Non-intrusive Lightweight Agents for Information Management

S. Varma D.B. Skillicorn

November 1997
External Technical Report
ISSN-0836-0227-
97-411

Department of Computing and Information Science
Queen's University
Kingston, Ontario, Canada K7L 3N6

Document prepared November 5, 1997

Abstract

An information management system should reduce the workload of the user. It should also learn about user's preferences and reduce the effort in managing information. We propose a non-intrusive approach to building information management systems, based on software agents. Software agents are programs that carry out actions on behalf of the user autonomously. We describe the issues involved in designing such systems. As an example, we design BMA (a Bookmark Management Agent). BMA is a non-intrusive, lightweight software agent that manages bookmarks (web browser pointers to favourite sites).

Tests show that BMA is a useful tool in managing bookmarks. This shows that software agents are a promising approach to information management systems. It also demonstrates the feasibility of a non-intrusive agent.

1 Introduction

1.1 Motivation

The Internet is growing [23] and the sea of information brought to our doorstep is expanding every day. It is estimated that more than 70 million users will be using the World Wide Web by the end of 1997 [4]. This leads to the issue of how well the user is prepared to take advantage of this information boom. Furthermore, new technologies have yielded faster computers and bigger applications on the desktop. Each application has a number of powerful features. These features continue to grow in number causing increased interface complexity.

Users would appreciate any help in dealing with this information overload. However, the question is how and where to offer this help. To address this question one needs to identify the areas where a user would require assistance. Moreover, it should be provided in such a way that it does not increase the complexity of her desktop environment and is as *non-intrusive* as possible.

The idea of a *software agent* has recently gained popularity due to its potential capacity to reduce the user's workload. The main idea behind software agents is that there are lots of tasks users have to do that can be delegated to the computer. An agent is a software program that runs autonomously and reduces the work load for the user by doing certain chores. Also, it tries to learn the preferences of the user and customises itself accordingly.

The World Wide Web is becoming very popular as indicated by the number of people using it [4]. However, the sheer amount of information makes things difficult. Search engines try to manage this information by indexing documents on the web. However, there are few tools that operate at the user's end.

Software agents can be used to reduce the work load for the user by managing her information. However, the idea of employing an agent has a potential risk of the agent trying to intrude into the user's work space. Many people don't like the idea of an agent continuously watching over their shoulders. Moreover, they don't trust the agent enough to hand over the controls.

The aim of this research was to take a non-intrusive approach to designing information management systems using the concept of software agents. We believe that such systems will play an important role in managing information in the future.

In this report, we address as a non-trivial application domain for this idea, the problems in the management of information already collected from the World Wide Web. This information usually presents itself as bookmarks,

pointers to users' favourite sites. The identified problems in bookmark management are as follows. The World Wide Web is dynamic in nature and this leads to several issues. These include

- **Change in bookmarked sites/pages:** Due to immense size of the WWW, users cannot easily be notified of any changes in sites. For example, if the user wants to check if a new edition of a magazine has been published on the web she has to keep going to the site to check that.

- **Movement of sites:** Web pages sometimes move from one server to another. Again, a user cannot easily be informed that a site has moved. The user, for example might go to her favourite site and not find it at all. She then has to go through the whole process of searching for it.

- **Removal of Web pages:** Pages are removed from web servers without notification and this sometimes leaves the user unprepared. For example, a student might have bookmarked several research papers and decides to go to them before an important presentation only to find the key paper missing.

The bookmarks list is one of the most frequently used menu items of a Web browser. Some of the problems with the bookmarks are:

- As the number of bookmarks increase it gets difficult to scan the bookmark file. For example, a user might have to go several folders in depth to reach a bookmark.

- There are several bookmarks that the user no longer uses and has forgotten. Since the user no longer requires them, they only make things difficult by crowding the bookmark file.

- A user might visit a favourite site on a particular day. For example, the bi-weekly magazine gets released on the 1st and 15th of every month. However, a user may not always remember to go to these sites on that day and miss seeing her favourite site.

The following sections discuss the various issues involved and demonstrate the construction of a software agent using the non-intrusive approach. Our implementation of the agent is called the Bookmark Management Agent (BMA) and it addresses the issues discussed earlier.

1.2 Contributions

The work presented in this report uses techniques from the field of software agents to solve the problem of information management. The contributions made by this report are as follows.

- This report proposes a non-intrusive approach to the construction of information management systems using the concept of software agents. It also addresses the various issues involved in taking this approach.
- It identifies a problem in information management and demonstrates how it can be addressed using techniques from the field of software agents.
- It demonstrates the design and construction of Bookmark Management Agent (BMA) that can be used to address the issue of managing bookmarks.
- It also validates that *software agents* are a promising approach to designing non-intrusive information management systems.

1.3 Outline of the Report

The rest of this document is divided into the following sections. Section 2 describes what agents are and gives an overview of the type of agents that are being researched today. It also discusses the Internet, the World Wide Web and narrows down to browsers and bookmarks to provide a setting for the Bookmark Management Agent (BMA). Section 3 describes the non-intrusive approach for incorporating the various properties of a software agent. It then discusses the resulting design and architecture of BMA. Section 4 discusses the technical issues encountered during the construction of the BMA. Section 5 discusses and demonstrates the working of the Bookmark Management Agent. Section 6 gives the concluding remarks and discusses the observations made during the course of this work.

2 Background

The intent of this chapter is to give a broad overview of *software agents*. We first explain what an agent is and review a few definitions. We also describe properties of an agent. Then we provide a panoramic overview of different agent types that are being developed and researched. Furthermore, we discuss

the techniques being explored by various research groups to enhance the functionality of user agents. At the end we discuss how knowledge about user's behaviour can be extracted from a *bookmark file*.

We begin by describing the concept of an *agent* and its interpretations.

2.1 Agents

The idea of an agent has attracted attention from many fields such as psychology, sociology, and artificial intelligence. Research in this field has spawned many names and definitions. For example, agents are also known as intelligent interfaces, adaptive interfaces, knowbots, softbots, userbots, personal agents and network agents. However, no agreement has been reached on an exact definition of an agent. Most of the definitions [9] are based on the techniques used to build an agent and on the functionality they have to offer.
For example, according to the IBM [11] definition,

> *Intelligent agents are software entities that carry out some set of operations on behalf of a user or another program with some degree of independence or autonomy, and in so doing, employ some knowledge or representation of the user's goals or desires.*

The above definition sheds some light on the goals being pursued by major companies like IBM [11, 6] and Microsoft. Essentially, work is being done to improve existing applications and make them more user-friendly.
Maes [16] defines them as:

> *Autonomous agents are computational systems that inhabit some complex dynamic environment, sense and act autonomously in this environment, and by doing so realize a set of goals or tasks for which they are designed.*

Maes has done much work [16, 18, 17] in the field of agents that employ AI techniques to perform tasks for the user. The agents usually learn from users' behaviour and aim to automate repetitive tasks.
Another definition given by researchers at MIT, Stanford and AT&T [3] is:

> *Agents assist users in a range of daily, mundane activities, such as setting up meetings, sending out papers, locating information in multiple databases, tracking the whereabouts of people, and so on.*

The above definitions highlight the views held by various researchers both in commercial and educational institutions.

2.2 Agent Characteristics

An agent possesses certain properties which differentiate it from a user-driven program. In fact, the notion of an agent is being pursued [17] to break away from this paradigm. Of course, an agent is a program, but the way it operates changes.

According to Wooldridge et al. [28] an intelligent agent should exhibit the following properties.

- Autonomy : Agents operate without the direct intervention of humans, and have some kind of control over their actions and internal state.
- Social ability: Agents interact with other agents (and possibly humans) via some kind of agent-communication language.
- Reactivity: Agents perceive their environment, (which may be the physical world, a user via a graphical user interface, a collection of other agents, the Internet, or perhaps all of these combined), and respond in a timely fashion to changes that occur in it.
- Pro-activeness: Agents do not simply act in response to their environment, they are able to exhibit goal-directed behaviour by taking the initiative.

Ideally, an agent should strive for all of the above attributes. However, it is clear that the essence of an agent is to act without direct manipulation on the part of the user, and to be useful to the user.

2.3 Agent Classification

There are essentially six types of agents being developed. Nwana [22] gives a good overview of the typology of agents. The following typology tries to cover the more specific categories mentioned in [2, 12].

2.3.1 Collaborative Agents

Goals of collaborative agents are similar to goals of *distributed artificial intelligence*. The aim is to exploit distributed resources and thus make an agent more useful. Collaborative agents cooperate with other agents autonomously to carry out tasks for the user. They interact with other agents in a multi-agent environment and reach a mutually acceptable decision. Collaborative

agents can help solve problems too large for a centralized single agent and they can form an effective interface between heterogeneous systems. Other benefits include enhanced modularity, higher speed due to parallelism, reliability, and flexibility and reusability of existing resources.

An example of such a system [22] is the Pleiades project at Carnegie-Mellon. In this system agents cooperate with each other to carry out a task specified by the user. The agents communicate using Knowledge Query and Manipulation Language (KQML) [10] and negotiate to reach agreements.

2.3.2 Interface Agents

The idea of an interface agent is being promoted by the artificial intelligence community. Interface agents are autonomous agents that learn to perform tasks for the user. An agent monitors and observes the actions taken by the user and suggests better ways of carrying them out. The agent can also offer to automate repetitive tasks for the user. An interface agent learns by observing and imitating the user, either receiving negative and positive feedback from the user, receiving explicit instructions from the user, or by communicating with other agents. According to Maes [16, 18, 17] the goal is to move away from the paradigm of a user-initiated interaction and to delegate some tasks to interface agents. There are several functional examples of interface agents. These include

- Maxims (Electronic Mail Agent) [17]: This agent helps the user with email. It continuously looks over the shoulder of the user as she deals with email. It learns to set priorities, delete, forward, sort and archive mail messages for the user.

- NewT (News Filtering Agent) [24]: It helps the user filter netnews. The user either gives the agent instructions as to what are the topics she is looking for, or the agent tries to keeps track of users' preferences. The agent learns from the feedback given by the user and also by observing the user's behaviour. However, detecting a pattern can take a considerable amount of time.

- Web Watcher [1]: This tour guide software agent helps the user in browsing the web. It guides the user along an appropriate path through the collection, based on the knowledge of the user's interests, of the location and relevance of various items in the collection , and of the way in which the other users have interacted with the collection in the past.

2.3.3 Mobile Agents

Mobile agents can traverse networks to carry out tasks specified by the user. A mobile agent can gather information or perform tasks by interacting with the various hosts in the network. Mobile agents help reduce communication costs, make better use of distributed resources, and carry out tasks in parallel. There are presently two major platforms compatible with mobile agents.

1. Telescript Development Environment (TDE) [27]: This operating environment was invented by General Magic (Mountain View, CA). It consists of a telescript engine which allows multiple agents to execute and share information among themselves. These agents are developed using the telescript language. A mobile agent is packed with all its information by the host engine and sent off to another engine across a wide area network. The receiving host then unpacks the agent, lets it execute, and sends it back to its originating destination. These agents can carry out interprocess communication to use services offered by other agents.

2. IBM's Aglet Workbench [26] : This environment has been created by IBM Japan and uses Java to write mobile agents known as aglets. Java's platform independence, security features, and ability to execute on any host, made it the language of choice for developing mobile agents. The Aglet Workbench provides a set of APIs for developing aglets. Aglets have the ability to maintain their state while traveling across a network. They can carry out operations on one machine, and then move to another one to continue the same operation without starting from scratch.

2.3.4 Information Agents

Information agents address the issue of information management and overload. These agents help the user in information search and retrieval. They typically issue a request to various search engines and extract the information most relevant to the user's request.

The Internet Softbot [8] is an example of such a system. It accepts a high-level request from the user and then is able to carry out the request by invoking appropriate services provided by the Internet. The agent uses an automatic planning algorithm. After getting the request from the user, the agent generates a sequence of steps based on its knowledge of available information resources, databases, utilities, and software programs. The agent is intelligent in the sense that the user or the programmer does not have to

specify the steps it has to take to attain a goal. Other information agents like Web Compass [5] and Surfbot [25] have to be initiated by the user and they go out and look for the information specified. Thus, they are not totally autonomous.

