Agent Based Personalized Information Retrieval

by

Joshua David Kramer

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degrees of

Master of Engineering in Computer Science and Engineering

and

Bachelor of Science in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 1997

Author ....
Department of Electrical Engineering and Computer Science
June 2, 1997

Certified by ....
Lynn Andrea Stein
Associate Professor
Thesis Supervisor

Accepted by ....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Agent Based Personalized Information Retrieval

by

Joshua David Kramer

Submitted to the Department of Electrical Engineering and Computer Science
on June 2, 1997, in partial fulfillment of the
requirements for the degrees of
Master of Engineering in Computer Science and Engineering
and
Bachelor of Science in Computer Science and Engineering

Abstract

There is no record being kept by computers about our personal information consumption. By capturing the content and context of all the information we interact with, computers can help us manage this information. This thesis describes such a tool, which we call a personal information management system. We build on traditional and personalized information retrieval tools, software agents, and the advanced multi-modal user interface provided by the MIT AI Lab's Intelligent Room.

Thesis Supervisor: Lynn Andrea Stein
Title: Associate Professor

Acknowledgments

I must thank Michael Coen first and foremost. For both his mentoring, advice, humor, and friendship as well as his amazing creation, SodaBot. I am honored that he let me tinker under its hood, and after 2 years, I think I finally understand how the heck it works, sorta. I can hardly wait to see HAL, which I know will be *slick* and will hopefully find your keys for you.

Professor Lynn Stein, thank you so much. I greatly appreciate the time and effort you spent to force me to figure out my own thesis and just what exactly I was trying to say.

Of course, I must mention the Intelligent Room folks. What an awesome playground. Special thanks to Kavita Thomas for her work on the speech system and teaching me about Dragon Dictate. And Brenton, without those `if` tests I'd be nowhere.

On a parellel note, equal thanks to the Haystack folks. I can't wait for the real thing to catch up with my attempts. Special thanks to Mark Asdoorian for the time he spent explaining Haystack internals to me.

How would I have done this without Ian, Tony, JJ, Jared, Dan, and the rest of Zeta Psi. Dan, tell Adam to give me my instruction book back now.

Oh Amy Jane – you keep me going and smiling all the time, you let me know what spring is like on Jupiter and Mars. **H*pp*=

No acknowledgments would be complete without mentioning our historical support by the Richard A. Kramer Foundation. Someday, your building will come.

I have been continually supported my Mom and Dad. Words cannot express my graditude and love.

This research is supported in part by the Advanced Research Projects Agency of the Department of Defense under contract number F30602–94–C–0204, monitored through Rome Laboratory and Griffiss Air Force Base; the National Science Foundation under Young Investigator Grant No. IRI-9357761 awarded to Professor Lynn Andrea Stein; and by the Mitsubishi Electric Research Laboratories. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis is about the fundamental need for computers to *manage* our information. It is about how to embed information retrieval into everything we do. The growth of networked computers has created an information overload problem. People are overwhelmed by both the supply of and demand for information. By expanding the scope and accessibility of current information management tools, new applications can be built which ease this information burden. In particular, not enough information is being kept about our personal information, the documents we read and write, yet this is precisely the type of information which is so often sought after. There are many solutions for managing much larger corpa, such as the set of World Wide Web documents or the set of Wall Street Journal articles; yet there are few tools which facilitate robust management of our personal information. This thesis describes our implementation of a prototypical personal information manager. Our solution involves combining software agents and an information retrieval system. We show how, in the right environment, this combination of technologies is a powerful tool for building a variety of new information enabling applications.

1.1 The Problem

The growing presence of networked computers is dramatically increasing the amount of information we generate and consume as well as our dependence on this information. Computers are quickly becoming a crucial component of many workplace functions. More and more paper forms are being converted to online equivalents. Indeed, the possibility of the paperless office is almost realistic; document scanners will soon outnumber document printers. With this growth of information accessibility will come both new applications and new problems. The growth and popularity of the Internet and World Wide Web (WWW) have provided a glimpse into this future. New technologies and applications are emerging which harness the power of these networks. Electronic banking and commerce, new forms of communication, and the ability for anyone to publish on the Web are just a few examples of these new services. However, this tremendous growth has created an entirely new sort of problem: there is too much information. Although all of this information is what ultimately empowers and enriches our online experiences, it can also overwhelm us and over-complicate simple tasks. How then can we gain control over this information? From the very start of the Web, at the 10,000 document level, indices were found to be useful. As the Web has grown to the level of 100 million documents, this has not changed.

Indeed, a whole new industry has emerged based on the need for tools capable of searching the Web. These Web sites use a variety of different information retrieval techniques to build what is essentially an index of all the documents they can find on the Web. Given the Web's unstructured nature and enormous size, these search engines have been crucial in making the Web usable. Without any sort of central authority or hierarchy, it is often quite difficult to say where a user should start looking for a document. Hence, Web sites such as Lycos[18] and Yahoo[19] each routinely get over 5 million search requests in a day[1]. Given the advertising and licensing revenues these sites generate, Internet search engines are here for the foreseeable future. So, while these tools and technologies have gained a handle on the vast wilderness of the Web, are our information problems solved? In the future will natural language queries be answered by powerful Internet search engines which will quickly scour the

[1]These sites are free services which allow users to perform keyword searches of the entire Web. They were both initially developed in academic environments, but became commercial sites due to the huge demand for their services. They generate revenue by displaying an advertisement to each user.

billions of documents available on the Web and return the list of the 10 documents which best answer our questions? Certainly this technology is excellent at answering questions such as, "What is the population of Ohio?" [2] The ability to quickly find answers to questions like this is one of the reasons the Internet is such a powerful resource.

But what about some other sorts of questions we might be interested in asking? Consider questions such as "What was that paper I read about robots last week?" and "Has Michael read it?" These questions do not want to search the entire Internet for a document. They require a more personalized data store. They represent an entirely new kind of information retrieval which we feel has been largely ignored to date. For the most part, this stems from the fact that computers fundamentally lack any real knowledge about their users. True, they do keep lists of frequent e-mail contacts, hotlists of favorite Web sites, schedules, financial information, and so on. But they don't ever use this knowledge on their own. While computers may have become a unified data store of all our personal information, they rarely make use of this information in any way beyond feeding it right back to the user who put it there in the first place. They don't analyze the data on their own, they don't share the data with each other in interesting ways, and, most importantly, they don't keep enough information about their users. Computers "know" nothing about their users' documents, correspondence, and actions; essentially, the information they create and consume. Put simply, computers rarely do anything with our data beyond what we tell them to do.

1.2 Information Retrieval Can Be Better

By keeping this in mind, we can leverage information retrieval to enhance our personal information interactions. Instead of an information retrieval (IR) system, we would

[2] In fact, one can currently go to the Infoseek[7] search engine, enter the query "What is the population of Ohio?" and get back as the best document the 1995 Ohio State of the Environment Report section on population[1]. This tells us that the 1990 census reports a population of 10,947,115 people.

like an information *management* (IM) system. We envision the following interaction with a fictional computerized personal information agent called Oren:

> **Lynn:** Oren, what information do I have about teaching Java.
>
> **Oren:** Just a moment while I check.
> *Pause. Then a list of documents with summaries appears.*
> **Oren:** There are 3 papers, 2 journal articles, and 13 Web sites you've already seen. You asked me to remind you about the 1st Web site. Also, Michael highly recommends the 1st paper and he has new 2 Web sites he recently saw which he liked. Finally, you have 37 email messages on the topic.
>
> **Lynn:** Who else in the Lab might be interested in this?
> *Pause. Then a new list of documents comes up.*
> **Oren:** Besides Michael, Gerry and Hal also have numerous documents about teaching Java.

While this may seem far fetched, it demonstrates our desire for more personal IM. In particular, while the natural language interaction is currently unrealistic, in Chapter 4 we describe a system that otherwise could do much of this. Our vision of the improved information management system includes four attributes:

- **Personalized:** The IM system should not be limited to the world of information available to everyone (i.e. the Internet). It should concern the personal documents of its user. In this respect the IM system will act like an augmented memory (or a very efficient assistant who can find everything[3]).

- **Ubiquitous:** The system should have access to a user's entire online experience. Simply put, using an IR system as an augmented memory is useless if it doesn't contain what you are looking for in the first place. In the extreme, such a system would quite literally see everything its user sees. In practice, we will see that what is needed is an architecture where it is possible for other programs to have access to information about the documents used and produced by a user.

[3]Or like your mother, who can **truly** find everything.

- **Automated:** The fact that this system is essentially watching over users' shoulders, capturing and indexing everything they do, should not really be apparent to users. This is crucial to the system's success. Users should not have to manually enter all of their documents into the system. Basically, that just isn't a realistic model – users will forget to enter some documents or simply grow lazy. The goal of the system should be to make information retrieval easier for users, not harder.

- **Networked:** Finally, by networking personal information management systems together, we create a new kind of web of information. This would allow users to query the information repositories of their colleagues. For searches relevant to their interests, this would yield a far richer set of documents than a similar search of the Internet. These types of searches make an excellent intermediate step between searching your own personal set of documents and searching the world of documents.

The question now becomes, how do we build such a system? Conventional IR systems are by no means user friendly; they typically concentrate more on the engine and less on the interface. Indeed, the online Web search tools are one of the few examples of genuinely useful deployments of information retrieval systems. Why is this? It is primarily due to the programmatic accessibility of documents on the Web. By programmatic accessibility, we mean the ability for programs to easily interact with documents without having to go through a user interface. In the case of the Web, this means that programs can fetch HTML documents directly. This allows Web search engines to effectively seek out and find every document on the Web. Thus, these systems are working with a well defined and very accessible set of documents. Furthermore, the hypertext nature of the Web allows for powerful searching techniques to be applied to this set.

A personal information management system will need a corresponding level of knowledge about a user's personal documents. Essentially the system will need a way to track what users are doing on their computers. This is no easy task. Furthermore, the system will need to provide users with useful interfaces to these documents. Thus, we propose a marriage of two technologies. We will use software agents to provide the tools for monitoring the user and generating autonomous and distributed applications. Information retrieval systems will provide these agents with powerful and easy access to large data stores. Together, these tools can meet our information needs in novel ways.

1.3 Thesis Overview

The remainder of this thesis describes a system which satisfies many of these desiderata: a personalized, ubiquitous, automated, and networked information management system. We use the Haystack information retrieval system as the underlying personal data repository for storing, annotating, and querying data. SodaBot software agents provide the tools for ubiquitous and automated information capturing. They also provide us with the connectivity tools for networking as well as the ability to integrate our system into the existing infrastructure of the Intelligent Room. This integration allows us to take advantage of the Intelligent Room's multimodal user interface to create an information retrieval interaction based on a spoken dialog between users and the Room.

This system is innovative in the following additional ways:

- It is integrated into the user's workspace. It provides a framework for capturing, indexing, and querying a user's personal information space. This is demonstrated specifically on Web pages, but is more generally applicable to a range of information sources.

- It supports a grammar for speech based interaction with an information retrieval system. This system allows users to verbally query their information repositories.

- It allows users to query each other's personal information spaces. This new source of information is normally not readily available for searching.

- It has been modified to provide a multimedia datastore for more than a human's personal information; it is the information handler for an interactive environment.

The next chapter provides background on this and other similar works, including more detail on the problem and approaches taken. Chapter 3 will discuss the specific projects on which this work builds: the Haystack per-user information environment, The Intelligent Room and its multimodal user interface, and the SodaBot software agent environment and construction system. We'll show how we integrate the these systems and demonstrate our applications in Chapter 4. In Chapter 5, we summarize our results and evaluate our work. Chapter 6 describes some directions for future work using these technologies and provides an overall summary.

Chapter 2

Background

There have been two primary motivations for this work. The first is our desire for a good personal information management system. We have become overloaded with information yet there are few tools for handling this at a personal level. The second is more concerned with our research in human computer interaction in the Intelligent Room at the MIT Artificial Intelligence Lab. We hope to provide the project with a powerful new tool for information storage and retrieval. The Room is an excellent example of a system which is empowered by information, yet does not have adequate means for representing it.

