controls to ensure that it changes appropriately and maintains a good balance.

- **Earlier detection of groups.** To circumvent the "cascading group" problem, the system requires that a small number of pages be left behind in a parent agent before a child agent can be created. In particular, the interface agent, which is not capable of searching itself, will not create any corporations if there are not a sufficient number of pages that remain after the corporation has been created. This leads to the system apparently not able to find a group until after the user has left it, because only by visiting pages that are inconsistent with the previous pages can a new group be formed (leaving those most recent pages behind).

- **Real user testing.** This system was tested with a model of real users over pages which have known properties. The next stage of testing will involve real users and a much broader set of pages with more variable qualities, such as with the general Web. Preliminary testing in this manner has yielded promising results. In more open testing, users would be able to rate the search results discovered and provide a qualitative score to both determine the real success of the search agents and to provide feedback for the system to choose which avenues of search are most fruitful.

References

[Asnicar and Tasso, 1997] Asnicar, F. A. and Tasso, C. (1997). ifWeb: a prototype of user model-based intelligent agent for document filtering and navigation in the World Wide Web. In *Proceedings of the Workshop on Adaptive Systems and User Modeling on the World Wide Web, Sixth International Conference on User Modeling*.

[Balabanovic and Shoham, 1997] Balabanovic, M. and Shoham, Y. (1997). Combining content-based and collaborative recommendation. *Communications of the ACM*, 40(3).

[Brusilovsky, 1996] Brusilovsky, P. (1996). Methods and techniques of adaptive hypermedia. *User Modelling and User-Adapted Interaction*, 6(2-3):87–129.

[Chan, 1999] Chan, P. (1999). Constructing web user profiles: A non-invasive learning approach. In *KDD-99 Workshop on Web Usage Analysis and User Profiling*, pages 7–12, San Diego, CA, USA.

[Chen and Chen, 2002] Chen, C. C. and Chen, M. C. (2002). PVA: A self-adaptive personal view agent. *Journal of Intelligent Information Systems*, 18(2/3):173–194.

[Cosley et al., 2002] Cosley, D., Lawrence, S., and Pennock, D. M. (2002). REFEREE: An open framework for practical testing of recommender systems using researchindex. In *28th International Conference on Very Large Databases, VLDB 2002*, Hong Kong.

[De Bra and Ruiter, 2001] De Bra, P. and Ruiter, J. P. (2001). AHA! adaptive hypermedia for all. In *Proceedings of the WebNet Conference*, pages 262–268.

[Fink et al., 1997] Fink, J., Kobsa, A., and Nill, A. (1997). Adaptable and adaptive information access for all users, including the disabled and the elderly. In Jameson, A., Paris, C., and Tasso, C., editors, *User Modeling: Proceedings of the Sixth International conference, UM97*, pages 171–173, Vienna, New York. Springer Wien New York.

[Freitag et al., 1995] Freitag, D., Joachims, T., and Mitchell, T. (1995). WebWatcher: A learning apprentice for the World Wide Web. *Working*

Notes of the AAAI Spring Symposium: Information Gathering form Heterogeneous, Distributed Environments, pages 6–12.

[Godoy and Amandi, 2000] Godoy, D. and Amandi, A. (2000). PersonalSearcher: An intelligent agent for searching web pages. In *Advances in Artificial Intelligence, IBERAMIA-SBIA 2000*, volume 1952 of *Lecture Notes in Artificial Intelligence*, pages 43–52. Springer.

[Godoy and Amandi, 2002] Godoy, D. and Amandi, A. (2002). A user profiling architecture for textual-based agents. In *Proceedings of the 4th Argentine Symposium on Artificial Intelligence (ASAI 2002) in the 31st International Conference on Computer Science and Operational Research (JAIIO 2002)*, Santa Fe, Argentina.

[Google, 2004] Google (2004). Google search engine. `http://www.google.com`.

[Java Stemmer, 2005] Java Stemmer (2005). Porter stemmer in java. `http://www.tartarus.org/~martin/PorterStemmer`.