2.3.5 Reactive Agents

This is a relatively specialized class of agents. Reactive agents do not possess any symbolic model of their environment and react to the present state of the environment. This makes the agent adaptable, flexible, and more tolerant to changes in the environment. An example [16] of such a system is the ALIVE (Artificial Life Interactive Video Environment) project developed at MIT. In this system a user is allowed to enter a virtual world where she interacts with agents. The agents are in the form of animated characters. They react to the situation presented to them. For example, an animated character in the ALIVE system moves in the direction pointed by the user. Another such example is the Julia Chatterbot [20] project at Carnegie Mellon University. The project involves developing a *chatterbot* which can communicate using human expressions.

2.3.6 Hybrid Agents

All types of agents have their strengths and weaknesses and a hybrid agent tries to combine two or more agent philosophies to come up with a better combination.

For example, if an information agent and mobile agent philosophies are combined, we could have an agent that effectively helps the user look for specific information. The agent can travel the network looking for the specified information and get back to the user after accomplishing its task.

2.3.7 Web Access

The World Wide Web provides the opportunity to access resources all over the world. However, the number of resources is so huge that it is not easy for an untrained user to enjoy their benefits without the use of a search engine. Search engines (e.g. WebCrawler, Yahoo, Lycos etc.) [14] play an important role in locating the information available on the network. A search engine does this in the following way. It either retrieves the documents from the various web servers or relies on users providing them. It then processes documents

to extract titles, keywords etc. This information is then stored in a database. The user can now submit a query to this database and retrieve a list of relevant documents. Search engines are sometimes referred to as internet agents as they autonomously look for documents and index them for the users.

However, the increase in the number of engines has lead to problems like network congestion. To keep their indexes up to date these engines traverse the web looking for HTML documents. Since the number of engines is large, there is an immense load on the network.

Recently, there has also been an explosion in the number of Web Wanderers/Robots [15]. Robots are programs that traverse the Web collecting pages that might be of some interest to the user.

2.4 Information Management

As more information sources are added to the World Wide Web, there is a growing need for managing this information. Unfortunately, there are few software tools available to help. Search engines keep track of information by indexing documents, but the number of documents available is increasing rapidly, as is the network traffic generated by the process of capture and index. For example, the Lycos engine [19] which went public in July 1994 with an index of 54,000 documents had 60 million documents by November 1996.

The dynamic nature of the Internet makes it impossible to keep the index information up to date. Documents are added, moved and deleted every day, and it is hard to keep up.

Services like search engines address information management globally, but they are not responsive to the needs of users individually. There is an evident need for software tools which operate and manage information from the user perspective. There are some signs of this – for example, some sites allow users to receive customised information. However, economic pressures tend to favour 'push' technologies over the present user-controlled framework, making tools for finer control by users less attractive commercially.

2.5 The User

It is estimated that around 70 million users will be making use of the WWW by the end of 1997 and this number continues to grow rapidly [4]. The internet is no longer limited to researchers exchanging technical information. Users can be divided into two broad categories.

- Trained : The trained users are familiar with the Internet and the tools available to access the services provided by it. They make efficient use of the various facilities provided by the Internet.

- Untrained: Users who have just discovered the world of the Internet and make casual use of facilities like email, or just browse the Web for entertainment purposes. The number of such users is increasing at a high rate due to the introduction of services like online magazines, banking, shopping etc.

2.6 The User, the Agent and the World Wide Web

The basic reason for agent research [17] is to help all types of users take maximum advantage of the information that is available. However, the main hurdle [3] is to identify the issues that require agent help and to solve them effectively. A lot of valuable information is available on the Internet and users spend a lot of time trying to find relevant information. Many agents are focussing on addressing the issues arising due to this information overload.

The main fields that are being investigated are information filtering, information searching and information management. Information filtering agents like Newt [24] filter the information available through Usenet newsgroups according to users' preferences. Agents like Web Compass [5] and Sulla [7] search for information on the World Wide Web that might be of interest to each user. However, the user still has to choose relevant articles from the total number of articles retrieved by the agent. Information management agents include Maxims [17] that manages email: sorting messages into priority order, and deleting and archiving them. Since the idea of agents for the World Wide Web is new there are very few agents that address these issues.

To create an agent that is of any assistance to the user, one has to gain knowledge about the user. There are two ways one can do this

1. By observing the user's behaviour.

2. By asking the user what she wants.

One would have a more effective agent if both the above ways are incorporated into the agent. Let us consider the sequence of steps a user follows if she wants to view a document, or search for a document on the World Wide Web.

- The user starts a web browser.

- The user accesses the document by typing in a URL (address of the document) or uses a bookmark (address stored by the browser).
- If she is looking for a new document, she types in keywords in the Search Engine interface and the engine gives her the documents that match the query.
- If the user finds a document she is looking for, she bookmarks it for future access.

From the above sequence of steps one can deduce that the bookmarks are an important source for identifying user's preferences. Our survey of the various agents available on the Internet revealed that only two agents, namely URLMinder [21] and Surfbot [25], deal with bookmarks. However, they both require the user to register bookmarks with the agent. Surfbot helps by downloading the bookmarked sites on a schedule specified by the user, and URLMinder sends the user email if the registered site changes.

2.6.1 Bookmarks

Every time a user adds a bookmark, or goes to a bookmarked site, the browser makes an entry into the bookmark file. For Netscape, the bookmark file is an HTML file and stores the following information.

1. The date the bookmark was added.
2. The date the bookmark was last visited.
3. The date the bookmark was modified.
4. The URL for the bookmark.

2.6.2 Knowledge Extraction

To extract knowledge about the user's behaviour one can observe the user and establish a pattern in her habits. Such an approach is followed by agents like NewT and Maxims [24, 18]. Agents like URLMinder [21] and Surfbot [25] ask the user to register the sites that they want observed. A third option is to extract this knowledge from information that exists already. The bookmark file is a good source for extracting knowledge about a user's behaviour. Since it stores the addresses of all the sites the user visits, one can get a good idea about user habits.

3 Design

This section first discusses the attributes that are expected of agents in general. We then discuss a non-intrusive approach to incorporating these attributes. We also introduce some additional properties that an agent should posses in order to be non-intrusive. The various choices that arise while taking a non-intrusive approach are then described. Later, we discuss the requirements and architecture of an example system.

3.1 Attributes

As mentioned in Section 2, certain attributes are characteristic of an agent. The following section discusses the properties [9, 28, 12] that are desirable in an agent.

- **Autonomy:** The agent should operate without the direct intervention of humans and should have some kind of control over its actions. The user should be able to delegate certain tasks to the agent and it should carry them out without the user's cooperation.
- **Pro-activeness:** Agents do not simply act in response to their environment; they are able to exhibit goal-directed behaviour. The agent should start spontaneously and work towards its goals. The aim should be to develop software that functions without direct manipulation on the part of the user.
- **Pattern Recognition:** The agent should be able to detect patterns in the user's actions. Usually, the patterns are indicative of repetitive behaviour. It should either try to automate repetitive actions or assist the user in performing the tasks.
- **Interaction/Social ability:** The user should be able to communicate her preferences to the agent. The agent, on the other hand, should have the ability to communicate with its user. The communicated information could be suggestions relating to a task or messages notifying the user of accomplished tasks.

3.2 The Non-intrusive Approach

We believe that agents will play an important role in managing information in the future. However, if a number of agents inhabit the user's environment

there is potential risk of agents getting too intrusive. This might lead users to consider them as a nuisance instead of assistants. We propose a non-intrusive approach to the construction of agents. For this we need to consider the attributes expected of an agent from a non-intrusive perspective. Moreover, we need to aim for some additional properties that promote this approach.

3.2.1 Goals

- *Autonomy:* The agent needs to be independent. However, it should operate independently in such a way that it does not bother the user. It should not perform any intrusive operation without the user's consent.

- *Pro-activeness:* The agent has the capability to act without the user initiating the action. However, to maintain non-intrusiveness it should limit its activities to opportune times so that there is minimum interference with user actions.

- *Pattern Recognition:* The agent needs to determine patterns in the user's behaviour in order to provide assistance. However, detecting patterns should be done in such a way that it does not change anything in the user's environment or disturb her in any way.

- *Interaction/Social ability:* The interaction between the user and the agent should be non-intrusive. The user should not be forced to interact with the agent, but should be given the option of interacting when she wants. Moreover, the interaction interface should blend in with the user's environment so that the user does not have to learn or use special interface software.

- *Configurability:* The fact that an agent is running in the system and is watching over the user's shoulders is considered to be intrusion by some users. Thus, the user should have the power to stop the agent, or to configure it. The user should maintain a sense of control over the agent.

- *Privacy:* The agent will have access to confidential information. The agent should not compromise the user's security. It must ensure that all the user files it creates and uses are secure. If the agent uses the World Wide Web to communicate with the user it should take care that the messages are only visible to the user and not to the general public.

- *Ethics:* The agent should be a lightweight process. It should not have unauthorised access to system resources. If it interacts with the World Wide Web it should not cause unnecessary network traffic and should put minimum load on the system and the network.

3.2.2 Issues

To take a non-intrusive approach, the design should consider the user, the host system and the systems with which the agent interacts. A non-intrusive system should be designed with the goals mentioned earlier in mind. This section discusses the choices that arise while designing such a system.

In an environment like Unix where several processes can execute at the same time, autonomy and non-intrusiveness can be achieved by running the agent in the background. The choice of executing the agent as a continuously-running process or as a process that wakes up occasionally depends on the task in hand for the agent. If one needs to monitor the activities of the system and the user all the time, then a continuous process is a good idea. However, if activities during interaction between the user and the system have to be monitored then something like a process that starts up when the user logs in (e.g. using a .login or a .xxxrc file) seems to be a better solution. Nevertheless, one should avoid adding another continuously-running process to the system if the work can be done by using an existing process. For example, one can use the services of the cron clock daemon to carry out scheduled tasks. This helps to reduce the load on the host system.

If the agent is autonomous and pro-active it starts on its own to carry out certain chores for the user. It can do it either when the user is using the system or when the user is not using the system. To be non-intrusive it should try to run at times when the user is not using the system and/or there is a minimum load on the system. For example, the agent can be run late at night. One can do this by programming the agent process to wake up at an opportune time or use the facilities of cron to execute it. Again, if tasks can be achieved through cron, one should avoid adding another daemon to the system.

Patterns in the user's behaviour can be recognized by: observing her key strokes, mouse clicks etc.; asking her questions; or by observing the impact of her actions on the file system. Asking questions while the user is working might not be appreciated as they easily become annoying. Continuously watching her moves without disturbing her can be used, but requires the agent to run all the time. Analyzing her impact on the file system may be the best alternative as it can be done while the user is not using the system and does not require

the agent to run continuously. However, this approach does depend on the problem at hand. For example, "Open Sesame" from Charles River Associates Inc. is an agent that monitors the keyboard and mouse watching for repeating patterns. Such a system would need to operate when the user is on the system.

There are also several options for making the interaction non-intrusive. The agent could interact with the user by: sending her email messages; starting a new interface application; or by blending the interaction interface into an application used regularly by the user. Sending email forces the user to interact with the agent. Moreover, too many email messages might annoy the user. However, this approach can be useful in transmitting information that requires immediate attention. Another application interface increases the complexity of the work space. If the problem addressed by the agent is independent of other applications then this might be a good idea. However, if the problem requires collaboration with an existing application then the third choice seems appropriate, as it preserves the user's work environment and makes the interaction less novel.

For interaction with the web, the agent can either carry out all the operations at once, or space out its requests to put a minimum burden on the network and the web servers. The second approach makes the agent non-intrusive and well behaved. For example, to index the documents on a certain site, the agent could download all the documents frequently. This will increase the load on the targeted server and might even choke it if several agents decide to do so. Taking the non-intrusive approach, designing a system in which tasks are properly spaced and execute at opportune times is advisable. It should put a minimum burden on the host system and the systems with which it interacts.

The agent can either be configured from the interaction interface, or through a set up file provided with the agent. To reduce user effort, using the interaction interface is a better solution.