2.1 Personalized Information Management

In Section 1.2, we expressed our desire for an information management system which was personal, ubiquitous, automated, and networked. In many ways we already have IR systems which are ubiquitous for a specific domain, automated, and networked – Web search engines. Systems such as Lycos and AltaVista automatically spider[31] the Web, attempting to find every document on the Web and index it. Furthermore, systems such as SavvySearch[9] and the MetaCrawler[40] send queries to multiple Web search engines and aggregate the results to produce a better result set than any individual engine does. So, what is it we mean by personalized information retrieval, and why is it useful?

The key to a personalized information retrieval system is that it should contain all those and only those documents which its user has already seen. This is very different from conventional IR systems where the user is normally searching a large unknown body of documents. Karger and Stein[22] liken this to searching one's *bookshelf* versus searching a card catalog at a library. Furthermore, the fact that users have most likely already seen the document they are looking for enables more powerful queries. For example the user might be able to place temporal constraints on the query, limiting the search to only those documents placed in the archive during a certain time period. However, even for normal queries, the quality of the documents returned is likely to be far superior to those returned in a search of the global document space. After all, one does not normally keep bad books on one's bookshelf. The key idea is that the average document we read tends to be, from our perspective, of a higher quality and more relevant than the average document on the Web.

Furthermore, by providing facilities to annotate documents, both *actively* and *passively*, we can further increase a given document's perceived utility. By active annotation we mean giving users the ability to annotate digital documents about their content or relevance. Taking this one step further, we can let our computers annotate our documents with meta-information about how we are using them, as in Hill et al.[16]. This passive annotation requires no explicit action by users and can provide new metrics for evaluating a document's worth. For example, our system automatically annotates each Web page we see with the current time. We can then use this information to constrain future queries. By keeping closer tabs on what we actually do with our documents, computers will be all the more powerful in trying to help us find them at a later date.

Such a system is really more than just a bookshelf. In the idealized case, it is a copy of everything we've ever read, seen, or heard, fully annotated with our thoughts about them. Its purpose is to augment our own all too frail memories. Moreover, unlike our memories, certain portions of this bookshelf are open to the public. This is yet another alternative to searching the entire Internet for a document. Before going to the library, why not check your neighbor's bookshelves? Furthermore, this presents an exciting new opportunity for resource discovery – someone with a highly annotated set of documents about computer architecture is probably an excellent person to go to for assistance in computer architecture[1]. Also, users might often have their questions answered by documents in another person's personal IM system, and thus not have to directly disturb the other person.

2.2 Role of the Environment

In designing a personal IM system, it becomes clear that the application will need access to a variety of subsystems. We essentially want our computers to be aware of what we are doing on them. For instance, the personal IM system will want to "know" about each document its user accesses. Most applications, and indeed most operating systems, do not provide this level of service. While several standards for exchanging information between applications (CORBA[46], Microsoft's OLE, Apple's OpenDoc, UNIX IPC, etc.) exist, they are certainly not in widespread use. The simple problem is that there is no single universal interface for applications to communicate with each other about their activities. What is missing is an common infrastructure to support these activities.

To examine this problem in more depth, let us continue our example. How could our personal IM system know which documents we are reading? In general most programs do not provide a means for other programs to find out what they are doing. When a document is opened in the emacs text editor, how is the IM system told about this? One could imagine re-tooling every application we use to provide such an interface. Indeed, this is not very difficult with a program such as emacs, which is designed to be highly extensible, and whose source code is freely available for modification. But what about when we open a document in Microsoft Word, or view a postscript file, or simply print a postscript file for off-line reading? Another technique would be to monitor a user's actions at a very low level. For instance, the file system

[1] Of course one could easily fill up a bookshelf with books about computer architecture and annotate each with "I didn't understand this one either." But that would be silly. At the very least it would still be a collection of documents on Computer Architecture.

could be patched to notify the IM system whenever a file is accessed. Again, this works fine in an open environment such as Linux, but what about Windows 95? And what about networking these systems together – this presupposes both the ability to easily make use of the network as well as the ability to track groups of users. All together, we are looking for a very complex infrastructure, quite different from the typical computing environment found today.

2.2.1 The Intelligent Room

However, we have chosen to work in an entirely different environment; one which is designed to try and understand what its users are doing in it. This is an environment built around a very powerful and well defined framework of software agents. It is an environment which is not only aware of what documents are being used in it, but hopefully one which has an idea of its occupants' intentions. This environment is the Intelligent Room[45] at the MIT Artificial Intelligence Lab and its system of SodaBot software agents. The Intelligent Room is an experiment in enhanced human computer interaction. The Room is designed as an intelligent conference space void of keyboards and mice, yet fully interactive. The project's goal is to aid the users of the Room in tasks such as information retrieval, teleconferencing, or strategy planning. To this end, the Room tries to interpret and contribute to the activities occurring in it.

The Room uses cameras as its eyes and microphones as its ears. The microphones are attached to real-time speech recognition systems. The Room also contains a speech synthesis system which allows it to interact with its occupants in dialogs. Several output displays are used to show media such as Web pages, videos, camera views, or arbitrary applications. The Room uses a variety of vision systems such as a multiple-person tracker, a finger pointing detection system, and a laser-pointer detection system. These systems provide information about where a person is and where they are pointing. The pointing system, for example, allows users to control a virtual mouse pointer projected onto a wall display. By bringing the computer out into the human's world, we hope to enhance our interactions with it and to use it to enhance our interactions with each other. This thesis is very much in the spirit of both these goals.

In fact the Room's architecture is designed to provide the type of information we desire. The Room is interested in knowing **who** does **what**, **where**, and **when**. There are many subsystems each of which can answer part of these questions. The Room's software agents tie this information together to produce a coherent view of the activities occurring in it. Furthermore, they provide this view to any other agents which are interested in it. For example, when a user standing in front of one of the Room's wall displays says "show the map *here*," agents need to resolve where exactly "here" is referring to. In this particular context, the tracking system is used to locate the speaker in the Room to determine which display to use. Thus, the Room's architecture typically contains agents which know who is in the Room, what they are saying (for a limited grammar), what information they are looking at, and some sense of what they are doing with this information. By working in this environment, our personal information management system is able to make use of these existing agents.

2.2.2 The Room as a Testbed

The Intelligent Room has proven to be not only an excellent testbed for new ideas, but also an inspiration for many of these same ideas. Coen[5] has made the point that the Intelligent Room is more than a platform for HCI experiments; it is an excellent environment for conducting core AI research as well. Once we have an environment in which rich interactions with the computer are possible, a whole new set of applications will emerge. The key is to develop a stable platform on which we can build these new applications. Just as the de facto standardization of Microsoft Windows by corporate America brought about an explosive growth in software development, we hope that the Intelligent Room will prove to be the launch pad of many new types of applications. Indeed, in many ways the Room has developed its own new operating system. Much of this is due to the use of the SodaBot software agent system, which is discussed in greater depth in Section 3.1. Essentially, the use of agents as a ubiquitous middleware has provided the Room with a strong framework for building higher level applications.

Many of the benefits of working in this environment are derived from embedding computers more firmly in their surroundings. The Room has systems for tracking and identifying its occupants, for recognizing what they are saying, and for determining where they are pointing. These systems are accessible to other higher level applications running in the Room. This provides these programs with both powerful new means for interacting with their users and a wealth of information about what their users are doing. By giving applications more access to their operating environments they can provide more context sensitive information to their users. Enhanced information management is a natural application for this environment. Imagine annotating documents with who was in the Room and what was being discussed at the time they were displayed. This is in fact largely what we set out to accomplish. We have leveraged this powerful operating environment by building our personal information management system in it.

Providing the Intelligent Room with access to its own information management system is a major contribution of this work. While the Room has an existing interface for using the START[23] natural language information retrieval system, it is very limited in its use. All the information in START has to be entered by hand and is generally high level symbolic information, such as statistical summaries. Thus its data set is quite limited. Since the Room cannot directly add new information to the system, we only use it for searching for information we already suspect is in the system. Our work has given the Room a system for managing the many different sources of information it has at its disposal. It has given the Room the capability to search a datastore of documents which it maintains. One of the primary goals of the Intelligent Room project has been to make the Room more pro-active in providing information. Instead of users constantly initiating information searches themselves, the Room should offer relevant information whenever it is appropriate. For instance, if two occupants are discussing improving the Room's multi-person tracking system, the Room might bring up a list of relevant documents, such as the design specification of the current tracking system.[2] If we can give the Room's systems access to more information, then they can provide more information to its occupants. Creating an integrated information management system for the Room has done just this.

2.2.3 The Room as a User Interface

From a purely human computer interaction standpoint, the Intelligent Room provides a rich multimodal environment. The physical layout consists of two wall projected images which act as the Room's primary output displays. There is a video multiplexer which can route any of a variety of signals to these displays including the output of three different SGI workstations, two VCRs, and nine camera views of the Room. Typically, the main display shows a Netscape Web browser from one of the SGIs. Occupants can point to specific locations on the displays using either their fingers or a hand-held laser pointer. Furthermore, their locations and trajectories are watched using a vision-based tracking system. Occupants wear wireless microphones connected to a speech recognition system which allows them to "talk" to the Room. Conversely, the Room can "talk" back via a speech synthesis system. When necessary, users can always fall back to a standard mouse and keyboard.

Thus, there are several means available for any information exchange between the Room and its occupants. For example, there are many ways to follow a hypertext link in a Web browser being displayed on the wall. Users can point to the link, with either their fingers or a hand-held laser pointer, and verbally command the room to "click here." Alternatively, they can explicitly name the text of the link, as in "follow the 'MIT Home Page' link."

The Room uses a novel speech recognition control system to provide a rich grammar for most interactions. This system allows the speech recognition system to dynamically change its grammar based on the perceived context. For example, when an agent in the Room asks a "yes-or-no" question, the Room can load a subset of its

[2] In a simpler scenario, when a user in the Room is browsing the Web and goes to the Web page which describes the Room's tracking system, the Room offers to play a video clip about how the tracking system works.

grammar which allows dozens of ways of replying to these types of questions ("yeah, sure, nope, OK, please do, yes, go ahead," etc.). This is a complex, expressive, and not very limiting set of responses. However, the output of the speech system which the agent gets is a simple "yes" or "no." This allows the Room to support complex input speech grammars with having to support a corresponding complex set of grammars for processing what the speech system outputs. That is, the agents do not have to worry about synonyms or equivalence classes. At the same time, humans do not have to worry about memorizing a limited set of commands. This support for *natural* interactions is what makes the Room such a novel user interface.

2.3 Enabling Technologies

Before we describe our specific implementation technologies, it is useful to abstractly describe what exactly software agents and information retrieval systems are to us. This will provide some basis for evaluating SodaBot and Haystack. It also builds some background for looking at the other systems described in Section 2.4

2.3.1 Agents

Software agents are a recent class of computer programs which have been given many definitions. Qualities ascribed to software agents by Etzioni and Weld[11] include being "autonomous, goal-oriented, collaborative, flexible, self-starting, communicative, adaptive, mobile, temporally continuous, and having character." This grab-bag of features describes both a very generic tool as well as a very powerful one. Indeed, Stein[43] goes as far as to suggest that there isn't a viable definition of agency at all, that it is simply our individual perceptions of what a program is doing for us which make it an agent. Agents are what we make of them.

For us, software agents are an important part of a solution to our problems of information management. They can act as an intermediary, or buffer, between ourselves and the information we desire. Moreover, they provide the means by which our computers can take a more active role in serving our information needs. It will be software agents that take care of sorting through all the data returned by simple queries, which automatically catalog our personal stores of information, which keep track of what we do with our computers, and which initiate contact with agents belonging to other people, or entities, in search of the answers to our questions. The act of "initiating" searches or other activities on our behalf is what makes software agents such a compelling technology for us. Software agents really are many things for us. They are intelligent middleware – providing the glue which interconnects the various components of our system. Yet they are an *active* glue, able to take actions on our behalf, such as initiating searches. Software agents let us build light-weight abstract wrappers around complex systems which in turn allows us to create higher level applications.