[Joachims et al., 1997] Joachims, T., Freitag, D., and Mitchell, T. (1997). WebWatcher: A tour guide for the World Wide Web. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, pages 770–775. Morgan Kaufmann.

[Kleinberg, 1999] Kleinberg, J. M. (1999). Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5):604–632.

[Kobsa et al., 2001] Kobsa, A., Koenemann, J., and Pohl, W. (2001). Personalized hypermedia presentation techniques for improving online customer relationships. *The Knowledge Engineering Review*, 16(2):111–155.

[Lieberman, 1995] Lieberman, H. (1995). Letizia: An agent that assists web browsing. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 924–929, San Mateo, CA, USA. Morgan Kaufmann Publishers Inc.

[Lieberman et al., 1999] Lieberman, H., Van Dyke, N. W., and Vivacqua, A. S. (1999). Let's browse: a collaborative web browsing agent. In *Proceedings of the 1999 International Conference on Intelligent User Interfaces (IUI'99)*, pages 65–68, Los Angeles, CA, USA. ACM Press.

[Mladenic, 1996] Mladenic, D. (1996). Personal WebWatcher: design and implementation. Technical report, Department of Intelligent Systems, J. Stefan Institute, Slovenia.

[Mobasher et al., 2002] Mobasher, B., Dai, H., and Tao, M. (2002). Discovery and evaluation of aggregate usage profiles for web personalization. *Data Mining and Knowledge Discovery*, 6:61–82.

[Pazzani and Billsus, 1997] Pazzani, M. J. and Billsus, D. (1997). Learning and revising user profiles: The identification of interesting web sites. *Machine Learning*, 27(3):313–331.

[Porter, 1980] Porter, M. (1980). An algorithm for suffix stripping. *Program*, 14(3):130–137.

[Schwab et al., 2000] Schwab, I., Pohl, W., and Koychev, I. (2000). Learning to recommend from positive evidence. In *Proceedings of the 2000 International Conference on Intelligent User Interfaces*, pages 241–248, New Orleans, LA, USA.

[Wooldridge, 2001] Wooldridge, M. (2001). *An Introduction To Multiagent Systems*. John Wiley & Sons, Ltd, England.

[YAHOO!, 2004] YAHOO! (2004). Yahoo! search engine. `http://www.yahoo.com`.

Listing 1: Main cluster discovery method

```
public Vector cluster( double[][] sim ) {
    if ( sim == null || sim.length == 0
            || sim.length != sim[0].length ) {
        return null;
    }

    Vector clusters = new Vector();

    boolean allOk = true;

    boolean[] used = new boolean[sim.length];
    for( int i = 0; i < used.length; used[i++] = false );

    boolean[] done = new boolean[sim.length];
    for( int i = 0; i < done.length; done[i++] = false );

    for( int q = 0; q < sim.length; q++ ) {
        if ( !done[q] ) {
            Queue finished = new Queue();
            Queue inProgress = new Queue();

            inProgress.push(q);

            allOk = true;

            while( allOk && inProgress.size() > 0 ) {
                int p = inProgress.pop();

                Vector best = findBest( p, sim );

                if ( best.size() > 0 ) {
                    finished.push( p );
                    used[p] = true;

                    Queue leftovers = new Queue();
                    leftovers.push( best );
                    leftovers.pull( finished );
                    leftovers.pull( inProgress );

                    for( Iterator match = leftovers.iterator();
                         match.hasNext(); ) {
                        int other = ((Integer)match.next()).intValue();

                        if ( used[other] ) {
                            allOk = false;
                        }
                        else {
                            inProgress.push( other );
                        }
                    }
                }
                else {
                    allOk = false;
                }
            }

            if ( allOk && finished.size() >= minClusterSize ) {
                Vector cluster = new Vector();
                cluster.addAll( finished );
                clusters.add( cluster );
                for( Iterator page = finished.iterator();
                     page.hasNext(); ) {
                    int p = ((Integer)page.next()).intValue();
                    done[p] = true;
                }
            }
            else {
                done[q] = true;
            }
        }
    }

    return clusters;
}
```