If the interaction interface is to be made available on the World Wide Web, there are again two choices: make it available to all web users, or to individual users. If the results displayed by the agent are not confidential, they can be viewed by all users. However if they expose confidential information, they should either be password protected or be available only from user's environment.

3.3 An Example System

One of the main challenges in agent research is to identify areas where the services of an agent can be employed successfully to incorporate the agent in the identified environment. One such application domain is that of managing information already collected from the World Wide Web. The collected information presents itself in the form of bookmarks that are pointers to user's favourite sites. We design an agent that takes the non-intrusive approach to the management of bookmarks.

3.3.1 Requirements

The interactions between a user and the Netscape browser were observed and it was decided that the following functionalities can be delegated to a software agent. We divided the functionalities in two broad categories.

- **Bookmark Management**

  The agent can be used to manage bookmarks and to detect patterns in user behaviour. The identified functionalities are

  - **Sorting of bookmarks:** To reduce the time taken to reach a bookmark, the bookmarks can be sorted according to the time of last visit. This way the bookmark visited *last* will occur *first*. However, this will cause bookmarks to be rearranged. Some users might not like the rearrangement of their bookmark file.
  - **Extracting recently-visited bookmarks:** The bookmarks that have been recently visited are likely to be revisited. Presenting them to the user avoids the sequence of going to the bookmark file and finding the URL
  - **Extracting old bookmarks:** Many bookmarks are no longer required by the user. The bookmarks that have not been visited for a long time can be presented to the user for possible deletion.
  - **Detecting patterns in URL access behaviour:** Some users like to visit sites on certain days of the month. For example, they might check the Top Ten music videos every Sunday, or read the issue of their favourite magazine on the 1st of every month etc. Detecting such a pattern in the user's behaviour will be useful in reminding her to visit a page on a particular day.

- **Deleting bookmarks:** The user can be given the option of deleting bookmarks from the bookmark file. The user will not have to find the identified bookmarks individually.

- **Page status Monitoring**

  The World Wide Web is dynamic in nature and it is difficult, if not impossible, to keep users informed of changes. It is useful if the pages bookmarked by the user are monitored for changes.

  - **Modification status:** The pages bookmarked by the user can be monitored for changes. For example, if a new article is added to a list of research papers, a researcher can benefit from the knowledge that it has changed.
  - **Existence status:** The bookmarks can also be monitored for pages removed from servers. The user can be informed that a particular bookmarked page no longer exists on a server so she should delete the bookmark.
  - **Relocation:** Pages are often moved for reasons like better access time, more storage capacity etc. The server usually puts a notification saying that the page has been moved to a new site. This notification may be removed later from the original server. If the user does not find the relocation notification, she will have to go through the whole process of searching for the page again. To prevent this, the bookmark pages be monitored for movement.

3.3.2 Architecture

The Bookmark Management Agent (BMA) was designed using the attributes mentioned earlier. A conceptual design, shown in Figure 1, was first constructed. The design demonstrates that the agent interacts with the World Wide Web and the user to satisfy its goals that are derived from the needs and interests of the user. The resulting architecture, shown in Figure 2, is described in the following sections.

The agent consists of four software modules.

- Web Interaction Module
- URL Management Module
- User Interface Module

- Common Gateway Interface (CGI) Module

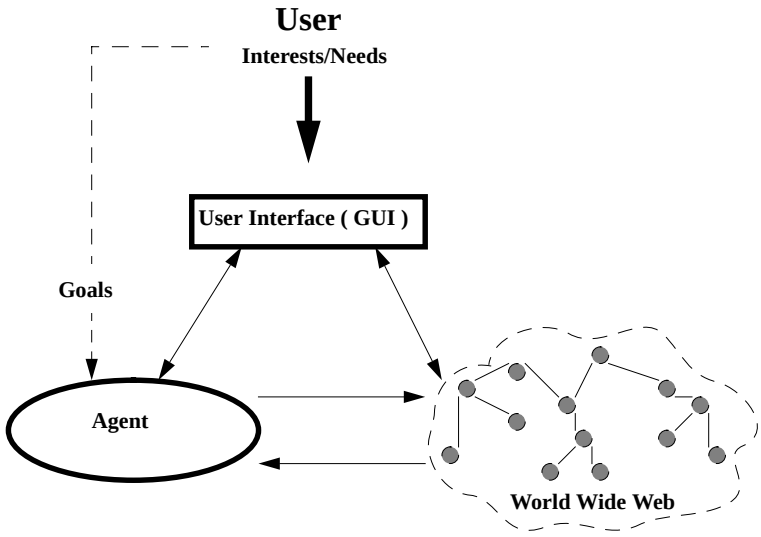

Figure 1: Conceptual design

The URL Management module and the Web Interaction module form the core of the agent. The URL management module consists of several sub-modules which perform the following tasks:

- Sort bookmarks
- Detect old bookmarks
- Delete bookmarks
- Detect and fix relocations
- Detect patterns

The bookmarks file is used by the URL management module to extract the URLs which are frequently visited by the user. These bookmarks are sorted so that the URL visited most recently appears at the top of the list and is more easily accessible. Furthermore, a list of URLs visited on the previous day is also made, to assist the user.

The list of bookmarks is scanned for URLs which have not been visited for a long time. The length of time can be specified by using the Customise Page.

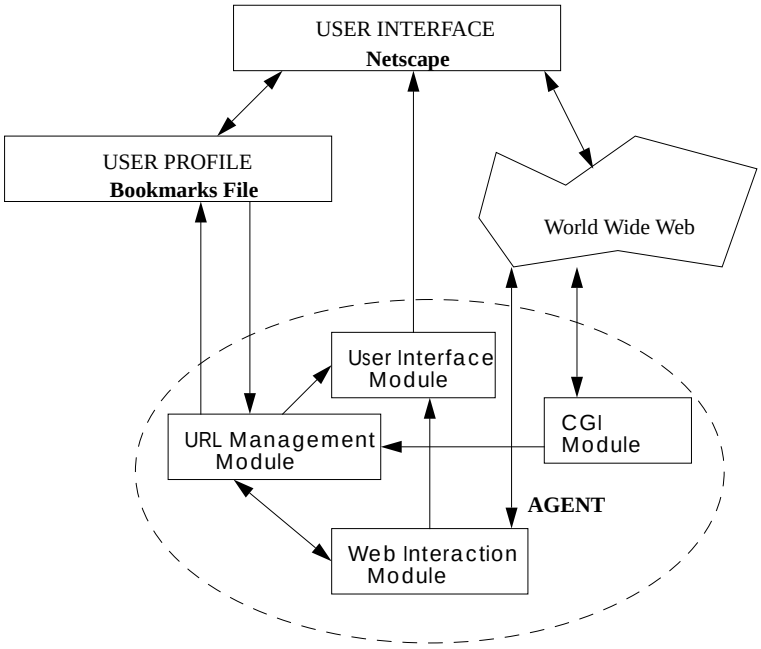

Figure 2: Bookmark Management Agent (BMA): System Architecture

A list of all bookmarks which are older than the specified time is prepared. Also, the user is presented with the option of deleting old bookmarks.

Periodically, the URL Management module tries to detect sites that have moved. It does so by scanning the bookmarked pages for relocation messages. If the module decides that a certain site has moved, it tries to fix the relocation by extracting the new URL and replacing the old URL with the new one. However, the decision to let the agent perform the substitution remains with the user.

The agent scans the bookmarks visited by the user every day and tries to determine patterns in the user's behaviour. For example, if the user visits a certain site every Wednesday, the URL management module will detect this. On the following Wednesday, the agent will present the user with a list of URLs she usually visits on that particular day.

The URL management module passes the URLs extracted from the bookmarks file to the Web Interaction module. The Web Interaction module performs two tasks.

1. *It checks the modification status of each of the URLs:* The server on which the bookmarked page resides is contacted and the time the page

was last modified is retrieved.

2. *Retrieves pages:* The server for each of the pages is contacted and the bookmarked page is downloaded.

The downloaded pages are passed on to the URL management module, which scans them for movement. Furthermore, a list of all the bookmarks which have changed since the user last visited them is made. The Web Interaction module also keeps track of URLs which no longer exist on the specified server and makes a list of such URLs. The user is given the option of deleting such bookmarks (dead bookmarks).

Once the various lists are prepared by the Web Interaction module and the URL management module, these lists are given to the User Interface module. The User Interface module generates the communication interface for the user. The user communicates with the agent using the forms presented to her. The various HTML pages and forms that are presented to the user are made by the User Interface module. These pages can be viewed by the browser. Furthermore, the user can go to the sites using the URLs displayed on the various pages.

The CGI module is responsible for collecting all the information from the user and passing it on to the URL management module for processing. It consists of various cgi-bin scripts which accept the information submitted by the user. The scripts convert the information into the appropriate format, and also generate an HTML confirmation to be returned to the user. This module is responsible for completing the dialogue between the user and the agent.

The agent is autonomous and runs at an opportune time or at a time specified by the user. The autonomy is achieved by using the cron clock daemon. The daemon executes the requisite software modules at a specified time.

At the time of installation, the agent is configured to put minimum load on the system and the World Wide Web. Thus, the frequency with which it performs various operations is set to a minimum. However, the user is given the option of customising the agent according to her needs. The user can customise the agent by specifying options using the *Customise Page*. This page is an HTML form which passes the customising information to the agent by using a CGI script. The user can specify, for example, the time she wants the agent to run, the frequency with which she wants the agent to perform certain operations etc. The script is a part of the CGI module that handles the information submitted by the user.

The customising information is passed to the URL Management module by the CGI module. This information is then used to reconfigure the agent according to the user's preferences.

4 Technical Issues

4.1 Introduction

The agent was constructed using Perl (Practical Extraction Report Language), Java, and HTML (Hyper Text Markup Language). An attempt to use the best features of each of these languages was made so that the processing is quick and efficient. Perl was used for recognizing patterns and text manipulation. Also, it was used to generate HTML pages on the fly. HTML pages were used to present the results to the user. This made the presentation of results, easily accessible and convenient. Java was used for interactions with the World Wide Web. Java's strong networking capabilities made retrieval of documents and checks for page status easier.

4.2 User Interface Module

It was decided that the User Interface would be completely HTML-based. The motivation for this was that the user would be able to access the agent pages through a browser (Netscape). This approach was preferred to running the agent interface as a separate application as it would increase the complexity of the user's environment. The HTML-based approach is more in line with keeping the agent simple and non-intrusive.

The User Interface module is responsible for presenting the user with a set of HTML pages. All communication between the agent and the user is through these HTML pages. The agent page consists of frames as shown in Figure 3. Frames let the main display window be divided into independent window frames, each simultaneously displaying a different document. One of the frames, which is static, displays a list of hyperlinks to all the agent pages. These pages display the status of various goals that the agent is trying to achieve. When the user clicks on any of these hyperlinks the adjacent frame displays the appropriate page. Since each of the agent pages presents the user with a set of URLs, the pages linked by the URLs can be viewed in one of the frame windows without leaving the agent page.