2.3.2 Information Retrieval

One way of looking at agents is to say that agents are empowered by information. They analyze, transport, filter, display, and create information. So the question is, how can they best make use of this data? Specifically, what should be their interface to information? By this we mean, how should agents best internally store and represent the large quantities of information we want them to handle. For this, we turn to Information Retrieval (IR) systems.

Traditional IR systems essentially analyze data (usually some kind of document, for example an e-mail message) and place it in some internal index. They then provide an interface for querying their indices and returning what they decide are the most "relevant" documents. A basic IR system, for example, would simply return all documents which contained any word in a given query[3]. More advanced systems make use of word frequencies, semantic structure, or complex query specification languages. Essentially, IR systems provide a means for accessing large stores of information through the use of simple, often unstructured, queries. This is important because it replaces the need for agents to maintain structured representations for all their

[3]In the extreme this could be a simple shell script which used UNIX's `grep -v` on a set of files.

data. By using an IR system to both store and index data, agents gain access to a tremendous knowledgebase of information with a very simple query interface.

We feel that a crucial component of any IR system should be programmatic access to the system. We will only begin to see fundamental improvements in information management applications when we allow other programs to query, parse the results, and generally make use of multiple IR systems for us. By replacing the traditional human interface with an API, we enable a whole new level of information retrieval. So, for us, information retrieval systems are black boxes – the datastore for our documents. By all means, they are certainly a powerful abstraction. But, we make no attempt to explore or improve their internal design. We refer the reader to Salton's[39] excellent overview of this area for a detailed explanation of IR. Thus, the choice of which IR system to use as our underlying engine is not terrible important, and given our choice of Haystack, actually not our choice to make. For, while Haystack is nominally the IR system in our design, it is itself really just an application built around its own pure IR engine.

2.4 Related Work

There are many earlier works which address portions of the personal information overload problem. They too typically make use of an IR engine to enhance higher level applications. Many of these systems are what are called Information Filters – they are designed to act as a firewall between their users and a stream of incoming information such as email or the Web. Very few have dealt with the issues of building an integrated personalized information management system. However, given the multi-faceted nature of our work, these systems are fertile ground for inspiration and comparison. We divide the works into four categories. These are pure filtering applications, collaborative filtering, active agents, and personal IR systems.

2.4.1 News, Email, and Web Filters

One of the great inspirations for information filtering is Usenet Net News, the Internet's version of a bulletin board. News filtering applications have attempted to deal with the fact that with over 30 million Internet users, news groups are overwhelmed with postings. Yet, for given user only a very small percentage of these articles are interesting and relevant. These applications typically develop an internal profile of their user and then try rank news articles according to this profile. The profile is generally used as an input to a IR engine which contains the set of news articles. Thus, each person can use their filtering tool to generate an individual view of the same data. For example, when reading rec.pets, one person might only be interested in articles about dogs, while another only those about cats. Many of these systems have been modified to support Web and email filtering as well. Email filtering is notable because mail tends to arrive sporadically, and thus individual messages must be evaluated on their own, not necessarily against a larger set. This information *filtering* is slightly different from the information *retrieval* we are interested in. While our work does not attempt to provide *filtering* capabilities, the architecture we deploy both supports and compliments filtering.

The Stanford Information Filtering Tool (SIFT)[50] is an representative example of the simplest implementations of a news filter. It uses a series of weighted keywords provided by the user as a profile. These keywords are then given to a customized IR system which uses a vector space model to find matching news articles which are "close" in vector space. Users can provide relevance feedback, which the system uses to improve the users' profiles. In SIFT, users can simply say they "liked" an article; there is no opportunity for negative feedback.

Relevance feedback is an area of research in and of itself. Essentially, queries can be improved by making use of keywords from good documents and removing the keywords from bad ones. Thus, a system such as NewsWeeder[26] has users rate each article they read using a scale of 1 to 5 (They point out the importance of making the rating process as simple and unintrusive as possible for the user.) This type of feedback can greatly improve a search, even with just a single iteration[39].

Our framework enables a slightly different sort of feedback. We are able to make use of passive observations to determine how useful a document is to the user. While this may not help an individual search, the goal is to build up information of time about which documents best answer which queries.

Browse[21] is an X windows based system which uses a neural-net to model its users. It uses a binary rating system by asking users to either accept or reject articles. This is not too different from SIFT. However, Browse also supports ad hoc queries of news and attempts to use its model to help answer the question. This is a step in the right direction. It is one of the few examples of a tool which uses a model of its user when answering queries. It is also a direction which we hope our own system will take. In some sense we already do this by modeling users through their documents.

Letizia[28], WebWatcher[2], and Syskill & Webert[35] all take a personalized approach to managing the wealth of information on the Web. While Web search engines have provided a valuable tool for finding sites in general, there is often a lack of local searching which makes navigating some Web sites difficult. These systems "look ahead" of the user and attempt to determine which links on a given page are most likely to be interesting to the user. The first two work by observing the types of pages the user goes to and learning a profile of the user. Syskill & Webert has the user first rank a series of web pages. It then builds a model based on the content of the ranked pages. Our system is poised to easily support the functionality provided by these tools. We already have access to the Web pages a user browses, and in fact, we keep them in an information retrieval system. So, an crude implementation of this type of tool would be to take the text of each link on a given page and see if the user has a large number of matching documents in their personal information repository. We could then highlight those links which seemed interesting by this criteria.

2.4.2 Collaborative Filtering

While developing a profile of an individual user's interests can be a powerful step in improving their information management, it is often not enough. However, by looking at the opinions of multiple users, we can provide a more detailed analysis of a given document. The combination of opinions with textual analysis provides a much better feel for a document's relative worth. If we know that we are interested in "robotics," that knowledge enables us not only to pick out documents about robotics, but to select *good* documents about robotics. This is an area we are eager to explore. As we build networked personal information management systems, it will be possible to use this collaborative network to enhance the information retrieval process.

The Tapestry[44] project at Xerox PARC is an excellent implementation of this idea. The Tapestry project focuses on annotating and filtering electronic mail and recent articles on net-news bulletin boards for a fairly small community of users. Several Usenet news reading tools[29] were modified to let readers vote on articles they like and dislike, as well as to annotate these documents with their comments. The votes are collected, tallied, and distributed using the basic Usenet transport mechanisms (nntp), and readers can view articles based on their total number of votes and who voted for them. This allows for some innovative filtering called Virtual News Groups. A user can essentially choose to see a group as moderated by another user. Another observation is that documents can become more interesting over time (i.e. as they gets better reviews). One problem, however, is the system's use of a rather complex query language, TQL, the Tapestry Query Language. The language allows users to specify filtering rules which are applied to the set of messages. One novel idea they describe is to first use binary *acceptors* to quickly filter articles and then rate the articles with *appraisers*. The GroupLens[36] project has similar goals but is built on a very open framework.

The Agents group of the MIT Media Lab has developed a number of collaborative filtering applications. Maxim[27] is a learning collaborative email reader. It is noteworthy for being exceptionally easy to use and thus much more geared toward beginners than Tapestry. FireFly[41] is a collaborative system for recommending new music selections. It uses a nearest neighbor approach to find other users in the system with similar likes. The underlying engine has been adapted for the Web in the form of WebDoggie[32]. An even more general approach is taken in Yenta[12] which attempts to play the role of match-maker. Yenta attempts to dynamically create interest groups by introducing people with similar interests who have never met. Yenta requires users to explicitly list their interests. We have a similar idea for using our networked personal information systems to create an *expert* finder. We envision searching our neighbors personal information systems to find the ones with the most relevant set of documents. Kautz et al.[24] have created a such system to automate the process of finding an expert in a given field. They use "referral-chains" of people with similar expertise to locate the nearest expert to a searcher. Like our own idea, they use a set of personal documents to determine an individual's interests.

2.4.3 Agents

There are a growing number of information agents being deployed. This includes systems on the Web, personal information agents, and "expert" systems for searching specific knowledge bases. The tendency seems to build an agent which can make use of several resources (often Web based) and tie them together in useful and easy to use ways. The University of Washington's collection of Softbots[10] are an example of this approach. They have build up systems for locating users, shopping assistance, travel reservations, and searching the web itself. These are exactly the types of applications which agents are excel at. They make use of a variety of computational resources to either solve high level problems or solve problems more efficiently.

The University of Chicago's FaqFinder[15] is a system for searching the set of FAQs (Frequently Asked Questions) associated with many Usenet news groups. These documents typically are a list of questions and their answers. For a given news group, they can be very informative. FaqFinder takes a natural language query and uses the SMART[38] IR system to find relevant FAQs. It then parses the query further and attempts to find a matching question/answer pair. The START[23] natural language information retrieval system provides similar functionality for its own limited knowledgebase about Bosnia. Unfortunately, both systems are quite limited in their knowledge and tend to have poor failure modes for dealing with queries they have no answer for. Thus, they are only really useful when you already suspect they have the answer to your question.

Sheth[42] describes Newt, a news reader which makes use of genetic algorithms to develop multiple user profiles. These profiles then compete to yield the best set of keywords to search the news corpus. This work is notable because it describes a generic framework for using genetic algorithms to deploy information agents. The information agents then compete to provide their owner with the best information, regardless of the source. Amalthaea[34] is an attempt to bring a large number of these information agents together. The goal is to collectively meet a user's information needs instead of building a large, complex, information system.

2.4.4 Personal Information Retrieval Systems

The Remembrance Agent[37] is the project with the most similar goals to our own – augmenting human memory. The system attempts to monitor every document its owner uses and add it to an information retrieval system. By watching windows of the user's last few keystrokes in the emacs text editor, the system attempts to find documents in its repository which best match the current context and suggests these to the users in a side buffer. Its constant stream of suggestions is unique in the field of personal information retrieval. However its current implementation has limited means for automatically adding documents to the repository. Furthermore, it is limited to textual documents saved in the user's file system. It also lacks the networking and annotating capabilities of our system.

Overall, there are relatively few systems for personal information management. AltaVista [17] has introduced a *personal* version of its popular Web search engine. The program is designed to index a user's hard disk and then provide text based querying. Glimpse[30] is designed to provide keyword searches of a user's file-system. It is essentially a more powerful version of UNIX's grep, which itself has been used as a crude tool for searching personal files for certain words. In fact, even operating system vendor Microsoft now supports a function for finding all files which contain a given text string. However, none of these tools provide the ability to annotate documents, either actively or passively. Furthermore, none of these tools are designed to work in a collaborative networked environment. Finally, they are not very well integrated into a user's desktop, watching everything, including documents such as transient email or Web pages in a browser, which do not get saved to disk.

Of course, we must mention the Haystack project itself. As we describe in Section 3.2, its goals are very much in line with and are the inspiration for many of our own.

Chapter 3

The Enabling Technologies

Our goal is an autonoumous ubiquitous networked personal information managment system. By providing software agents with an interface to an information retrieval system, we can leverage the active nature of the agents to provide an intelligent interface between humans and information. Moreover, by allowing software agents to easily maintain large stores of data in an IR system, we can simplify the creation (and growth) of new types of databases. We rely on our agent system to provide connectivity and logical control and on our IR system to provide a manageable and annotatable data store. Specifically, we have chosen the SodaBot software agent environment and the Haystack information retrieval system.

Before we begin a detailed look at these two systems, it is important to note that this work makes a claim beyond and our central point is independant of their particulars. It is true that our selection of SodaBot and Haystack is based on the lab in which this work was conducted. The techniques of combining agents with information retrieval are certainly generalizable. What we mean is that agents provide us with a tool which enable a more *proactive* use of information retrieval technologies. Currently most information retrieval applications involve users explictly requesting information. We see agents as autonomously requesting information for us, as the Rememberance Agent does. As far as our use of Sodabot and Haystack is concerned, we shall see that the SodaBot interface to Haystack is a well defined application program interface (API) – one which many other IR systems could be modified to provide. Given our black-box view of IR systems this is not too surprising. We must note, though, that Haystack's support for document annotation is not a common feature of most IR systems.

Is SodaBot necessary? Well, it is certainly required for our interactions with the Intelligent Room's existing infrastructure of agents. However, there are several other agent systems which provide similar levels of network accessibility and easy user interaction. AgentTCL[14], and to some extent TeleScript[48], could be used to build a similar infrastructure. Given the functionality gained by building our application in the context of Intelligent Room, SodaBot is the logical choice.