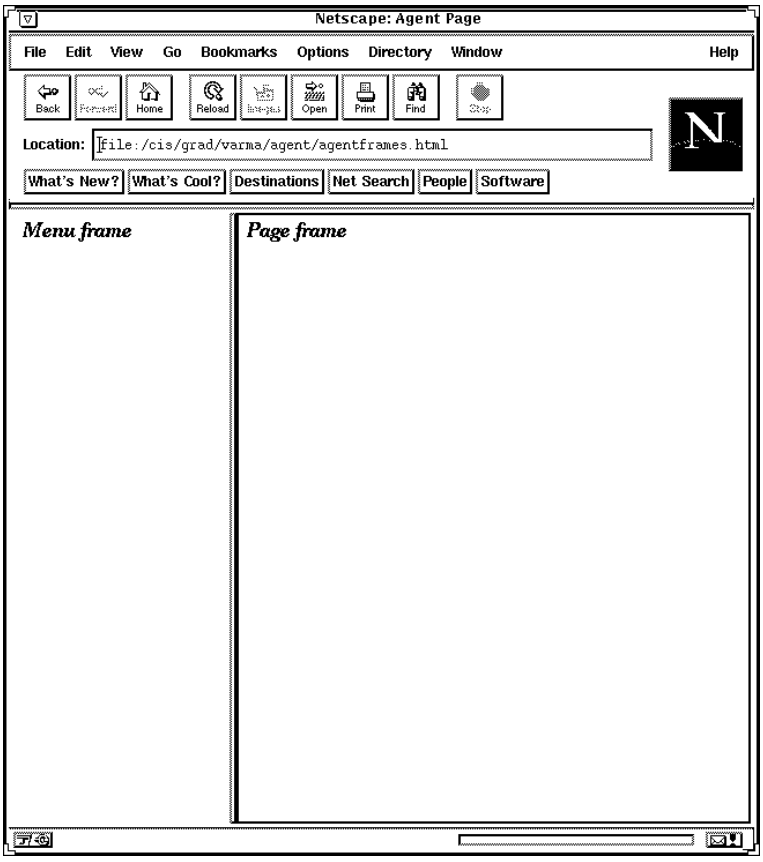

Figure 3: Frames

The other modules work on the URLs and prepare a list of URLs that have, for example, changed, moved, or are old. Each of the lists is stored in a text file. The User Interface module reads these text files and generates HTML pages to be presented to the user. These pages are either simple pages displaying the list of URLs, or are forms that can be submitted by the user. These pages are generated by Perl scripts. The text files are read by Perl scripts and converted into HTML format that can be displayed by the browser. Furthermore, the User Interface module generates an updated page every time there is a change in the information to be displayed by the page. To maintain privacy, these pages are not posted on the World Wide Web, but reside on the user's system. This makes them accessible only to the user.

4.3 URL Management Module

The URL management module was implemented in Perl due to its enhanced text manipulation features. The module first reads the bookmark file generated by Netscape, and extracts the bookmarked URLs. It then performs the following tasks:

- **Sorting of Bookmarks**

  The file is sorted according to the time of last visit. This makes the documents that have been accessed recently move above the others in the list. The user is now able to access the recently-visited site more easily, as she can reach it faster. However, this approach also has its drawbacks. For example some users get used to the location of the bookmark on the list and do not appreciate rearrangement of the bookmarks.

  The sorting is carried out in the following way: The bookmark file generated by Netscape is a tree structure. The file is divided into folders. These folders group the URLs on similar topics together. Each level of the tree is sorted in sequence until the whole tree of URLs is sorted. Note that, the original format of the bookmarks file is preserved.

  Another issue was realized while sorting the bookmarks. It was noticed that if the user has to access a bookmark that is in one of the sub-subfolders, she has to travel down the whole tree. To reduce this time in the next access the agent generates a list of URLs visited on the previous day and displays it on the next day. Since the user is likely to visit the sites again the next day, she does not have to traverse the whole tree looking for the bookmark.

- **Extracting Old Bookmarks**

  The agent allows the user to specify the length of time after which bookmarks are considered old. The agent scans the list of bookmarks and extracts the ones which are older than the time specified by the user. It does so by comparing the difference between the time of last visit and the present time with the length of time specified by the user. It then prepares a list of such bookmarks in the form of a text file. This text file is read by the User Interface module and a HTML form is generated. The form presents the user with a list of old bookmarks and check boxes. The user can select the bookmarks she no longer wishes to keep and submit to the agent. These bookmarks are deleted later by the agent.

- **Fixing Relocations**

  The normal sequence of events, when a document has moved, is that the user goes to a site, using a bookmark, and finds a page that informs her that the site has moved and leads her to the new site. This sequence of events can be automated by the agent. The way this is done is that the agent scans the page for relocation information. It uses the following criteria.

  - ***Keywords:***
    The agent reads each line of the page. If the page has keywords like *Moved*, or *Relocated* it guesses that the page has moved.
  - ***Number of URLs:***
    To get better results, the number of hyperlinks in the document are also counted. In a page describing a document that has moved there are typically at most two hyperlinks. If the number of hyperlinks is greater than two, the agent does not try to fix the relocation.

  Fpr those pages that agent is able to identify as pages that might have moved, it makes a list of such URLs as a text file. This file is used by the User Interface module to generate the HTML page for displaying the list to the user. The agent has the capacity to fix the relocations itself, but it needs permission from the user to do so. If the user wants the agent to fix relocations, the agent extracts the URL of the new site and replaces the old URL with the new one automatically.

  Presently, the agent can fix a relocation if the site only contains one hyperlink to the new page. If there is more than one hyperlink on the page the agent cannot determine which is the new location and thus does not fix the relocation.

- **Pattern Detection**

  The types of patterns that can be determined are: the sites the user visits daily, the sites that the user visits on a particular day of the week (e.g. Monday, Tuesday etc.) and the sites the user visits on a particular day of the month (e.g. 1st, 15th etc.).

  A pattern can be established by storing the URLs visited on each day and extracting the URLs that display the pattern after a certain number of days. To explain further, consider the problem of determining if the user visits a web site daily. The bookmarked URLs accessed on each

```
http://agents.www.media.mit.edu/groups/agents/
http://www.m-w.com/dictionary
http://www.cnn.com/
*****
file:/cis/grad/varma/agent/agent.html
http://agents.www.media.mit.edu/groups/agents/
http://www.cnn.com/
file:/usr/local/java_1.1.1/docs/index.html
*****
file:/cis/grad/varma/agent/agent.html
http://agents.www.media.mit.edu/groups/agents/
http://www.m-w.com/dictionary
http://www.cnn.com/
*****
file:/cis/grad/varma/agent/agent.html
file:/usr/local/java_1.1.1/docs/index.html
http://www.wp.com/75096/
http://agents.www.media.mit.edu/groups/agents/
http://www.m-w.com/dictionary
*****
file:/cis/grad/varma/agent/agent.html
file:/usr/local/java_1.1.1/docs/index.html
http://www.wp.com/75096/
http://agents.www.media.mit.edu/groups/agents/
*****
file:/cis/grad/varma/agent/agent.html
file:/usr/local/java_1.1.1/docs/index.html
http://www.wp.com/75096/
http://agents.www.media.mit.edu/groups/agents/
http://www.m-w.com/dictionary
*****
```

Figure 4: File for pattern detection

day are stored in a file shown in Figure 4. The agent adds a delimiter (*****) at the end of each block of URLs visited. Thus, one block has the URLs accessed on one day. Every day the agent appends this file. Once the number of blocks is six in number the agent calls the *pattern script*.

The pattern script extracts the URLs visited four or more times in the last six days. The assumption here is that if the URL is visited four or more times it is being visited on a daily basis. This takes into account the possibility of, for example, a long weekend where the user does not go to a site for three days. The agent also does not consider days when the browser was not used at all. It checks this by scanning the bookmark file. If no bookmarked URL has been visited on a day it assumes that the browser was not used on that day. However, just establishing the pattern does not help the agent adapt to changes in the pattern. To do this the first block from the file is removed after the call has been made to the *pattern script*. Thus, if the user stops visiting the sites the URLs will eventually be removed from the file and the agent will adapt to this

change.

The way this pattern is established can be extended to determining patterns for weekly and monthly visits. This would require the agent to store the URLs visited weekly and monthly. The agent can then determine if there is a pattern in the user's behaviour by observing the frequency. However, unlike the daily visit patterns, the threshold for establishing a pattern can be reduced. For example, the agent can check after every three weeks if the user has visited a site regularly for two or more weeks. If she has, it assumes a pattern. Note that more information about the user's behaviour needs to be stored if more patterns are to be determined.

- **Deleting Bookmarks**

  The agent presents the user with a list of bookmarks such as old bookmarks, dead bookmarks etc. These bookmarks might not be required by the user anymore. However, the agent does not assume this, but presents the user with the option of deleting them by presenting the user with a list of such bookmarks as a HTML form. The user can select the bookmarks she wishes to delete by checking the checkboxes. The CGI module makes a list of the bookmarks to be deleted. The list is used by the URL management module to delete the bookmarks no longer required by the user.

  Moreover, the agent keeps a list of bookmarks it has deleted in case the user changes her mind later. However, these bookmarks are not kept indefinitely. The user can specify the time she wants the agent to keep deleted bookmarks.

  To implement this, a time stamp is included with every deleted bookmark. The age of the deleted bookmark is compared to the time specified by the user. The deleted bookmark is removed from the list once it is older than the specified time.

4.4 Web Interaction Module

The Web Interaction module is responsible for communication with the WWW. The Java language was chosen to implement this module due to its strong networking capabilities. Java provides classes that make it easy to open connections with the host and retrieve documents. The Web Interaction module performs two main functions

- **Retrieve Documents:**

  With the increasing number of robots on the web there is growing concern about the number of automated software agents visiting various servers. Due to the recursive nature of automatic document retrieval, these servers get choked by the excessive number of requests. Even though our agent retrieves only the bookmarked pages from a server, it takes into account some of the suggestions for good robot behaviour [15]. These include running the agent at appropriate times, not visiting any unauthorized spaces, and spacing out requests to other servers. To reduce network traffic, the documents are retrieved one by one. Each downloaded page is scanned before another page is retrieved. This saves space as all the pages need not be stored. To make the process more efficient, the user can direct the agent to work on a chunk of URLs at a time. For instance, if the user specifies the chunk size of 10 URLs, the agent will retrieve 10 documents and then sleep for a 30 seconds before retrieving the next 10. As a consequence the agent puts less burden on system resources and allows other processes to run.

  Each retrieved document is passed on to the URL management module which scans it to look for moved or dead sites. While retrieving documents the agent also notices pages that no longer exist on servers. We call such URLs dead bookmarks. Usually, if a document is no longer present on a server, the server returns a HTTP error message such as:

  HTTP/1.0 404: Object not found

  The Web Interface module prepares a list of such bookmarks and this list is presented to the user as a HTML form. The user can deleted such bookmarks by using the form provided. Of course, sometimes, the server returns a page saying that the document no longer exists. In such cases, the agent has to scan the page and look for keywords like *Not found*.

- **Check Status of Pages**

  The agent prepares a list of bookmarks whose pages have changed since the user last visited them. It does so by contacting the servers and asking them the date a particular document was last modified. Note that the agent does not need to retrieve the page in order to check if it has been modified. It just compares the date of last visit of each of the document with the date it was last modified. The agent then prepares a list of documents that have changed since the user last visited them.

This method of checking is, however, dependent on the server on which a particular page resides. Some servers do not serve the date a document was modified, either for security reasons or because the document gets modified too frequently. The agent does not include such URLs in the list for changed bookmarks.

4.5 CGI Module

The CGI module is responsible for processing the user's requests and sending confirmation back to the user. It consists of cgi-bin scripts which carry out various tasks. Since the agent interface is HTML-based, CGI was chosen to execute the various user requests. The CGI module handles two kinds of requests.

- **Deletion Request**

  The agent presents the user with the option of deleting certain bookmarks. The user checks the bookmarks she wishes the agent to delete, and submits the form. The submitted form invokes a perl-cgi script that is executed on the server. The cgi script records the bookmarks to be deleted in a file and sends the user an HTML confirmation. The script also includes a time stamp indicating the time the deletion was requested. This time stamp is used to maintain the list of deleted bookmarks by the URL management module. The confirmation displays the list of bookmarks that will be deleted by the agent.

- **Customisation Request**

  To customise the agent, the user is presented with the Customise page. The page is a HTML form that displays options as drop down lists and radio buttons. The user selects from the given options and submits the form. The submitted information is collected by the cgi script and a confirmation is returned. The cgi script prepares a crontab file according to the submitted options. The crontab file is used by the cron clock daemon (described in the next section) to schedule the agent according to the submitted options.

4.6 Running the Agent

Running the agent sporadically can be implemented by using the *cron clock daemon*. A crontab file specifying when the agent has to execute is generated

and submitted to the daemon to execute. The daemon executes the scripts according to the schedule specified in the crontab file. However, this approach does require the cron clock daemon to be executing on the client system.

5 Use and Discussion

The agent was tested by simulating the user and World Wide Web behaviour over a period of four weeks. The methodology used to test the agent and the observations made while using the agent are presented in the following sections.