3.1 SodaBot

SodaBot[6] is a software agent environment developed at the MIT Artificial Intelligence Lab. It is designed to simplify the specification of agent interactions. SodaBot consists of both an agent programming language, SodaBotL, and a runtime environment to support these agents. One of SodaBot's major strengths is the ease with which it can programmatically make use of new resources made available to it[1]. For example, SodaBot provides an *expect* mechanism which allows SodaBot agents to interact in a scripted manner with any application which provides a textual interface. In developing SodaBot based applications, one often provides access to a given service – a software device, such as a web browser, or something more physical which has a software interface, such as a VCR – by encapsulating the service within an agent. Other agents in the system can then make use of the service without knowing the specific details of its direct interface. They can simply use a well defined and abstract set of functions provided by an agent. In a sense, SodaBot allows quick and simple deployment of network services for applications which themselves are not inherently networked based. SodaBot abstracts away the distributed networked nature of these services and allows agents to concentrate on the content of their actions, not the exe-

[1] By programmatically we mean that agents are able to make use of other software and hardware devices without having to use traditional user interfaces such as a GUI.

cution. Thus only the *VCR* agent need know the complex set of serial-line instructions for controlling a VCR; other agents can simply use the `vcr.play` command.

SodaBot consists of two distinct components. The agent programming language, SodaBotL, is used to code agents. Its syntax is similar to Perl[47], but SodaBotL has several additions which make it suitable for agent programming. These additions consist of language primatives which make it simple to specify agent interactions without having worry about the details of the execution. It is comparable to having a built in set of routines for fault tolerant network access and RPC[3]. In some respects SodaBotL is similar to an object oriented language, except that instead of objects, it has agents. We mean this in the simplest possible interpretation: Agents make use of other agents, and there is no code which is not part of an agent. Simple procedure calls are of the form *AgentName.FunctionName(Arguments)*. A typical procedure call might look like:

```
VCR.Play('The Cog Clip');
```

In this case we are calling the `Play` request[2] of the *VCR* agent with the string `'The Cog Clip'` as the one argument.

The Agent Platform

The second part of the SodaBot system is its runtime environment, the Agent Platform (AP)[3]. The AP is essentially the operating system for SodaBot agents. Every agent runs on an AP, with multiple agents allowed to run on the same AP. If an agent wants to run on a specific machine, there must first be an AP running there. Each AP has a name, which defaults to the machine name it is running on. When an AP starts, it registers with a *site server*, a special agent which provides agents with name resolution. These names, which look like Internet email addresses, are of the form `ap_name@domain_name`. So, an AP running on a machine named `lovebug` in the

[2] A request is analogous to a C++ method, a procedure which other agents can call. We use the term request synonymously with procedure, function, routine, and method.

[3] In the original implementation of SodaBot[6], these were known as Basic Software Agents, or BSAs.

AI Lab is canonically known as `lovebug@ai.mit.edu`. Agents can specify which AP they want other agents to run on when making procedure calls. So, the same call to the *VCR* agent could look slightly different if the agents were running on different APs:

```
VCR.Play('The Cog Clip')[lovebug@ai.mit.edu];
```

Here, we are using the same agent, request, and argument as above. But in this case, by appending `[lovebug@ai.mit.edu]` to the call, the request will execute on the `lovebug@ai.mit.edu` AP. Thus, each agent is uniquely identified by its name and the name of the AP it is running on. In the above call, the agent is named *VCR* and the AP is `lovebug@ai.mit.edu`.

However, SodaBot also allows agents to register aliases for their APs. For instance, the *VCR* agent can register an alias of `vcr@hci_room` for its AP. This makes its actual location transparent to other machines, since they now use the alias in their procedure calls:

```
VCR.Play('The Cog Clip')[vcr@hci_room];
```

As we shall see, this will prove useful for representing different users. The important point about the AP and its naming scheme is that it can remove individual machine dependancies on where agents are running.

3.1.1 SodaBot in the Intelligent Room

SodaBot is currently used as the primary control system for The Intelligent Room. It provides the *computational glue* which seamlessly integrates the various components of the Room by providing both information and control pathways between agents. Each system in the room is represented by an agent. For example, there is a *Netscape* agent for controlling the Netscape web browser and there is a *VCR* agent for controlling the Room's two VCRs. There are roughly 20 agents distributed over 10 different workstations with no centralized thread of control. We call this base level control system the *ScatterBrain*[5].

The *ScatterBrain* represents what are essentially the Room's reflexes - a set of basic behaviors that are constantly available. For instance, the *LaserPointer* agent is always running. It scans one of the displays for the location of the characteristic signature of a hand-held laser pointer and updates the mouse pointer on the display to correspond to that location. The agent also recognizes *click* events and passes them on to the display. This system, once started, is completely autonomous from the rest of the agents running in the Room. The design of the Room's architecture calls for higher level, more complex, applications (such as an information retrieval agent) to be built on top of the base "reflex" layer. These layers of many agents interacting in interesting ways was largely influence by Brooks' subsumption architecture[4] and Minsky's Society of Mind[33]. Coen[5] calls this 2nd tier "intermediate information-level applications."

An example of such a higher level system is a system for controlling a slide presentation. The *Slide* agent is essentially layered on top of the *Netscape* agent. It allows a user to define a series of web pages and provides speech control for sequential and random access. By layering it on the *Netscape* agent, the *Slide* agent can use several of the Room's default behaviors, such as the *LaserPointer* agent's ability to send mouse click events to the browser, or a standard vocabulary for browser control which the *Netscape* agent sets up in the *SpeechIn* agent. Making every component of the Intelligent Room accessible to the common middleware of SodaBot provides a solid framework upon which to build new high level applications. The *ScatterBrain* provides the ability to easily build these high level and more complex applications in a multi-modal interactive environment. It is also a natural level at which to integrate our personal IM system into the Room.

3.2 Haystack: Per-User Information Retrieval

The Haystack[22] information retrieval system is an project of the MIT Laboratory for Computer Science. Its goal, much like our own, is to be a "living" personal information retrieval system deployed on a community wide basis. Indeed, much of this work can be viewed as an experimental implementation of many of their goals. Haystack is more than an information retrieval system however. In fact, it is designed to be layered on top of more conventional IR systems. The current implementation is essentially a complex set of wrappers built primarily around the MG[49] textual IR system. Haystack is trying to provide a usable interface to IR by layering itself between the complex and difficult IR system and its users. For us, Haystack will provide the tools for managing our documents. However, Haystack also in many ways provides us with our inspriration. Thus, it is important to describe all of the project's goals before describing their current progress towards these goals.

3.2.1 A Bookshelf

Like the mythical information managment system of Section 1.2, Haystack very much wants to be a bookshelf for its users. To this end, the project has worked on a number of interfaces for getting all of a user's documents into their Haystack. They have created a simple Web based interface to Haystack which allows a user to browse their file system and select documents to add to their Haystack. This same interface supports a simple query syntax for searching Haystack. Preliminary work had been done to create some application specific archiving agents. For example, the emacs *RMAIL* mode has been modified to let users archive their mail as they read it. There is also a *directory walker* program which, much like a web spider, goes through users' home directories and archives their documents. However, the goal remains the same – archive whatever the users consume, be it news, email, papers, arbitrary documents, web pages, or scanned in documents.

3.2.2 Content Aware

One important attribute of Haystack is that it is a *content aware* system. Haystack attempts to determine the "type" of each document as it places it in the archive. This is a very powerful feature. For instance, it would normally be useless to archive a postscript document in a text based IR system, since the raw text of the file is completely illegible. Haystack solves this problem through the use of *textifiers* – programs which extract the text from non-textual documents such as postscript, HTML, or even by performing OCR on scanned documents. This allows Haystack to index documents which other IR systems simply cannot. Once Haystack has determined the type of a document, it can make use of that knowledge to perform additional information extraction. Through the use of content-type specific *field-finders*, Haystack can extract meta-data about the document. For example, when archiving an HTML document, Haystack can search the `<HEAD>` tag for information such as the document's author, creation date, or title. Furthermore, the system is extensible, allowing the easy addition of new content types as necessary.

3.2.3 Annotatable

Haystack not only provides the standard query and archiving mechanisms common to most IR systems, but it also contains a facility for annotating documents. These annotations come in the form of searchable user defined description fields for each document. This feature, for us, is perhaps its greatest strength. Users (or their agents) can associate each document with a set of arbitrary field name/value pairs. Haystack allows users to annotate their documents much as they would scribble notes in the margin of their physical texts. For example, a document might have an annotation called "`UserRating`" with a field value of "`8 out of 10`."

Documents can be annotated with much more than symbolic information about their content. Imagine attaching usage histories to every document, when was it read, how long was spent reading it, who it has been forwarded to, or who else has read it. This type of *meta-data* enhances both the ability to specify queries as well as the organize their results. Haystack currently automatically generates a limited set of annotations for each document, including guesses about author and title for certain types of messages. This is the feature of Haystack which makes it not only our inspiration, but also our implementation technology. There are few IR systems which support annotation. One of the main contributions of our architecture is the ability to easily automatically generate different types of annotations about a document's usage. Haystack provides us with a logical location to store these annotations.

3.2.4 User Profiling

One of the goals of the Haystack project is to enhance a given user's IR experience by using their query history as a guide for future queries. So if a given user typically makes queries about "software agents," further queries about "agents" won't rank documents about "travel agents" as highly as those about "intelligent agents." This can be used both for query construction as well as post query analysis (i.e. in what order to list the returned documents). This goal is inspired by the fact that many IR interactions are currently a very iterative process. A user makes an initial query, and then they keep refining it, by specifying more query terms or relations between them, until they find a satisfactory document. Often these refinements are simply a process of providing the IR system with additional context about the user which it could have retained from previous interactions. A goal of the Haystack project is to learn these addition contexts about a user and use them to aid future queries.

3.2.5 Multi-User

One of the primary goals of the Haystack system is to support multiple users and cross-user queries. Allowing users to search each other's Haystacks creates an entirely new type of information retrieval. They feel that this is an excellent area to apply learning techniques. For example, a user's Haystack could learn which other users typically provide good answers for certain types of queries. This can be extended to create an "expert finder" – by finding other users with Haystacks which contain many documents on a single topic. The argument has been made that searching other people's Haystacks is no better than effectively searching the entire Web[20]. We argue that Haystacks should typically contain documents which cannot be found on the Web or which cannot be indexed by conventional Web search tools. For example portions of email, scanned documents, annotated video segments, and simply personal documents which a user has not taken the time to explicitly make Web accessible will turn up in users' Haystacks. Moreover, as users annotate their documents, these types of searches yield a richer, and likely more relevant, result set. This is a set of information which currently just does not exist in any searchable format.

3.2.6 Haystack: The Current Reality

Unfortunately, many of the higher level goals of the Haystack project have not been implemented yet. To date, the project has built a subsystem for performing information retrieval on a variety of underlying information retrieval engines. While the primary underlying engine has been MG, experimental versions have supported using both the Savant IR system from the MIT Media Lab as well as `grep`. The current implementation provides basic indexing, querying, and some automatic annotation functionality with a fairly primitive web based interface. This base level of functionality is necessary before Haystack can tackle some of its more ambitious goals such as user profiling or multi-user support. Indeed, much of the work in this thesis can be conceived as a test, or even a validation, of some of these higher level goals. However, even the base level existing Haystack system provides us with a viable tool for enhanced information retrieval. What was missing, however, was a programatic means of using the system that our software agents could interface with. This need led to the development of the Haystack API.

Chapter 4

Putting Agents and IR Together

In order to achieve our goal of a personal information management system, we need to develop a number of subsystems. These subsystems will link up our enabling technologies, SodaBot and Haystack, in the context of the Intelligent Room. We begin by describing the Haystack API, a clean and stable interface to Haystack designed to be used by SodaBot. We then present a series of low level support SodaBot agents which our personal IM system will make use of. These are agents for using Haystack, manipulating documents, and handling times. This leads to the *User* agent – the heart of our personal IM tool. We will describe the implementation details of the *User* agent and give some examples of how humans and other agents make use of it. The highlight of these interactions is the ability to perform spoken language information retrieval. We further show how to extend this core set of agents to provide the Intelligent Room with a multi-media datastore.