5.1 The Agent Page

The Agent page, shown in Figure 5, presents the user with the interface to the agent. The user can either make this page the home page, or visit the agent page when she wants.

The page frame (frame on the right side) in the agent page starts with a page downloaded from the WWW. Various options were explored to decide the default page for the agent. These options were implemented to observe the behaviour.

- **Home Page:** The agent page starts with the home page site specified by the user.

  Results: This option was not very useful as the agent page displays the home page in a smaller frame.

- **Reminder Page:** The agent page starts with the page the user usually visits on that particular day. It serves to automate the task of the user going to that site on that day.

  Results: The results seemed favourable and useful, but did not seem to work well if there are few patterns in the user's behaviour or if the behaviour frequently changes. The agent will display a blank page if the agent fails to come up with a site that the user visits on that particular day.

- **Last Visited Page:** The agent page starts with the page last visited by the user.

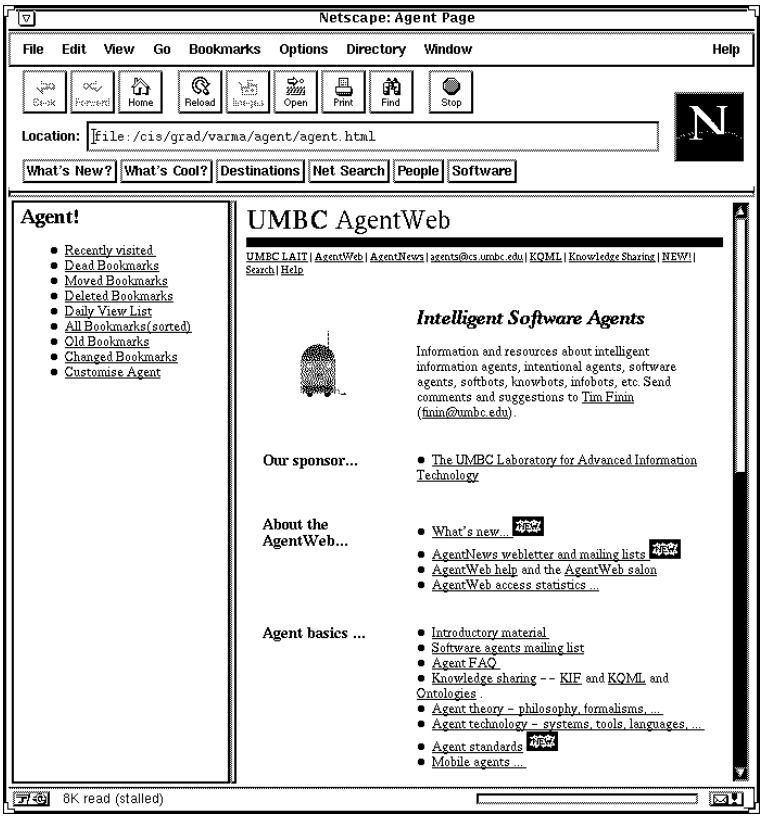

Figure 5: The agent page

Results: This option works on the assumption that the user is likely to look at the page she visited last. Compared with the Reminder page option, the agent page never starts with a blank page. However, it does force the user to go to a particular site and this affects the pattern followed by the user to some extent.

5.2 The Recently Visited Page

This page shown in Figure 6 is the result of clicking on the *recently visited* URL in the Menu frame. The page displays the URLs visited on the previous day.

Test: A set of URLs were visited for a few days and the results on the recently visited page were checked. A day was skipped in the middle to check if the agent displays a blank page.

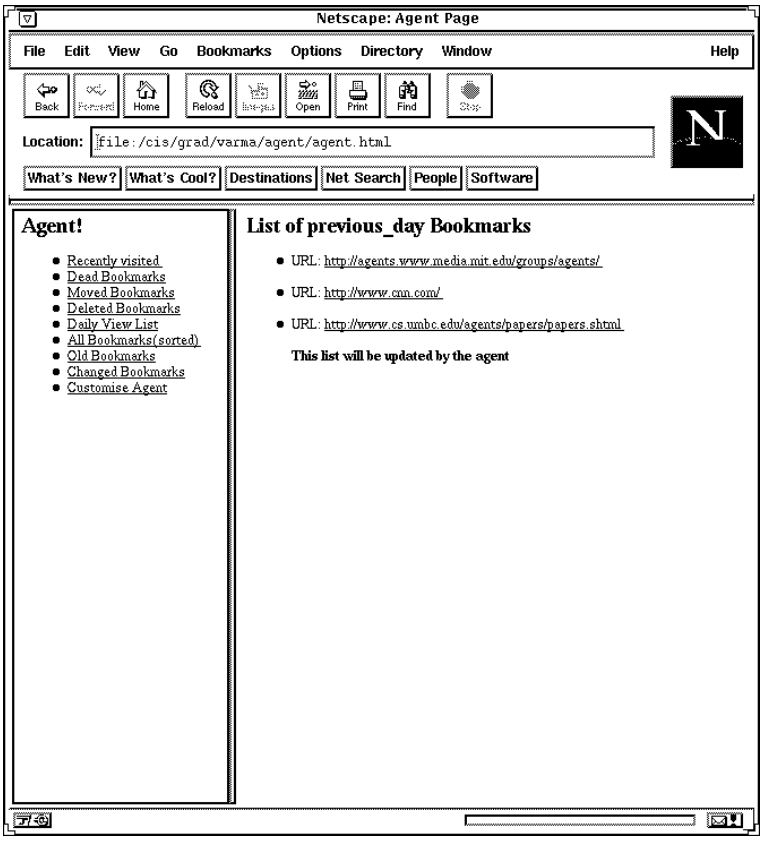

Figure 6: The recently visited page

Results: The agent displayed the URLs visited on the previous day. When a day was skipped the agent displayed the URLs visited on the day before the skipped day.

5.3 The Dead Bookmarks Page

This page, a snapshot of which is shown in Figure 7, displays URLs for bookmarked pages that no longer exist on servers. It can be visited by clicking on the *dead bookmarks* URL on the menu frame. The page is displayed in the page frame. The user can delete URLs by checking the check boxes provided next to the URLs and clicking on the Delete button.

Simulation: The dead bookmarks were simulated by bookmarking two test pages and then removing these pages from the server.

Results: As shown in Figure 7, the agent displays URLs that no longer

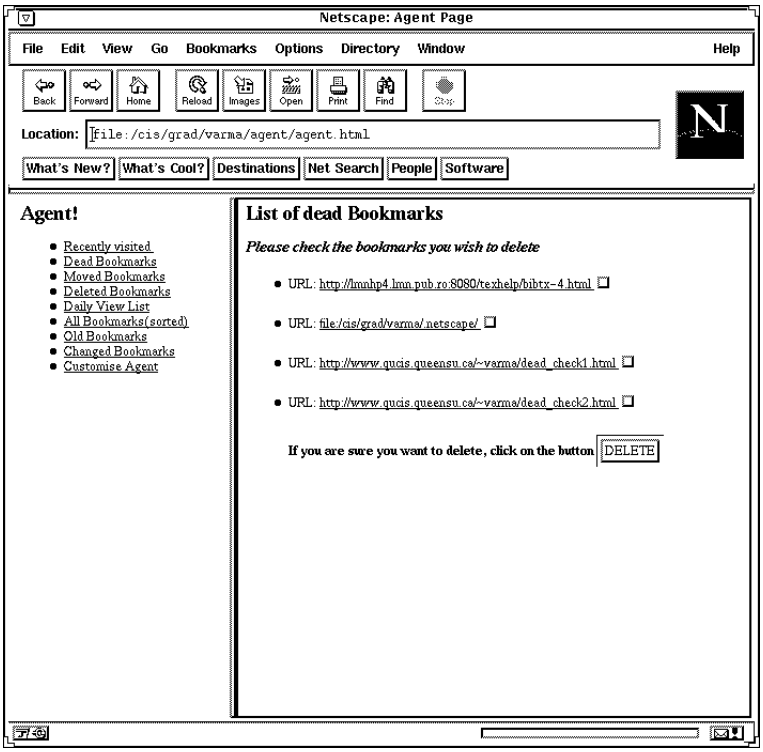

Figure 7: The dead bookmarks page

exist on servers. The user can confirm this by clicking on the displayed URLs. Figure 8 displays an example of this. However, the agent also considers a bookmark to be dead if the server does not respond. So sites that do not respond, or are very slow in responding, are also included in the list. These are not differentiated as the agent cannot determine conclusively whether the server is just slow or has been removed. This is useful, if the user does not assume that all the URLs listed on this page no longer exist. The delete option lets the user decide whether the bookmark is to be deleted. This is in line with the non-intrusive approach followed by the agent.

5.4 Moved Bookmarks Page

This page displays URLs for bookmarked pages that have moved. Figure 9 shows an example of a moved site detected by the agent. The moved bookmarks page can be viewed by clicking on the *moved bookmarks* URL on the Menu frame. As shown in Figure 10 the page will display the notification if

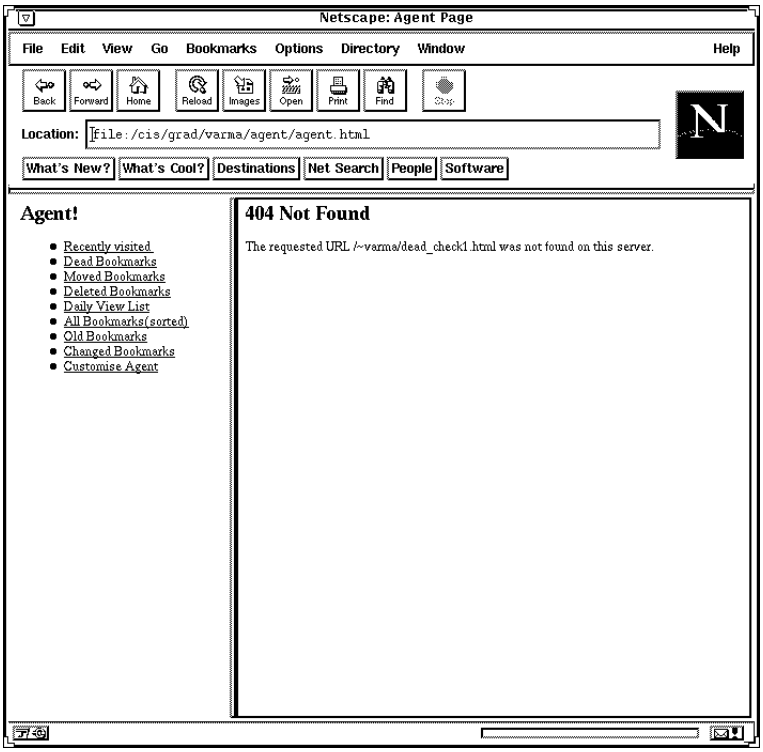

Figure 8: Dead bookmark confirmation

the page has indeed moved. The agent has the capacity to fix the relocation.

Simulation: The moved page condition was simulated by creating a page that had an appropriate notification and then bookmarking it.

Results: The agent was able to successfully detect that the test page had a notification and fixed the relocation. Note that the decision to let the agent fix the relocations still rests with the user. During its test run the agent detected one site that had a moving notification. This was expected considering that the agent was working with only one bookmark file. However, the agent does not fix all the relocations it encounters. It does so only if it is certain, and the rule for fixing the relocation is satisfied. Thus the agent only *tries* to fix the relocation but does not guarantee it. The heuristics used to detect and fix the relocations need refinement. With the present heuristic, the agent gets confused if there are more than two urls on the notification site and does not know which one to substitute.

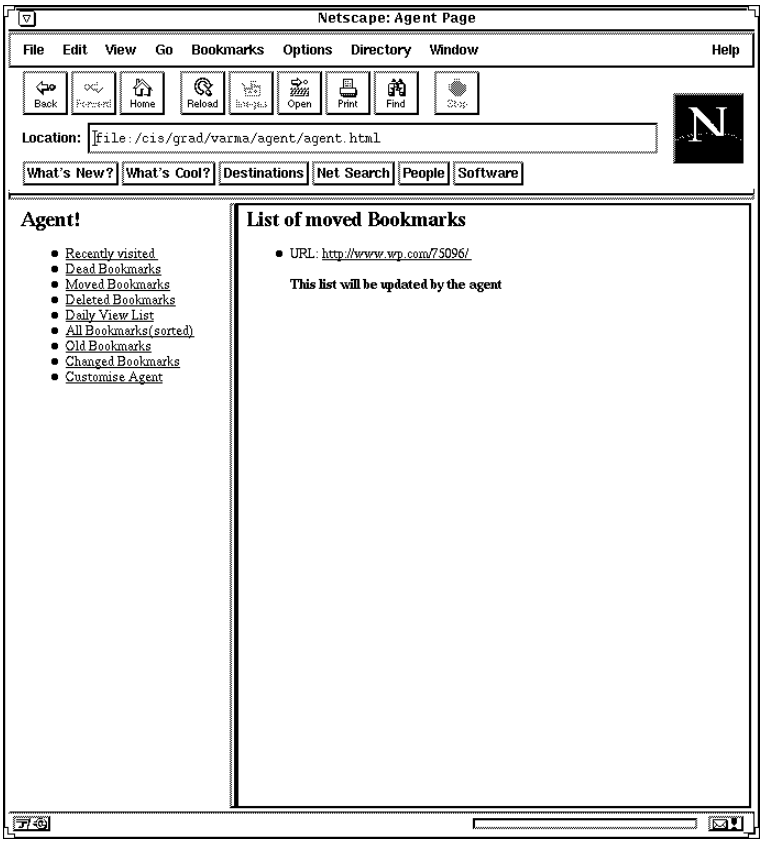

Figure 9: The moved bookmarks page

5.5 Daily To-view Page

The page shown in Figure 11 displays a set of URLs that the user usually visits on that day. This page can be reached by clicking on the *daily view list* URL on the menu frame. The page shown in Figure 11 was obtained after running a simulation on the agent.

Simulation: To check that the agent does detect a pattern, a set of pages were visited every day. The pattern in the visits was changed after a week to check that the agent adapts to the change. This was done by not visiting some sites after a week.

Results: It was noticed that the agent did not display any URLs on the page for the first six days. After six days, it started displaying the URLs visited more than four times in the last six days. Moreover, as the access behaviour was changed the agent stopped displaying the URLs that were no

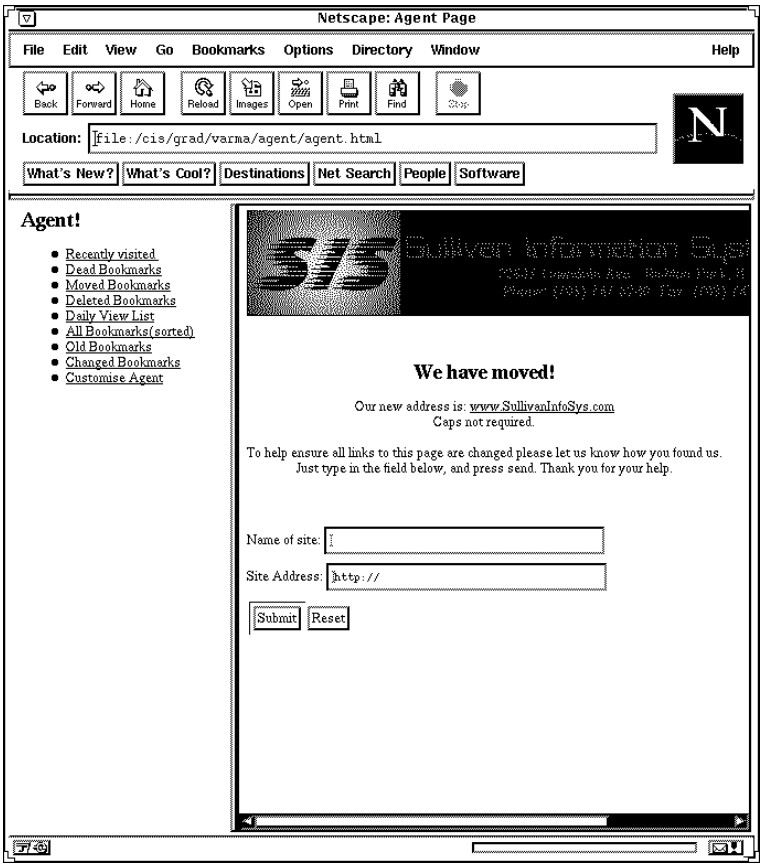

Figure 10: Page displaying the moving notification

longer being visited by the user.

5.6 Deleted Bookmarks Page

This page can be viewed by clicking on the *deleted bookmarks* URL on the menu frame. It displays the set of URLs that have been deleted by the user as shown in Figure 12.

Test: To test this, a set of bookmarks were chosen from the old bookmarks page shown in the Figure 14 and submitted for deletion to the agent.

Results: The agent deleted the requested bookmarks in the night and displayed them on the deleted bookmarks page the next day.

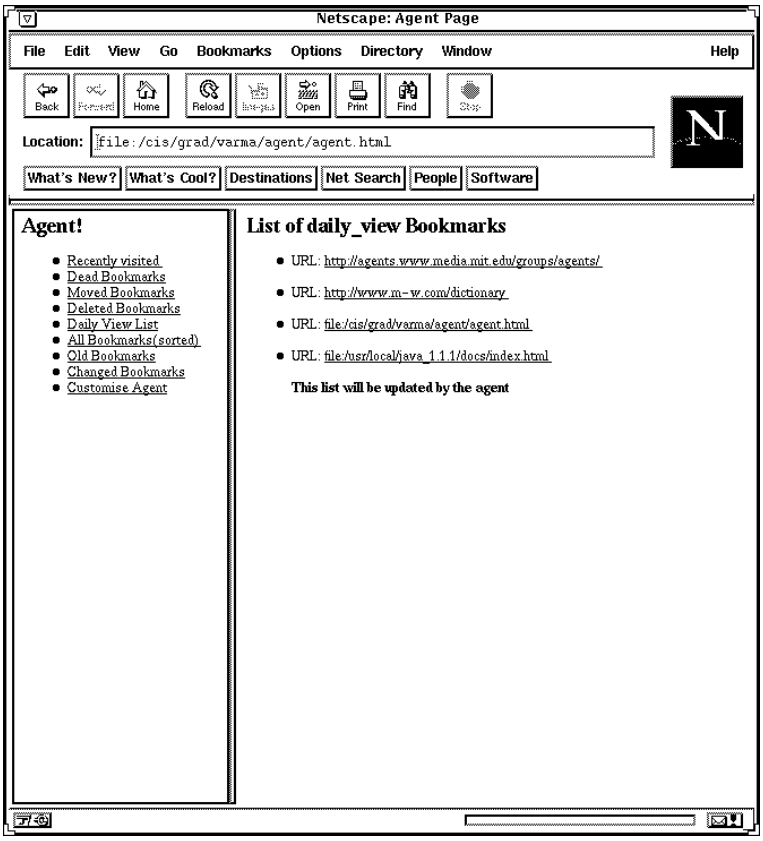

Figure 11: The daily to-view page

5.7 All Bookmarks Page

This page can be viewed by clicking *All Bookmarks(sorted)* on the Menu frame. It displays the sorted list of all the bookmarks. Figure 13 shows such an example. The sorting is done according to the time of last visit. The agent sorts the bookmarks at a frequency set by the user.

Test: The agent is given a sample bookmark file with several levels of bookmark categories. The file also had bookmarks scattered between the folders.

Results: The agent could sort the bookmarks in each of the folders. It also collected the scattered bookmarks and moved them to the top of the file. The drop-down menu displaying the bookmarks is also rearranged in sorted order. This improves the time taken by the user to reach popular bookmarks.

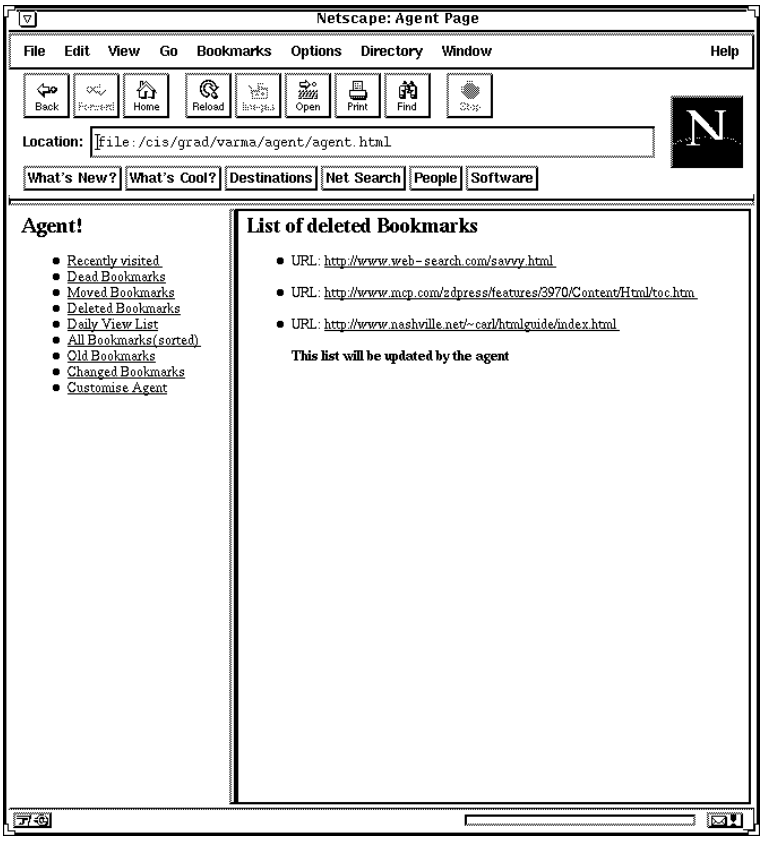

Figure 12: The deleted bookmarks page

5.8 Old Bookmarks Page

This page displays the bookmarks that are older than the time specified by the user. It can be viewed by clicking on the *old bookmarks* URL in the Menu frame.

Test: The agent was configured to display all the bookmarks that are older than one week.

Results: Figure 14 displays the resulting HTML page. It presented the user with a list of old bookmarks with checkboxes adjacent to each URL. This presentation allowed the user to delete the URLs that are older than the specified time. The user can do so by clicking on the *Delete* button.

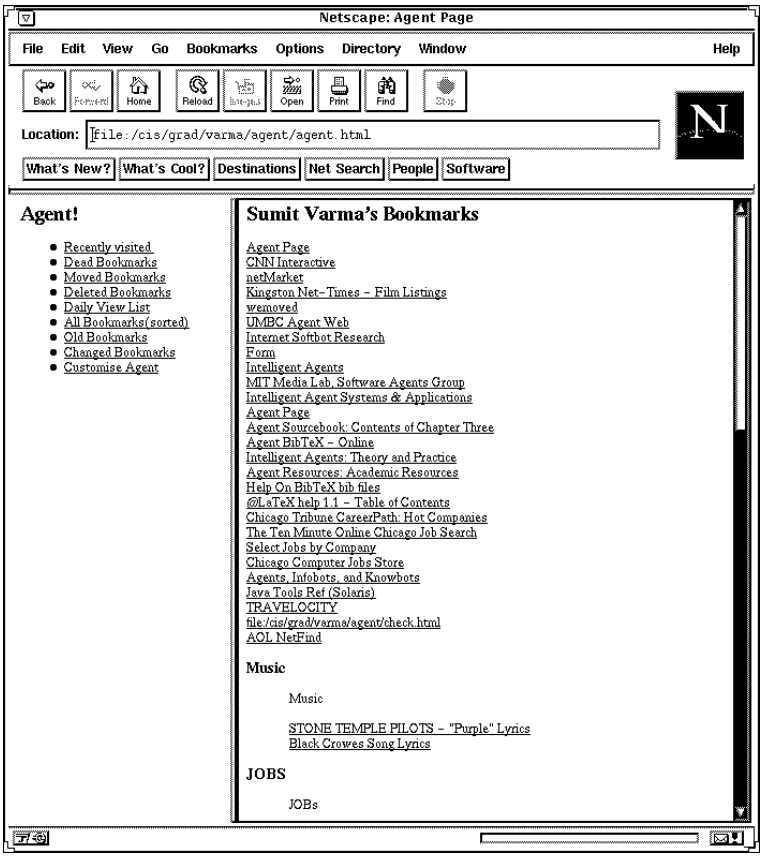

Figure 13: Sorted bookmarks

5.9 Changed bookmarks page

This page shown in Figure 15 can be viewed by clicking on the *changed bookmarks* URL in the Menu frame. Bookmarked pages that have changed since the user last visited them are displayed on this page.