4.1 The Haystack Application Program Interface

Before we can write SodaBot agents which use Haystack, we need a well defined interface for using Haystack. The existing Haystack system provides a series of command line functions for accessing its functionality. These are mainly used by Haystack's Web interface and are geared toward user interaction. For instance, the `query` function provides an interactive environment for browsing through the query results. We simply want a query to yield a list of relevant documents. The interface to the results should be determined at a higher level. The existing interface consists of a single shell command, `haystack`, which provides a number of different functions based on its first argument. There are three primary functions: `server`, which starts the Web interface, `archive`, which allows command line archiving of specific files or directories, and `query`. To create a more programmatic (agent friendly) version of Haystack, we created an analogous shell command `haystack_api`. The `haystack_api` command implements the set of commands described in Table 4.1.

These commands are all very similar to their cousins under the `haystack` shell command. However a key difference is the notion of a document ID number. Internal to Haystack, every document is assigned a unique archive number. The `haystack_api` program simply exposes these ID numbers to other programs to provide handles to specific documents. Thus, the `document` function allows programs to request more detailed information about specific documents. The `annotate` operation allows programs to simply specify which documents they want to annotate. Document IDs are essentially equivalent to unique document names, and within SodaBot, can be though of as document pointers. The `documents` command is an optimization for SodaBot which is essentially a document information server. It listens on its `stdin` for a series of Document IDs and returns detailed information about each of then. SodaBot takes advantage of this by spawning a separate thread just for interacting with this process. Thus, to get information about a new Document ID, the agent does not have to start a new process.

4.2 The Support Agents

Currently, the SodaBot system consists of a set of core basic services such as networked data connections, an expect mechanism, access to the Web, access to other agents, etc. These basic services are used to create groups of agents which can interact with each other. Typically, an agent will provide some specific service or skill that other agents might want to use. For example, all agents make use of the *Server* agent, which

Table 4.1: The Haystack Application Program Interface

Name	Arguments	Returns
`query`	A query string	List of relevant documents.
`document`	A document ID number	List of annotations for the document.
`documents`	A series of document ID numbers. Interactive	List of annotations for the documents.
`annotate`	A document ID number, Field Name, Field Value	0 on success, 1 on failure.
`archive`	A filename, directory or a Haystack DF file	The Haystack document ID created for the document.
`lookup`	A filename, directory or a Haystack DF file	The Haystack document ID of the document if it is in your Haystack. 0 otherwise.
`environment`	none	Lists Haystack environment variables.

acts as a name server, mapping agent names to machines. To make Haystack and its functionality available to other agents, we created a series of new agents. Thus both the personal "user" agents of Section 4.3 as well as agents in the Intelligent Room in general could make use of a standard set of Haystack functions.

4.2.1 The Haystack Agent

The *Haystack* agent is designed as an interface to the Haystack API described in Section 4.1. It consists of a number of requests corresponding to the functions of the Haystack API. In general these functions handle calling the proper command, process the I/O, and convert the results into usable representations. The *Haystack* agent contains the following requests:

Haystack.Query(QueryString) Calls the `haystack_api query` function with QueryString and returns an array of matching document IDs. A typical invocation might be:

```
@DocumentList = Haystack.Query('robotics');
```
[1]

[1]SodaBotL inherits Perl's variable type syntax. `@` begins an array variable name, `$` a scalar variable (one which can hold numbers or strings), and `%` a hash table variable (a symbolically

Haystack.Archive(FileName) Calls the `haystack_api archive` function on FileName. FileName can be any file, a URL, or a pointer to a Haystack DF file. The function returns the new Haystack document ID generated for the file. A typical invocation might be:

```
$NewDocumentID = Haystack.Archive('http://web.mit.edu/');
```

Haystack.Annotate(DocumentID, FieldName, FieldValue) Calls the `haystack_api annotate` function on the DocumentID and adds the given annotation, setting FieldName to FieldValue in the document's Haystack DF file. A typical invocation might be:

```
Haystack.Annotate($NewDocumentID, 'time', Time.Now());
```

Haystack.Document(DocumentID) Calls `haystack_api documents` function with the DocumentID and returns a hash table of document information. These hash tables contain annotations, pairs of field names and values. Some example fields are the document's author, title, location, or SodaBot specific annotations. A typical invocation might be:

```
%DocumentInfo = Haystack.Document($MyDocument);
```

Haystack.DocumentArray(DocumentArray) This request make use of the `haystack_api documents` function to get document information for each of the DocumentIDs in the array and returns an array of hash tables of document information. This is typically used to get information about about the set of documents returned by a query:

```
@RelevantDocs = Haystack.Query('agent theory');
@DocumentInfo = Haystack.DocumentArray(@RelevantDocs);
```

indexed array). Thus `$foo` could be a scalar such as `5` or `'hello'`. `@bar` could be an array such as `[1, 2, 'hello', 'world']`. . A hash table such as `%baz` is represented via an array of pairs, as in `['name', 'amy', 'age', 20]`

This simple set of actions is enough to give other agents access to the core functionality of Haystack. The goal is that some of the more powerful uses of Haystack, the collaboration, intelligent searches and annotation, will come from application agents developed on top of this interface. However, before we go on to show examples of other agents which use the *Haystack* agent, it is useful to discuss some other *support* agents we developed.

4.2.2 The Document Agent

In the course of developing the functionality of the *Haystack* agent, it became clear that SodaBot needs to have some notion of what exactly a document is in the first place. Since Haystack documents can take many forms (html, text, postscript, e-mail, etc.), it important for SodaBot to be able to recognize and handle these forms. Thus, in addition to the *Haystack* agent, we also built a *Document* agent. The purpose of the *Document* agent is to provide other agents with abstract means for manipulating documents. The *Document* agent provides one public[2] request, called `Display`, for displaying documents to users. It takes as its argument a document information hash table. The agent has several content specific private procedures for displaying documents. Thus, the `Display` request simply determines the content-type of the document and then calls the appropriate handler procedure. This is done using a map of content-types to agent function calls, similar to the mime-types convention which maps content-types to applications suitable for displaying them. It is a useful feature of SodaBot that both agent names and their requests can be stored in variables.

Actually, the *Document* agent would perhaps be better named the *Media* agent. There is no reason why we cannot support other content-types such as video or audio. In fact, the current implementation has handler routines for text, HTML, postscript, Intelligent Room video clips, and even camera views from the Intelligent Room. The design is easily extensible to have arbitrary agents handle display requests. Indeed, the many of the current handler routines already make use of other agents. For example,

[2]Public requests, like public methods in C++, are accessible to other agents

Content-Type	Handler	Other Agents Handler Uses
text/plain	Document.TextHandler	*Netscape*
text/html	Document.HTMLHandler	*Netscape*
application/postscript	Document.PSHandler	none
application/vcr	Document.VCRHandler	*VCR*
application/camera	Document.CameraHandler	*Mux*
application/speech	Document.SpeechHandler	*SpeechOut*

Table 4.2: Content-Type Handlers in the *Document* Agent

the `VCRHandler` procedure parses the document information table and then forwards the request to play the video to the *VCR* agent. Table 4.2 provides a summary of these handlers.

4.2.3 The Time Agent

For a number of reasons, we wanted to have a better means of manipulating times than the base SodaBot system provides. Initially, SodaBotL contained one primitive for handling times, `time`, which returned the number of seconds since January 1, 1970. This makes it easy to compare two times via subtraction. However, we wanted access to a more useful interpretation of the current system time in both machine and human readable formats. To provide this, we added two new primitives to the SodaBotL language. `ctime` provides a human readable version of the date such as "Sun Jun 1 17:47:29 EDT 1997" and `localtime` provides a machine readable version by returning an array of `[second, minute, hour, day, month, year, weekday, days into the year, a flag for daylight savings time]`. Thus now humans and agents alike can have an idea about what day of the week or what time of day a given time represents.

The *Time* agent provides access to these primitives through wrappers called `CurrentTime` and `PrettyTime`. It also provides a library of requests for comparing times with respect to common time periods. For instance there is a `Today` request which will return true if a time occurs in the current day. The agent also has `Yesterday`, `ThisWeek`, and `LastWeek` requests. Finally, to aid in agent human discourse, there is a `TimeOfDay` request which converts times into "morning," "afternoon," "evening," or "night." The *User* agent makes use of this function when it greets its owner.

4.3 The User Agent

Now that we have established a group of support agents, we can build our personal information management agent. The *User* agent is designed with two purposes in mind. The first is to give the Intelligent Room a representation of its individual users. The second is to provide these users with a personal information retrieval agent. This agent will attempt to capture information about its owner's documents, including their content and how they are used. It will then allow users to search the set of documents and information it captures. While the *User* agent is designed to integrate well with the Intelligent Room, much of its functionality is independent of the Room. We shall see that its use of Haystack, Netscape, and Zephyr are in fact completely separate from the Room.

4.3.1 Maintaining the Haystack

One of the primary responsibilities of the *User* agent is to maintain its owner's Haystack. When the *User* agent starts, it starts local copies of both the *Haystack* and *Document* agents for its use. Its goal is to automatically add new documents to its owner's Haystack, and, where possible, annotate these documents with information about their usage. The primary implementation of this has been through the use of the *Netscape* agent we originally developed for the Intelligent Room.

When the *User* agent starts, it starts a *Netscape* agent. The *Netscape* agent provides full agent control over the user's Netscape Web browser. This includes the ability to load documents into the browser and to detect where the browser goes. To be notified about any new pages the user loads, the *User* agent sets up a callback routine with the *Netscape* agent. This routine is called whenever the *Netscape* agent detects the user loading new pages[3]. This is done quite simply:

```
Netscape.URLWatchReg(User.NewUrl, "$UserName@ai.mit.edu");
```

This calls the *Netscape* agent's `URLWatchReg` request and tells it to call the *User* agent's `NewUrl` request whenever the *Netscape* agent sees a new page. As discussed below, `$UserName@ai.mit.edu` is the name of the AP the *User* agent is running on. This is just a small step in our quest to provide our agents with more information about what we are doing. However, from even this small bit of information, we can build useful applications which previously did not exist.

Figure 4-1 shows the *User* agent's `NewUrl` request, which gets called by the *Netscape* agent. It is actually quite simple. Its purpose is to automatically add each Web page it gets told about to the user's Haystack. It also timestamps each entry by adding an annotation, called `sodabottime`, to the document once it is in Haystack. The important thing to note is that this all happens transparently to the actual user. They can surf the web without noticing that their *User* agent is watching over their shoulder[4].

4.3.2 Searching the Haystack

The simple act of archiving every Web page a user goes to creates an excellent data set for querying. It is actually quite a useful data set to be able to search. Web sites are often browsed very causally, and due to the hypertext nature of the Web, users often are quite unaware of the URL of the site they are looking at. But they most certainly are aware of the topic of the site. Why not let them search their Web browsing history based on content? This is the function of the `User.InfoTime` request. It is called with two arguments, a query string called the `InfoTopic` and an optional

[3]This is accomplished through the use of a proxy Web server. The *Netscape* agent has a thread which starts this server. The server is a customized version of SFProxy[13], a proxy server written in Perl. The server outputs information about the pages that browsers request from it. This consists of the URL being loaded as well as the title of the page and, in one implementation, details about all the links on each page.

[4]This of course raises some privacy issues, which we will discuss in more detail in Section 5.4. For now, **caveat browser**.

```
# Callback for handling when user goes to a new web page.
Public NewUrl($URL) {

    print "Went to $URL";
    $Now = Time.CurrentTime();
    $NewDocID = Haystack.Archive($URL);
    print " - archived as $NewDocID";

    if ($NewDocID) {
        Haystack.Annotate($NewDocID, "sodabottime", $Now);
    }
}
```

Figure 4-1: The `User.NewUrl` Request

time predicate called the `TimePeriod`. When the `User.InfoTime` request is called, it first passes the `InfoTopic` to the user's *Haystack* agent in the form of a query. The `Haystack.DocumentArray` request then builds up document information hash tables for each document the query returned. If there is a `TimePeriod` specified, the `sodabottime` annotation is used to see which documents satisfy the time predicate. Once the agent has finalized a list of "good" documents, it then build up a Web page of choices. This page lists each document's title and any other relevant information Haystack and the *Haystack* agent can gleam from the file. Users can then follow the HTML link to any documents they are interested in seeing. For instance, the *Haystack* agent tries to run a simple algorithm for determining the title and author of postscript papers. Figure 4-2 shows the results of a query looking for all the documents about "java" the user saw "yesterday."