Test: The agent was configured to check for changed bookmarks every two weeks. A test page was set up and changed after it had been visited.

Results: Several bookmarks that had changed were detected by the agent. The agent displayed them on the Changed Bookmarks page shown in Figure 15. The user can visit these sites by clicking on the appropriate URL. Also, the agent detected that the test page had changed and included it in the list of bookmarks that had changed. The result of this test was confirmed by going to that page. It is shown in Figure 16.

Experiment: The agent detects that a page has been modified by retriev-

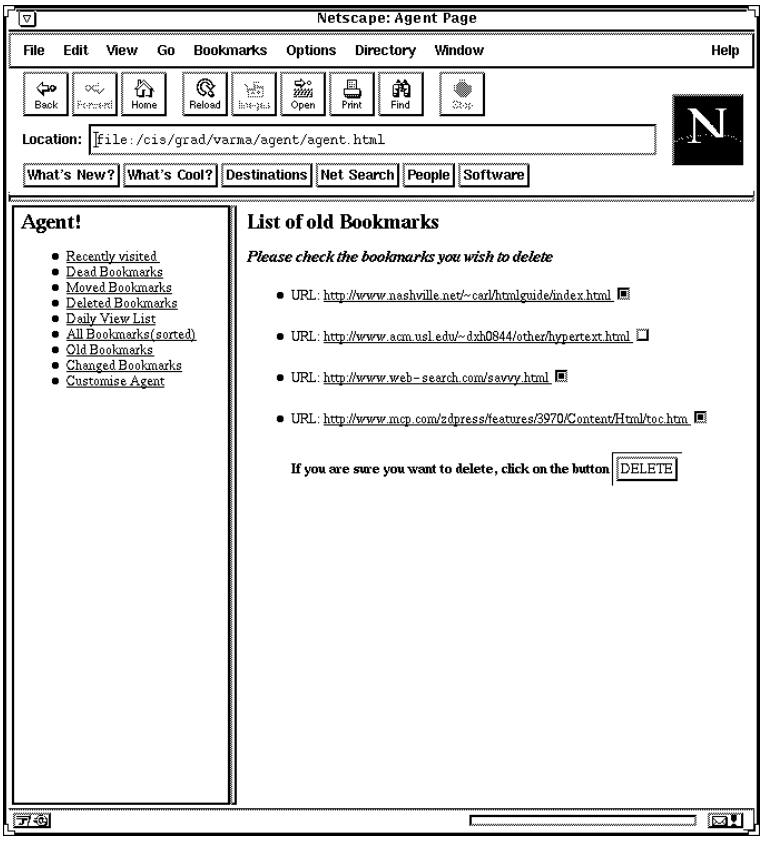

Figure 14: Old bookmarks page

ing the last modified date of the page from the server. However, some servers do not respond to this query. The status of these sites is indeterminate and is not included in the list by the agent. To estimate the percentage of these sites an experiment with 100 URLs picked randomly from several search engines was carried out. The aim was to count the number of hits.

Results: Out of the 100 sites 68 sites responded to the query of the last modified date for a page. On analysing the sites that did not respond we found that most of these site changed too frequently. Examples include CNN, Alta vista, and Yahoo.

5.10 Customise Agent Page

The Customise agent page shown in Figure 17 lets the user schedule the agent according to her preference. A set of options are presented to the user. The

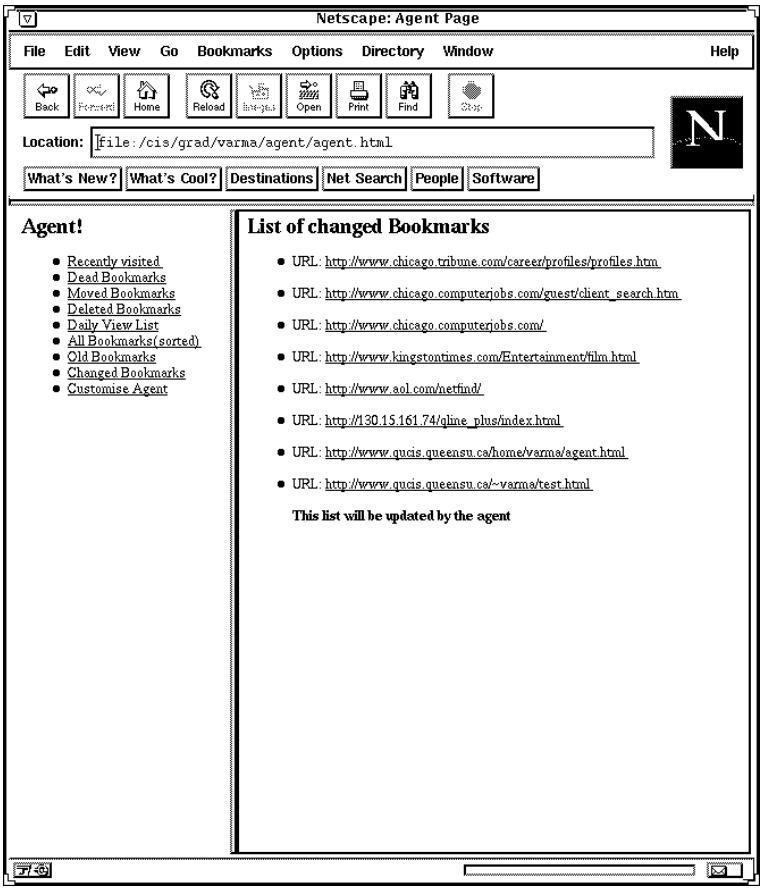

Figure 15: Changed bookmarks page

agent can be customised by selecting options and clicking on the *Customise* button. The option to stop the agent is also presented to the user.

Test: The agent was customised and stopped to test if the agent reconfigures itself according to the user's preference. The customisation was checked by scheduling the agent at a different time with a new set of options.

Results: If the agent is stopped it sends back a confirmation message. If the user chooses to customise the agent, a confirmation like the one displayed in Figure 18 is returned. The agent could reconfigure itself according to the user's preferences while running.

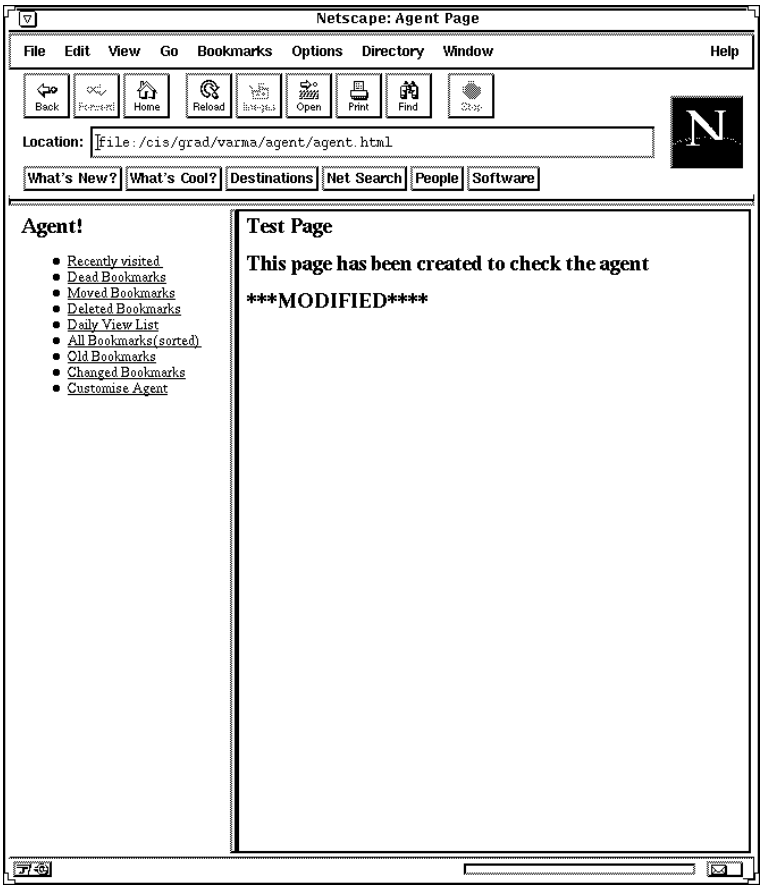

Figure 16: Confirmation of change

5.11 Comparison

Lieberman and Maulsby [13] discuss a spectrum of approaches to accomplishing tasks with a computer, comparing them in terms of their effectiveness and ease of use. The following section discusses the Bookmark Management Agent (BMA) in comparison to other approaches. Among the various approaches, direct manipulation approaches include applications like word processors where, for example, a user has to click on a button to execute a task. The "English Butler" is a pre-educated intelligent agent that understands a good deal of natural language and already knows how to carry out a variety of tasks; the user needs only give a vague description and the Butler reasons its way to an executable program. [8] presents one such example.

Intelligence: C++ and direct manipulation interfaces show little intel-

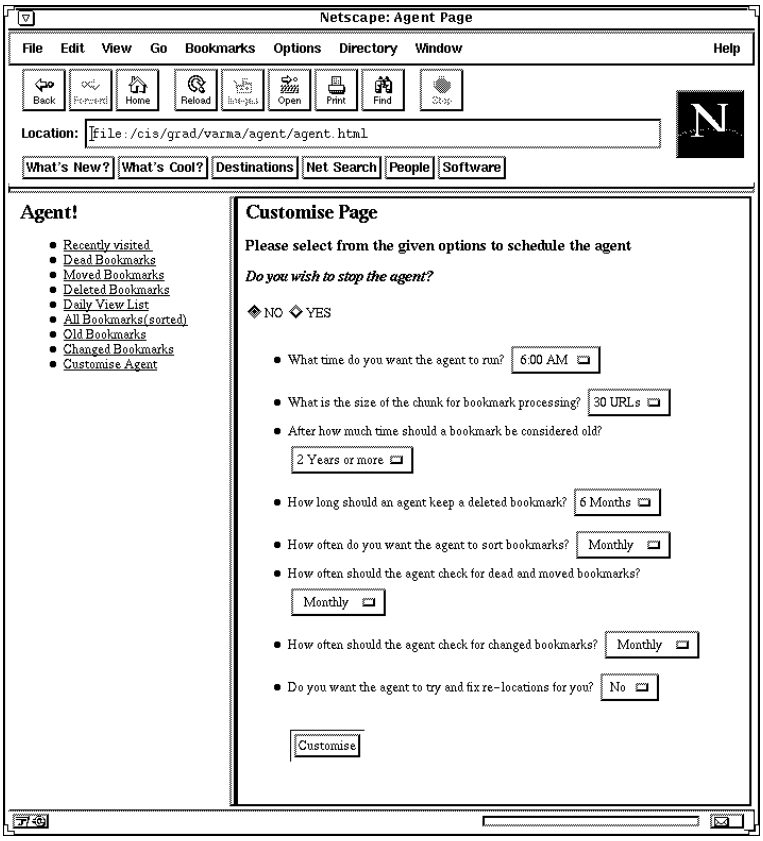

Figure 17: Customise page

ligence, other than program optimization and smart modes like automatic formatting of documents, or correction of words. BMA shows intelligence in determining patterns and fixing relocations. The approach is based on a set of rules. The English Butler rates higher on the intelligence scale as it is backed up by a knowledge base and a greater number of rules.

Run-time Adaptability: This refers to the system's ability to change its program in response to changing conditions. A C++ program, once compiled, can no longer adapt to changes. Direct-manipulation user interfaces generally do not adapt their behaviour except to store default values. BMA is adaptable in the sense that it can reschedule/reconfigure itself according to user's preferences if the user chooses to customise the agent. It also adapts to the changes in behaviour of the user. For example, if the user stops going to a site she visits daily the agent will detect this change in the pattern and adapt eventually. However, the agent fails to adapt if it faces a condition it has not

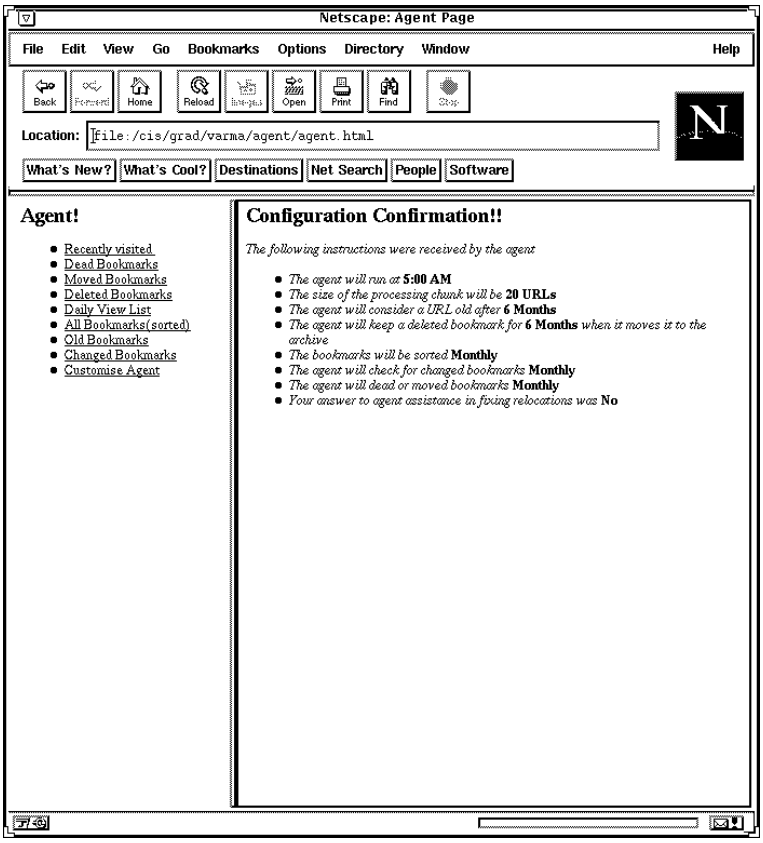

Figure 18: Customise confirmation

faced before. This is so because the agent decides most of the things based on certain rules. If the rules are not satisfied, it does not act. The English Butler must be highly adaptable to convert a vague idea into an executable program.

Programmability: This scale concerns the degree to which programmers can determine a system's behaviour. A C++ programming system enables programmers to describe any computable task and to choose an optimal method for doing so. Direct-manipulation interfaces are inherently nonprogrammable except some that offer macros for automating a sequence of actions. Our agent and the English Butler are also not truly programmable as they decide themselves how to accomplish a task.

End-user Control: The issue of end-user control is a controversial one in the field of agents. A C++ program gives the user no control except through the user interface. A direct manipulation interface, on the other hand, provides a high degree of control as long as it is easy to understand. However, the direct

manipulation interfaces today are too feature-rich. BMA tries to let the control remain in the user's hand by not doing any intrusive operations without the user's consent. The English Butler can get out of hand if it misinterprets some instruction.

Effort of Instruction: This involves the cognitive and physical operations needed to transmit knowledge from the user to the computer. Programming in a formal language requires high-effort. Direct manipulation interfaces minimize this effort - until the tasks get repetitive. Delegating tasks to the English Butler is simple in theory. In practice, carefully describing a task in words is often more difficult than doing it. BMA, on the other hand, requires minimum effort on part of the user.

Effort of Learning: Programming a computer is the greatest hurdle that the users encounter. Direct manipulation interfaces have reduced this effort to a great extent, but with the growing number of features the effort in learning is increasing. BMA requires little effort of learning as it is executing most of the tasks autonomously. The English Butler requires little or no effort unless it has unpredictable limitations of which the user must be aware.

6 Conclusions

An information management system must reduce the workload of the user. It should help the user deal with the information overload by carrying out certain tasks autonomously. We used the techniques from the field of software agents to address information management. We have found that software agents are a useful approach to designing information management systems, as they reduce the burden on the user. Furthermore, software agents can carry out certain chores without direct manipulation by the user.

The work presented in this report takes a non-intrusive approach to the design of software agents. Our search for an example domain to apply this approach led us to realize that agents can play an important role in managing information by cooperating with existing applications. We used this approach to design a prototype system (Bookmark Management Agent) that helps the user deal with the information available from the World Wide Web by managing her bookmarks. It also detects patterns in the user's behaviour and tries to help. Our experience with the construction of BMA led us to draw the following conclusions:

- Results show that our efforts to keep the agent non-intrusive were re-

warding. The agent gives total control to the user and works for the user in the background. Sorting of bookmarks is the only intrusive operation it performs without the user's consent. Since it is aimed at helping the user, we let the agent have that functionality. However, the real judges of this will be the users.

- The agent was successful in detecting patterns in the user's behaviour but it takes some time for the agent to establish a pattern. Another approach would be to let the user tell the agent what sites she visits on which days. However, this would be a tedious thing to do and will put a burden on the user

- The agent was scheduled using the cron daemon. This approach was preferred to running an agent daemon that would wake up occasionally. However, we could not establish how the system would behave if several agents were running. Moreover, while testing the agent it was noticed that the cron daemon would sometimes miss executing a task when two tasks were scheduled at the same time.

- To incorporate the agent in the user's environment we wanted to make the user interface available through the browser. There existed two choices, a Java applet or a HTML page. Java applets take time to load and the browser must have the option to execute applets turned on. HTML provided a much easier and better option. We found that user interfaces consisting of buttons, drop-down lists, text boxes etc can easily be constructed using HTML. The communication was carried out using CGI. However, a potential problem with CGI is that not all users have access to cgi-bin directories and this would limit the agent in some cases.

- To decrease the burden on the network and the system we followed some advice given by [15]. The approach does slow the agent, but creates less load on systems and networks. Moreover, the slower approach can be taken here as the user does not expect immediate results.

- Text processing was a major component of implementing the agent. Both Java and Perl were tried for this purpose. We found that Perl is far better equipped in dealing with regular expressions and is much faster. Java on the other hand does not handle regular expressions very well. Moreover, simple operations like appending a file required writing of an Append file

class. However, Java is equipped with classes that can make networking easier.

- While checking for pages that had changed we found that some servers did not respond to our queries. We realized that most of these sites changed too frequently. For example, news sites like CNN or search engines update their pages almost every twenty minutes. We also realized that a page does not have to be retrieved to check if it has changed. This led us to separate the functionality of checking for moved pages from checking of pages that have changed. We feel steps like these help in reducing the load on networks.

- The functionality to fix relocations automatically is included in the agent. However, while running the agent we discovered some sites have moving notifications with two or three URLs. This confuses the agent and the agent does not fix such a relocation. We feel that this limits the agent and definitely leaves room for improvement.

- The agent reminds the user to visit certain sites. This increases the number of times a particular site is visited. In a way, this proves beneficial to the owners of that site by making the site more popular.

- The Bookmark Management Agent is strongly coupled to the Netscape browser and this limits its capacity to help the users of other browsers. Nevertheless, this approach enhances the usefulness of an existing application like the Netscape browser and helps to keep it simple.

To sum up, agents are a promising approach to designing information management systems. Moreover, taking a non-intrusive approach to constructing agents helps us design agents that we believe will be more readily accepted.

References

[1] R. Armstrong, D. Freitag, T. Joachims, and T. Mitchell. Webwatcher: A learning apprentice for the world wide web. March 1995. Appeared in the 1995 AAAI Spring Symposium on Information Gathering from Heterogenous, Distributed Environments. URL: http://www.cs.cmu.edu/afs/cs.cmu.edu/project/theo-6/web-agent/www/project-home.html.

[2] A. K. Caglayan and C. G. Harrison. *Agent Sourcebook*. Wiley Computer Publishing, 1997.

[3] M. Coen, B. Selman, and H. Kautz. Bottom-up design of software agents. *Communications of the ACM*, 37:143–146, July 1995.

[4] International Data Corporation. World Wide Web statistics. A web page showing research findings about the World Wide Web. URL: http://www.idcresearch.com.

[5] Quarterdeck Corporation. Web compass. The Web Compass is available for download at this site., 1997. URL: http://www.quarterdeck.com.

[6] D.Gilbert and P.Janca. IBM intelligent agents. Appeared as a white paper on the IBM Intelligent agents home page, 1997. URL: http://www.raleigh.ibm.com/iag.iaghome.html.

[7] David Eichmann and Jun Wu. Sulla - a user agent for the web. *WWW*, 1996. URL: http://rbse.jsc.nasa.gov/eichmann/home.html.

[8] O. Etzioni and D. Weld. A softbot-based interface to the internet. *Communications of the ACM*, 37:72–76, 7 1995.

[9] Stan Franklin and Art Graesser. Is it an agent, or just a program?: A taxonomy for autonomous agents. In *Proceedings of the Third International Workshop on Agent Theories, Architectures and Languages*. Springer-Verlag, 1996.

[10] M. R. Genesereth and S. P. Ketchpel. Software agents. *Communications of the ACM*, 37:48–53, November 1995.

[11] Don Gilbert, Manny Aparicio, Betty Atkinson, Steve Brady, Joe Ciccarino, Benjamin Grosof, Pat O'Connor, Damian Osisek, Steve Pritko, Rick Spagna, and Les Wilson. White paper on intelligent agents, 1996. URL: http://activist.gpl.ibm.com:81/WhitePaper/ptc2.htm.

[12] K. Heilmann, D. Kihanya, A. Light, and P. Musembwa. Intelligent agents: A technology and business application analysis. Technical report, Intelligencia Inc., 1995. URL: http://haas.berkeley.edu/~heilman/agents/.

[13] D.Maulsby H.Lieberman. Instructible agents: Software that just keeps getting better. *IBM Systems Journal*, 35, 1996.

[14] K. Indermaur. Baby steps. *Byte*, March 1995.

[15] Martijn Koster. Robots in the web: threat or treat. *ConneXions*, April 1995.

[16] P. Maes. Agents that reduce work and information overload. *Communications of the ACM*, 37:31–40, November 1995.

[17] P. Maes. Artificial life meets entertainment: Life like autonomous agents. *Communications of the ACM*, 37:108–114, July 1995.

[18] P. Maes. Intelligent software. *Scientific American*, 273:84–86, September 1995.

[19] M. M. Maudlin. Lycos: Design choices in an internet services. *IEEE Expert*, pages 8–11, January 1997.

[20] Michael. M. Mauldin. Chatterbots, tinymuds, and the turing test: Entering the loebner prize competition. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*. American Association for Artificial Intelligence, August 1994.

[21] Inc NetMind. Url-minder. URL: http://www.netmind.com/URL-minder/.

[22] Hyacinth S. Nwana. Software agents. *Knowledge Engineering Review*, 11, No.3:1–40, 1996.

[23] Tony Rutkowski. Internet trends. A collection of slides made available by Tony Rutkowski and General Magic. URL: http://www.genmagic.com/Internet/Trends.

[24] Beerud D. Sheth. A learning approach to personalized information filtering. Master's thesis, MIT Media Lab, January 1994. ftp://media-lab.media.mit.edu/pub/agents/interface-agents/news-filter.ps.Z.

[25] Surflogic. Surfbot. The Surfbot agent is available of download at this site. URL: http://www.surflogic.com.

[26] B. Venners. Under the hood: The architecture of aglets. *Java World*, 2, Issue.4, April 1997.

[27] P. Wayner. Free agents. *Byte*, March 1995.

[28] M. Wooldridge and N. R. Jennings. Intelligent agents: Theory and practice. *Knowledge Engineering Review*, 10, No. 2, June 1995.