Due to the novel nature of the interface, we will defer describing the specific means for initiating searches until Section 4.4. An important point though, is that searches are currently entirely user initiated. That is, users have to make all information requests themselves. However, as the Room and its agents, particularly the *User* agent, are able to gain a better understanding of the current goals of its occupants, this will hopefully change. What we want is to have the system place a user's actions

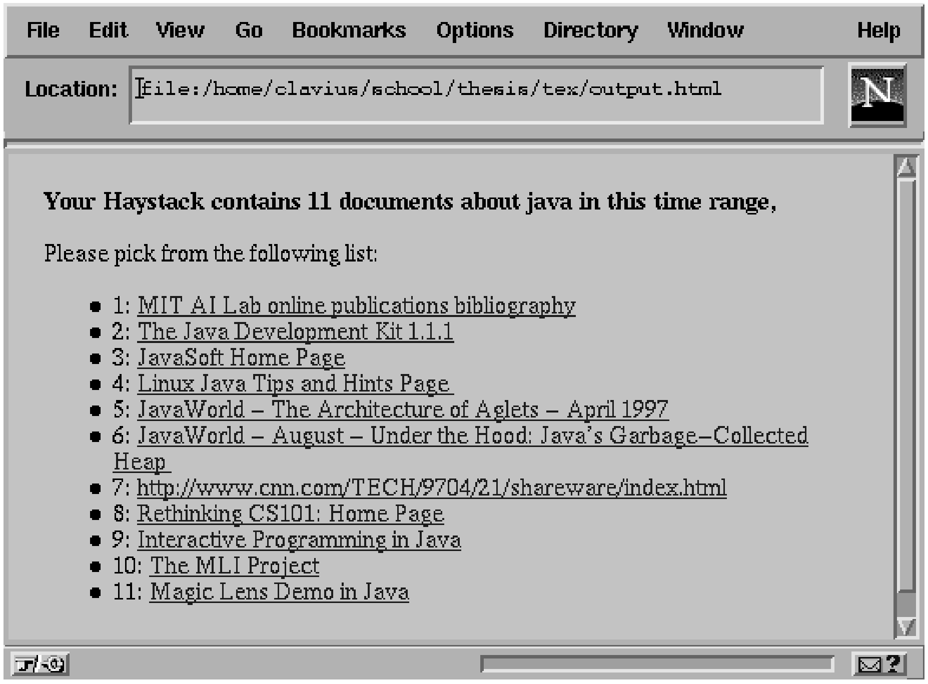

Figure 4-2: Results of Query About Java Documents Seen Yesterday.

in a specific context. Then, in an unobtrusive manner – similar to the Remembrance Agent – the Room should provide a list of suggested documents which best help the occupants meet their goals, given the current context. This will be the subject of future work, as the problem of determining a person's goals is still under study. In the Room, there are many more cues about occupants' goals than their keystrokes. In fact, often there are no keystrokes at all.

4.3.3 User Agent Naming

In Section 3.1 we describe the naming syntax for SodaBot Agent Platforms (APs). Every agent has a unique name and AP pair. By making use of AP name aliasing, it is straightforward to setup systems of agents which have no hard coded machine dependencies in them. We have extended this notion to the *User* agent as well. When a *User* agent starts, it registers an alias of `username@domainname` for its AP. This should typically correspond exactly to the user's email address. Thus, if another agent would like to query Michael's Haystack for information about robotics, it would make the following call:

```
@DocumentList = User.Query('robotics')[mhcoen@ai.mit.edu];
```

This provides a nice mapping from a person's email addresses to their agents. It follows the original design specification of SodaBot that each user would run their own AP.

One important side effect of the use of agents to search Haystacks is the property that the query is sent to the data. That is, when an agent makes a query request of a specific *User* agent, the *User* agent can be running on the machine of its choice. Presumably, this should be a machine where the user's Haystack data files are kept locally. This has the effect of reducing network bandwidth, since only the query and it's corresponding result set need traverse the network. However, it does have the effect of concentrating the search load on one machine. It would be relatively simple to create a load balancing set of search agents using SodaBot. The key is that it is important to have this tradeoff available when designing a large scale system. We realized a marked improvement in Haystack's performance by storing our data files on a disk local to the agent instead of the AI Lab's NFS file server.

4.3.4 Summarizing the User Agent

Even with a simple Web based interface for entering queries, we have described an interesting system so far. The automated archiving of Web pages into Haystack with annotations about their usage has provided us with a unique data-set about our

Web browsing history. Already we can find Web pages via our remembered ideas of their content. True, the querying interface is not very exciting. But we can now build upon the framework of the *User* agent combined with the Intelligent Room to improve this. Additionally, we will be able to make use of *User* agents to open the door for networked collaboration of personal information.

4.4 A Grammar for Speech Based IR

The current interface for a user to search their Haystack involves using the Intelligent Room as an input/output device. This was more than an exercise in building another application in the room. We found this type of interaction to be very rewarding as compared to the standard HTML forms based interface Haystack provides. Users can initiate their queries via a spoken language interface. The results are returned as a combination of a verbal summary as well as the textual descriptions in a Netscape window as presented above. This multi-modal combination of speech with the display of information sets up a dialog between the user and the Room. The use of the displayed list of information allows a common reference point for both the user and the Room. Thus, users no longer have to click on a document to see it, they can verbally request that the Room show it to them.

To do this, we developed a simple grammar in the Room's speech recognition system for information retrieval. Due to the restricted nature of the current speech recognition system, we limited the topic set to five predefined categories. However, we anticipate using an unconstrained recognizer as soon as a suitable one can be properly integrated in the Room. While this will not require changing any of our existing work, it will very much improve our information retrieval interactions. After all, information retrieval should not be limited a priori in its scope, especially if there are no limitations on the document set. By limiting the set of search topics we are really removing much of the power delivered by an IR system.

The speech grammar for requesting information consists of two production rules and several word classes. They are detailed in Table 4.3 and Table 4.4. Again, we

<InfoRequest> <InfoTopic>
<InfoRequest> <InfoTopic> I saw <TimePeriod>

Table 4.3: Speech Recognition Rules for Information Retrieval

Class	Members
<InfoRequest>	I need information about | Show me information about | Show me the document about | Show me the page about | What information do I have about
<InfoTopic>	robotics | agents | isreal | stocks | java
<TimePeriod>	today | yesterday | this week | last week

Table 4.4: Speech Recognition Classes for Information Retrieval

use callbacks to register interest. The *User* agent notifies the *SpeechIn* and provides a callback procedure, `User.InfoTime`, which we described in detail in Section 4.3.2. When the speech recognition system recognizes a sentence from this subset of its grammar, it sends calls the request with the "important" text of the statement. In this case, by important we mean the `<InfoTopic>` word and the `<TimePeriod>` if there is one. The nice thing about this is that while the speech recognition can support large grammar for rich interactions (i.e. multiple ways of asking for the same thing), in the end, the agents only want to know about what information topic the user is interested in, not how they asked for it. Thus for two typical spoken requests:

1. **User:** Computer, I need information about robotics.
2. **User:** Computer, show me the page about agents I saw yesterday.

The speech system sends the following to the *User* agent:

1. <InfoTopic>: robotics:
2. <InfoTopic>: agent: <TimePeriod>:yesterday:

Thus, the request for documents about "java" seen "yesterday" described in Section 4.3.2 would now be something like:

> **Lynn:** Computer, show me the page Java I saw yesterday?
>
> **Room:** Please wait while I search your Haystack.
>
> *Pause. Then Figure 4-2 appears.*
>
> **Room:**Your Haystack contains 11 documents about java in this time range. Please choose one.
>
> **Questioner:** Computer, show me the third one.

By implementing our personal information management system in the Intelligent Room, we have been able to provide it with a powerful spoken language interface. We have also set the stage for more interesting verbal interactions with the Room. We imagine providing tools for dictating annotations about the documents we get back from a search or providing verbal feedback about the search. For example, users could tell the Room that they "like this document because it's also about compilers." In a simpler example, a user might just be mumbling "this is interesting" about a document. The key is that the Room's architecture allows us to create high level systems which combine its many modes of interaction.

4.5 The RoomInfo Agent

The *RoomInfo* agent is to the overall Intelligent Room what the *User* agent is to individual users. The goal of the *RoomInfo* agent is to use Haystack to answer questions about the Intelligent Room itself. For instance, a visitor to the Room might want to ask "How does the tracking system work?" The Room should be able to answer a question like this in several ways – by describing the system verbally, by showing a Web page or paper about the tracker, or even by playing a video segment. Haystack, with its multimedia support, is the perfect repository for such information.

The actual SodaBotL code for *RoomInfo* agent is in fact very similar to the *User* agent. We added a similar grammar for querying the agent via the speech recognition system. It essentially consists of different ways to ask about five of the Room's subsystems: pointing, tracking, its agents, speech recognition, and the Room as a whole. For each topic, we manually found relevant documents and/or media which best explained the topic. These were entered into the Room's Haystack. We further annotated each of these documents with a field called `sodabotabouttheroom` and set its value to `true`. At the same time, we set up the Room's *Netscape* agent to automatically archive the web pages displayed on the Room's primary display. This provided the Room's Haystack with an unconstrained source of documents. The Haystack contained not only Web pages, but also postscript papers about the Room, information about video segments about the Room, and even descriptions of various camera views around the Room.

People in the Room can ask it about its systems verbally. The *RoomInfo* agent is notified about the topic and queries its Haystack for documents about this topic. It then sorts the results into two sets, those documents specifically about the Intelligent Room (i.e. those that are annotated with `sodabotabouttheroom` = `true`), and documents which are about the topic in general. The Room then displays both lists in the primary Netscape display, as in Figure 4-3. The questioner can now choose to see any of the returned documents. A sample interaction might be:

> **Questioner:** Computer, how does the tracking system work?
>
> **Room:** Please wait while I search my Haystack.
>
> *Pause. Then Figure 4-3 appears.*
>
> **Room:** I have 5 documents about the tracking system. Please pick one.
>
> **Questioner:** Computer, show me the second one.

When the questioner requests that a particular document be displayed, the *RoomInfo* agent uses the *Document* agent's `Display` request display the actual document. For the types of documents in the Room's Haystack, this can involve a number of actions. For instance, if the questioner wants to see a document which is a video segment about the tracking system, a number of things occur. First the Room says

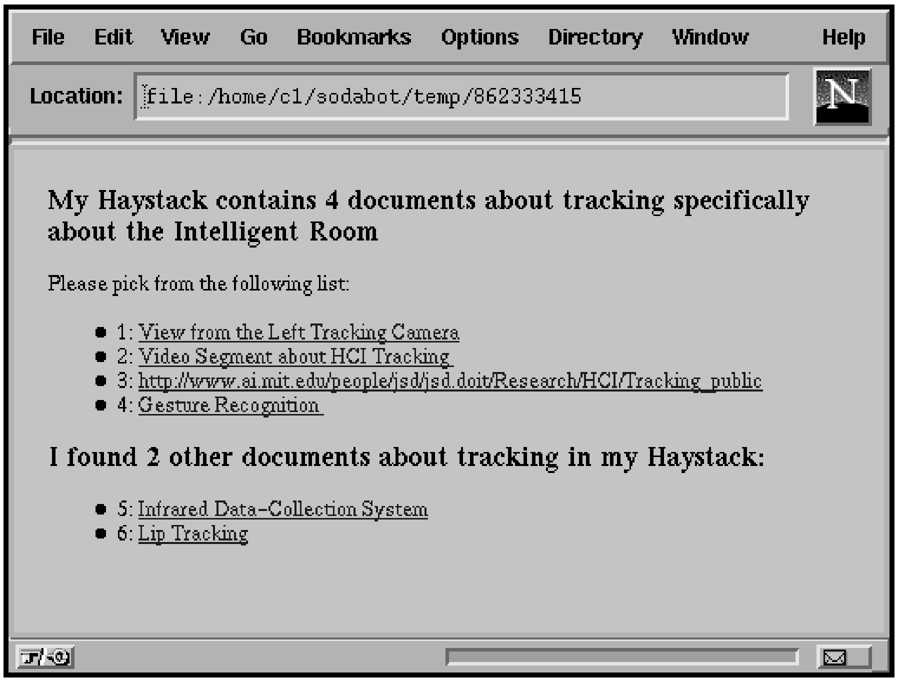

Figure 4-3: List of Documents About the Tracking System.

"Please wait while I fetch the segment." Then, while the VCR is cued to the proper location, the displays are adjusted so that the VCR is now displayed on the primary display, and the Netscape browser is moved to a secondary display. Finally, the video segment is played, for the tracking system this is a 30 second segment from the middle of a tape about the Room. The important thing to note about all of this is that the *RoomInfo* agent did not to know how to do any of this. It simply had a document it wanted to display. In fact even the *Document* agent had little to do with this. It simply knew it had a VCR segment to play. So, it passed the request to the *VCR* agent. This is where most of the work is performed.

So, we have applied the same base support agents used in our personal information management system to produce an information management system for the Intelligent Room. This is very much in the spirit of our goal to embed information retrieval into everything we do. Having developed a framework for information retrieval and document management, it was only natural to try and apply this technology in as many situations as possible. As more and more systems take advantage of the power to control and search information which IR systems provide, and as these systems are able to make use of each other, our information problems will subside.

4.6 Networked Haystacks

We have shown how Room users can search their own Haystacks, and the Intelligent Room now has a multimedia information repository at its disposal. In this section, we show that the same infrastructure that made this demonstration possible supports networking these agents together. By providing the *User* agent with the `Query` request in Figure 4-4, we have essentially set up a Haystack query server for each user. The example in Section 4.3.3 is how we might search another user's Haystack. Thus, if another agent would like to query Michael's Haystack for information about robotics, it would make the following call:

```
@DocumentList = User.Query('robotics')[mhcoen@ai.mit.edu];
```

To test this, we added a new rule to the speech recognition system for handling requests such as "What documents does Michael have about agents?" We also took advantage of context to allow the following type of interaction:

Lynn: Computer, I need information about Java.

Agent: Please wait while I search your Haystack.

Pause.

Agent I'm sorry Lynn, but your Haystack does not contain any documents about Java.

Lynn: Computer, what about Michael's haystack?

```
# This request returns an array of document information
# hash tables of documents matching $Topic in user's Haystack.
Public Query ($Topic) {

    print "calling query";
    @HSDocs = Haystack.Query($Topic);

    # get the docinfo for these documents
    @HSDocInfos = Haystack.DocumentArray(@HSDocs);

    reply(@HSDocInfos);
}
```

Figure 4-4: The `User.Query` Request

Pause. Then a list documents appears.

Agent: There are 12 documents about Java in Michael's Haystack. Please pick one.

These are exactly the types of rich interactions the Intelligent Room is designed for. Notice how the request to search "Michael's" Haystack does not respecify the search topic, it is implicit from the dialog context. Moreover, "Lynn" does not have to know anything more about "Michael" than his name. SodaBot and the AP naming system take care of finding "Michael's" *User* agent, and ultimately his Haystack. The system uses a mapping of names to email addresses to achieve this.

Of course, the sharing of Haystacks raises issues of security and privacy which for now we have not addressed. Section 5.4 discusses this in more detail.

Chapter 5

Discussion of Results

5.1 Summary

We have demonstrated several systems in this thesis. Primarily, we have constructed a set of agents which provide an interface between SodaBot and Haystack. Using these agents, we describe a personal information management system. By situating this system in the the Intelligent Room, we were able to leverage both the Room's multi-modal environment as well as its infrastructure of software agents. One of the results is our ability to use speech recognition as the input to our personal information management system and the Room's multimodal environment as the output. The user and the system engage in, what is now, a simple dialog for information retrieval. Furthermore, this system is integrated into the user's environment. It provides a framework for capturing, indexing, and querying the user's personal documents. We have also shown how these same technologies can be applied to provide a multimedia data store for information about the Intelligent Room.

5.2 Evaluating this work

There are a number of ways we can evaluate this work:

1. Does it satisfy the criteria of the introductory chapter (Section 1.2)? These are our desiderata for a personal information manager – personalized, automated,

ubiquitous, and networked.

2. Independent of these criteria, is the system useful and helpful?

3. Is it novel? Does the system provide new and unique functionality?

In the following sections we will examine each of these in depth.

5.2.1 Essential Attributes of a Personal IM System

In Section 1.2 we outline the four desiderata we feel are essential in a personal information management system. These four attributes (personalized, automated, ubiquitous, and networked) provide a base set of functionality against which we can evaluate our work.

- **Personalized:** The *User* agent is certainly personalized. Indeed, for simple queries, it can *only* provide us with documents we have already seen before. This is what we wanted: a system which performs information retrieval on our set of personal documents. However, this is too limiting. There is a certain lack of *serendipity* in such a system. This is addressed to some degree by providing accesses to other users' Haystacks, via networked *User* agents. However, it is sometimes refreshing to get back a completely unexpected result set when searching the entire Web. What we probably want is a middle ground where if the agent fails to find relevant documents in a our personal IM system, it gracefully degrades to searching, first our neighbors' IM system, and then the entire web. This solution also provides us with an answer to every question. Still, in the end, the *User* agent provides the specified functionality: it is the efficient assistant who can find the documents we vaguely recall seeing.

- **Automated:** For the limited world of the Web, the *User* agent discussed in Chapter 4, automatically archives every document we see. The key for automating this process is the development of supporting agents such as the *Netscape* agent. These agents need to be aware of what users are doing with specific applications. The other area of automation which the system demonstrates is passive annotation. In the case of the Web, it means annotating each document with when we viewed it combined with Haystack's attempts to extract author and title information. As the system is further integrated into users' desktops, this should grow to include numerous other automatically generated annotations. For instance, we imagine annotating documents with how long they are used, who they are sent to or shared with, or any other cue about how interesting a document is to the user and why it is interesting.

- **Ubiquitous:** Again, for the limited world of the Web, the *User* agent archives every document we see. While this definitely a subset of the documents a typical user interacts with, our primary concern is that if the *User* agent is told about a document, it can index it. That is, as other agents are written which are aware of other document interactions (e.g. an email agent), the system can easily make use of them to provide information retrieval. We have briefly experimented with a *Zephyr* agent to integrate the Zephyr[8] instantaneous messaging system with the *User* agent. Once we had provided the *Zephyr* agent with access to each incoming message, having the *User* agent index each message was simple. Unfortunately, we encountered problems with Haystack due to the high rate of archiving. We would typically get bursts of a dozen messages in less than a minute. The present implementation of Haystack running in a Linux environment cannot handle such a high rate of indexing.

- **Networked:** As described in Section 4.6, we achieve most of this goal simply through our use of SodaBot. *User* agents are able to query each other using the same interface they use to query themselves. The main impediment to truly analyzing this criterion however is the lack of a substantial use community. So, while the architecture is in place for networked personalized information retrieval, there is not much more we can say, at this point.

Overall, we feel the *User* agent embodies these principles and is a fine foundation to build upon. Section 6.1 discusses some of this future work.

5.2.2 Is it Useful and Helpful?

In limited testing, the *User* agent certainly tries to fill a definite gap in information retrieval. For example, in the course of researching much of the previous work cited in Section 2.4, our *User* agent watched over our shoulder. This proved useful weeks later when we wanted to find a reference to, say, Java based agents, that we knew we had seen before. In this respect the system worked remarkably like the augmented memory we want it to be.

Unfortunately, without a larger user group it is difficult to analyze the usefulness and effectiveness of these *User* agents, particularly the aspects of networked agents. Without conducting user studies it is hard to claim we have solved the problem of personal information management. So, while our limited results have been promising, it is not possible to make an adequate assessment of the systems success.

The limited results we do have suggest that networked queries can be quite useful. Essentially, networked *User* agents find things that web searches would not, and perhaps could not. Consider a search of a colleague's Haystack which yields internal or unpublished documents. While a good *Intranet* search engine (of which there are, as of yet, few) would yield similar results for some queries, it is impossible to also get colleagues' annotations, whether explicit or implicit. Furthermore, such a search is again limited to only documents which are explicitly published by their authors.

5.2.3 Is it Novel?

There are few if any existing systems which provide this level of personalized information management. As we fill in the gaps in document coverage to capture *all* of a user's information interactions, we will truly have a novel system. The closest system to our own, the Remembrance Agent[37], certainly has similar intentions. However its strength lies much more in its automated *search* facilities, not in automatically *archiving* documents. The Remembrance Agent basically requires users to hand-pick which documents to store in their repository. Additionally, they do not support the networked search capabilities our agents give us.

The key is our architecture, which combines the framework of SodaBot, largely as deployed in the Intelligent Room, and the power of Haystack to manage information. This level of information retrieval integration into the desktop[1] is uncommon, but clearly valuable. We hope to see our system or a descendant in wider use to further validate this proposition. Perhaps the most *novel* aspect of our system is its extensibility. As Section 6.1 describes, we see a great deal of additional functionality being integrated. This is primarily due to the ability of new agents to easily make use of the existing set of agents, and visa-versa. SodaBot's design will allow us to create new agents, such as *E-Mail* or *Net-News* agents, and quickly have the *User* agent make use of them.

5.3 An Experimental Haystack

Once we had provided SodaBot with an interface to Haystack, we were able to implement several of the higher level goals of the Haystack project in an experimental system. As we outline in Section 3.2, these goals include being a personalized, content-aware, annotatable, multi-user, user-profiling information management system. We have used SodaBot and the Intelligent Room's infrastructure to create a system which either meets or provides support for all of these goals. Admittingly, we leverage the annotatable content-aware nature of the existing Haystack implementation. However, we have both added new automated annotation facilities as well as support for new content-types such as video and camera views. Moreover, the networked nature of SodaBot made this a multi-user system with very little additional work. While we have not done any work towards the goal of supporting user-profiles, we certainly provide a strong framework upon which to do so.

This is perhaps one of the most valuable results of our work. While our system is certainly not as complete and "bullet-proof" as the Haystack specification calls for, it does provide most of the functionality. Or, perhaps more importantly, it provides an excellent framework for experimentation. For instance, it would be very easy to

[1] Even if this "desktop" is the rather virtual one in the Intelligent Room.

use the existing *User* agent to experiment in collaboration and learning. *User* agents could learn which other users tend to answer different types of queries best. While the Haystack project hopes to build such a system, they are currently far away from the multi-user networked stage. We very quickly have provided them with a tool for experimenting with new ideas. Moreover, the usefulness of our system has, in a sense, validated the very idea of Haystack.

5.4 Privacy

One issue we have continually put off has been the utter lack of privacy in our system. Every page a user browses is both added to their Haystack and is made available to other users. There are numerous reasons why we would not want other users to know about *every* Web page we browse. Consider what would happen if an employer noticed an employee's extensive browsing of a competitor's "We're Hiring!" page. Furthermore, as we extend the system to encompass email and other documents, there is no question that some documents truly are *personal* for a good reason.

There are a number of ways we can address this issue. In the most optimistic solution, our *User* agent is smart enough to know which documents are private (e.g. letters from friends and family) and which are public (e.g. work related mail or Web sites). While this is actually not too unrealistic for many situations, there will always be exceptions. Two other solutions require user intervention, one at archive time, and one at any later time. Whenever the system adds a document to its index, it could allow the user to choose the document's specification. This could also provide a nice interface for entering annotations. However, this goes against our goal of automatically and unobtrusively archiving every documents. Users will quickly grow tired of being prompted about each document. One modification would be to have the user periodically process batches of documents. This too is unsatisfying. We could provide a tool for browsing a user's document set and assigning visibilities. Again, this requires both human intervention and processing each document.

A Draconian solution would be to mark each document as private unless otherwise specified. We prefer a compromise. Allow the system to be in "Private" or "Public" modes. The system would attempt to use some AI techniques to guess which mode to be in. However, the user would always be aware of which mode the system was in, and could easily override it. Furthermore, if a user was about to research about their upcoming kayak trip, they could set the system in permanent private mode for the duration of that session. While this is better, it can still lead to mistakes.

Overall, we feel this will be an interesting area for future research. As there are more networked personalized information management systems this will become more of a problem.

Chapter 6

Conclusion

We have built a personalized information management system using a system of SodaBot software agents which make use of the Haystack information retrieval system. This system has provided us with a new tool for gaining control over the wealth of information around us. Primarily we now have means for indexing and searching the set of Web pages we browse. To search even this limited set of documents has proven to be quite useful. We feel that our system provides an excellent framework upon which to build a more complete implementation. Furthermore, we also claim that our work has validated many of the goals of the Haystack project. Personal information management is not only useful, but soon to be necessary. As individuals become dependent on information for their workplace needs, a tool such as ours will be indispensable. Moreover, in the spirit of our first sentence that "this thesis is about the fundamental need to embed information retrieval into everything we do," there is still a tremendous amount of data to be mined. As information is more textual and less structured, it will be by making IR systems more accessible that we enable new applications.

6.1 Future Work

While our existing system of agents has proven to be quite powerful, it opens more doors than it closes. We see a number of possible uses of this technology as well as additions to the existing system. Many of these ideas stem from the power of adding information retrieval to a active and "aware" environment such as the Intelligent Room.

Our favorite example of what a future system invovles recording and indexing design sessions. Imagine a group of engineers in an intelligent conference room designing a section of an airplane wing. The entire conversation is being recorded, the speech run through a recognition system, and the video being index with the recognized speech. Furthermore, the system is saving whiteboard images. We'll ignore the fact that the engineers are also using an IR system to pull up various design specification documents and blueprints. What we are interested in is recording the design of this airplane wing for future recall. For, what happens three years from now when a plane crashes because its wing snaps? How useful would it be, three years in the future, to be able to say "How was this wing designed?" and get as a response, not only some papers and email on the wing design, but also the entire recorded design session? We think that this will be a major application of the future. The key is to seamlessly integrate information recording and retrieval into an active environment.

A less ambitious addition to the current system is to build a *specialist finder* agent. The premise is simple – given a topic or question, find the person who knows the most about it. This would involve searching multiple Haystacks for the user with the set of documents which seems most relevant. This would be tremendously useful and time saving. Since the person who answers a question is usually the last one we ask, why not skip the middlemen and make them the first? Such an extension recognizes the fact that while actual documents and retrieval systems contain a great deal of knowledge, they cannot hope to compete with the brain of an expert in a given field. Kautz et al.[24] have built such a system based around a similar *User* agent to our own. They make use of users' email and files to develop profiles which agents share with each other. Given the historical connection of their "visitorbots" [25] to our own SodaBot, it is no surprise that they too took advantage of their architecture to build such a system.

We hope to more completely integrate the *RoomInfo* agent and the Intelligent

Room. We envision automated multimedia tours of the Room, and, meeting one of the Room's earlier goals, of the entire AI lab. Such a tour would consist of speech synthesized narration, web pages, and video segments. We could use the *Haystack* agent to provide a source of documents about each of these topics to allow users to get more information about topics they are particularly interested in.

There are several possible additions to our existing use of Netscape and the Web. As described in Section 2.4.1, we can enhance our Web browsing experience along the lines of Letizia or Web Watcher. This would involve the *User* agent looking ahead of the current page a we are browsing and having it pick the links which are most likely to be interesting to us. A collaborative version of this could highlight the links our colleagues have followed.

One important consideration is that the Haystack project itself hopes to one day encompass much of our current functionality. Keeping our system synchronized with Haystack will be an important future task. As Haystack begins to provide more features, we may be forced to concede some of them to the native system. Furthermore, it may well prove to be easier to use a simpler, off-the-shelf IR system in place of Haystack. This remains to be seen. We hope, however, that the two projects continue to work together.

Bibliography

[1] Ohio Environmental Protection Agency. The 1995 Ohio State of the Environment Report. http://www.fsu.edu/~cpm/segip/states/OH/pop.html.

[2] R. Armstrong, D. Freitag, T. Joachims, and T. Mitchell. Webwatcher: A Learning Apprentice for the World Wide Web. *Proceedings of the 1995 AAAI Spring Symposium on Information Gathering from Heterogeneous, Distributed Environments*, March 1995.

[3] A. Birrell and B. Nelson. Implementing remote procedure calls. Technical Report CSL-83-7, Xerox, October 1983.

[4] R. A. Brooks. A Robust Layered Control System for a Mobile Robot. Technical Report 864, MIT Artificial Intelligence Laboratory, Cambridge, MA., 1985.

[5] M. Coen. Building Brains for Rooms: Designing Distributed Software Agents. To Appear in AAAI 1997. http://www.ai.mit.edu/projects/hal/brain.ps.

[6] M. Coen. SodaBot: A Software Agent Environment and Constuction System. Technical Report 1493, MIT Artificial Intelligence Laboratory, Cambridge, MA., June 1994.

[7] Infoseek Corporation. Infoseek. http://www.infoseek.com.

[8] C. A. DellaFera, M. W. Eichin, R. S. French, D. C. Jedlinksy, J. T. Kohl, and W. E. Sommerfeld. *Section E.4.1: Zephyr Notification Service*. MIT Project Athena, Cambridge, MA, December 1987.

[9] D. Dreilinger. SavvySearch. http://guaraldi.cs.colorado.edu:2000/.

[10] O. Etzioni and D. S. Weld. A Softbot-Based Interface to the Internet. *CACM*, July 1994.

[11] O. Etzioni and D. S. Weld. Intelligent Agents on the Internet: Fact, Fiction, and Forecast. *IEEE Expert*, August 1995.

[12] L. Foner. Yenta: A Multi-Agent, Referral Based Matchmaking System. In *The First International Conference on Autonomous Agents (Agents '97)*, Marina del Rey, California, February 1997.

[13] N. Glovert and U. Pfeifer. Sfgate: A Gateway Between the WWW and WAIS. http://ls6.informatik.uni-dortmund.de/ir/projects/SFgate.

[14] R. S. Gray, D. Rus, and D. Kotz. Transportable information agents. Technical Report PCS-TR96-278, Dartmouth College, Computer Science, Hanover, NH, February 1996.

[15] K. Hammond and J. Kozlovsky. Knowledge-based Information Retrieval for Semi-Structured Text. In *Working Notes from AAAI Fall Symposium on AI Applications in Knowledge Navigation and Retrieval*, 1995.

[16] W. C. Hill, J. D. Hollan, D. Wroblewski, and T. McCandless. Edit Wear and Read Wear: Their Theory and Generalization. In *ACM CHI'92 Human Factors in Computing Systems Proceedings*, pages 3–9. ACM, ACM Press, 1992.

[17] AltaVista Inc. AltaVista. http://www.altavista.digital.com.

[18] Lycos Inc. Lycos: Catalog of the Internet. http://www.lycos.com.

[19] Yahoo! Inc. Yahoo! http://www.yahoo.com.

[20] J. Jannotti. Private communication with the author., 1997.

[21] A. Jennings and H. Higuchii. A Personal News Service based on a User Model Neural Network. In *IEICE Transactions on Information and Systems*, March 1992.

[22] D. Karger and L. A. Stein. Haystack: Per-User Information Environments. http://www.ai.mit.edu/projects/haystack/karger-stein-9702.html.

[23] B. Katz. Using English for Indexing and Retrieving. In P. Winston and S. Shellard, editors, *Artificial Intelligence at MIT: Expanding Frontiers*, volume 1, Cambridge, MA, 1990. MIT Press.

[24] H. Kautz, A. Milewski, and B Selman. Agent amplified communication. *Proceedings of the 1995 AAAI Spring Symposium on Information Gathering from Heterogeneous Distributed Environments*, March 1995.

[25] H. Kautz, B. Selman, M. Coen, and S. Katchpel. An Experiment in the Design of Software Agents. In *Proceedings of the Twelfth National Conference on Artificial Intelligence, AAAI-94*, Seattle, Washington, 1994.

[26] K. Lang. Newsweeder: Learning to Filter Netnews. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 331–339, Tahoe City, CA, July 1995.

[27] Y. Lashkari, M. Metral, and P. Maes. Collaborative Interface Agents. In *Conference of the American Association for Artificial Intelligence*, Seattle, WA, August 1994.

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

[29] D. Maltz. Distributed Iinformation for Collaborative Filtering on Usenet News. Master's thesis, Massachusetts Institute of Technology, May 1994.

[30] U. Manber and S. Wu. Glimpse: A Tool to Search Through Entire File Systems. Technical Report TR93-34, The University of Arizona, Department of Computer Science, Tuscon, AR, October 1993.

[31] M. Mauldin and J. Leavett. Web Agent Related Research at the Center for Machine Translation. Presented at SIGNIDR meeting, August 1994.

[32] M. Metral and M. Eng. The WebDoggie Personailzed Document Filtering System. http://webhound.www.media.mit.edu/projects/webhound/www-face.

[33] M. Minsky. *The Society of Mind*. Simon and Schuster, 1986.

[34] A. Moukas. Amalthaea: Information Discovery and Filtering using a Multiagent Evolving Ecosystem. *International Journal of Applied Artificial Intelligence*, 1997.

[35] M. Pazzani, J. Muramatsu, and D. Billsus. Syskill & Webert: Identifying interesting web sites. In *Proceedings of AAAI Conference*, 1996.

[36] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl. GroupLens: An Open Architecture for Collaborative Filtering of Netnews. In *Proceedings of ACM 1994 Conference on Computer Supported Cooperative Work*, pages 1175–186, Chapel Hill, NC, 1994. ACM.

[37] B. J. Rhodes and T. Starner. Remembrance Agent: A continuously running automated information retrieval system. In *The Proceedings of The First International Conference on The Practical Application Of Intelligent Agents and Multi Agent Technology*, pages 487–495, 1996.

[38] G. Salton. *The SMART Retrieval System - Experiments in Automatic Document Processing*. Prentice Hall, 1971.

[39] G. Salton. *Automatic Text Processing: The Transformation, Analysis, and Retrievalof Information by Computer*. Addison-Wesley Publishing Company, Inc., Reading, MA, 1989.

[40] E. Selberg and O. Etzioni. Multi-Service Search and Comparison using the MetaCrawler. In *Proceedings of the 4th International World Wide Web Conference*, 1995.

[41] U. Shardanand and P. Maes. Social Information Filtering: Algorithms for Automating "Word of Mouth". In *Proceedings of the CHI-95 Conference*, Denver, CO, May 1995. ACM, ACM Press.

[42] B. Sheth. A Learning Approach to Personalized Information Filtering. Master's thesis, Massachusetts Institute of Technology, Febuary 1994.

[43] L. A. Stein. In the eye of the beholder. To appear in *IEEE Expert* Sepcial Issue on Software Agents.

[44] D. B. Terry. A Tour Through Tapestry. In *Proceedings of ACM Conference on Organizational Computing Systems (COOCS)*, New York, 1993. ACM Press.

[45] M. Torrance. Advances in Human Computer Interaction: The Intelligent Room. In *Working Notes of the CHI '95 Research Symposium*, Denver, Colorado, May 1995.

[46] S. Vinoski. CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Environments. *IEEE Communications Magazine*, 14(2), Feburary 1997.

[47] L. Wall and R. L. Schwartz. *Programming Perl.* O'Reilly & Associates, Inc., Sebastopol, CA, March 1992.

[48] J. White. Mobile agents white paper. http://www.genmagic.com/agents/Whitepaper/whitepaper.html, 1996.

[49] I. H. Witten, A. Moffat, and T. C. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images.* Van Nostrand Reinhold, New York, 1994.

[50] T. Yan and H. Garcia-Molina. SIFT - A Tool for Wide-Area Information Dissemination. *Proceedings of the 1995 USENIX Technical Conference*, pages 177–186, 1995.