ABSTRACT

Agents are emerging technology that is making computer systems easier to use by allowing people (computer users) to delegate work back to the computer. They act on user's behalf and achieve user's goals. The advent of these software agents gave rise to much discussion of just what such an agent is, and of how they differ from programs in general. Agents are in research area for a long time. Earlier they were considered a part of just Artificial Intelligence. Nowadays they are just not in research area anymore. Commercial agent based applications are existing in the market because of the popularity of different technologies and toolkits. Now it is easier and convenient to incorporate artificial intelligence in a programming practice. Software Agents are suitable for various kind of applications. Mobility is one of the important properties, agents can have which make them superior to other agents. Mobile agents can facilitate message transfers between systems when used in System and Network Management. IBM's Aglets Software Development Kit [ASDK] is popular and de facto standard for developing a mobile agent. Information Search and Filtering is another domain where agents are useful. NJPIES (New Jersey Program for Information Ecology and Sustainability) is associated to environmental data collection, compilation, integration and provision. 'EnviroSearch' application mentioned in this report is for NJIT's NJPIES program and is actually a search agent developed in Object Oriented Java language. Agent searches environment related URLs (web sites) for documents, articles, etc. on the Internet, analyzes them based on their content. Selected URLs are then stored in Oracle database with JDBC interface.

AGENTS: CONCEPT, TECHNOLOGIES & APPLICATIONS AND SEARCH ENGINE TO COMPILE URLs CONTAINING DOCUMENTS RELATED TO ENVIRONMENT SCIENCE FOR NJPIES

by
Dhaval P. Shah

A Thesis
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Computer Science

Department of Computer and Information Science

August 1999

APPROVAL PAGE

**AGENTS: CONCEPT, TECHNOLOGIES & APPLICATIONS
AND
SEARCH ENGINE TO COMPILE URLs CONTAINING DOCUMENTS RELATED TO ENVIRONMENT SCIENCE FOR NJPIES**

Dhaval P. Shah

Dr. Franz Kurfess, Thesis Advisor Department of Computer and Information Science New Jersey Institute of Technology, Newark, New Jersey.	Date
Dr. Marcus Healey, Committee Member Department of Environment Engineering, New Jersey Institute of Technology, Newark, New Jersey.	Date
Prof. Jason Wang, Committee Member Department of Computer and Information Science New Jersey Institute of Technology, Newark, New Jersey.	Date

BIOGRAPHICAL SKETCH

Author: Dhaval P. Shah

Degree: Master of Science

Date: August 1999

Undergraduate and Graduate Education:

- Master of Science in Computer Science,
  New Jersey Institute of Technology, New Jersey, 1999

- Bachelor of Science in Computer Science,
  Gujarat University, Ahmedabad, India, 1990

Major: Computer Science

Presentations and Publications:

Dhaval Shah, Franz Kurfess, Klaus Holthaus and Felip Mirrales,
"Monitoring Distributed Applications with Intelligent Agents",
Engineering of Computer Based System (ECBS) Conference & Workshop,
March 8-12, 1999. Hotel Marriot, Nashville, TN.

Dhaval Shah
"Research on Intelligent Agents in Industry",
Software Engineering Lab Weekly Meetings, NJIT, October 1998.

Dhaval Shah
"Developing Mobile Agents using Aglets (ASDK by IBM)",
Software Engineering Lab Weekly Meetings, NJIT, November 1998.

Dedicated to my family ...

ACKNOWLEDGEMENT

I would like to express my greatest appreciation to Prof. Franz Kurfess without whom the Intelligent Agents would have been still unknown area to me. I also thank him for providing me with an excellent opportunity to work on couple of interesting projects which helped me writing a paper that published in ECBS '99 Conference Journal. I also want to thank Klaus Holthaus and Felip Miralles, senior students at NJIT who actually started the project "Monitoring Distributed Applications with Intelligent Agents" and did most of the work I used in my paper.

I am also obliged to Dr. Marcus Healey for his inspiration, support and adjustable nature. He really had trust in me and offered all possible help during development of "EnviroSearch" search engine in NJPIES. I also want to express my gratitude for Prof. Leon Jololian, George Chang and Deepak Girigaddi (Oracle Administrator). They have helped me number of times during my thesis period.

I would also like to thank the Graduate Students Association and Mr. Arvind Goel (CEO of UnixPros,Inc.) for providing me with the financial support which enabled me to attend ECBS '99 conference at Nashville, TN. I also take this opportunity to thank Annette Damiano at Graduate Studies Department, NJIT for helping me with my thesis documentation.

TABLE OF CONTENTS

Chapter	Page
1 OVERVIEW	1
2 INTRODUCTION	5
2.1 Agents - Definitions and Concept	5
2.2 Environmental Information System and NJPIES	7
3 AGENTS AND THEIR APPLICATION IN INSM	11
3.1 Agents - A Broad View	11
3.2 Agent Technologies	18
3.3 Monitoring Distributed Applications with Intelligent Agents	29
3.3.1 Background	29
3.3.2 Intelligent Agents for Systems Management	30
4 DESIGN AND ARCHITECTURE OF ENVIROSEARCH	42
4.1 Requirements	44
4.2 System Design	45
5 IMPLEMENTATION	53
5.1 Front End Tools	53
5.1.1 Meta-Search Engine	55
5.1.2 Database Search Script	57
5.2 Back End Tools	58
5.2.1 Database	58
5.2.2 Agent Program	59

Chapter **Page**

5.2.3 JDBC 65

6 EVALUATION 69

7 CONCLUSION 71

8 FUTURE WORK 73

REFERENCES 76

LIST OF FIGURES

Figure **Page**

1. The Home Page of NJPIES 10
2. General Architecture 32
3. System Components 33
4. Distribution Window (INSM) 35
5. A Sample HTML Document 46
6. Actual content of a HTML page after removing HTML tags 47
7. Retrieved links (URLs) from the sample HTML Page 47
8. System Components of an Architecture 48
9. CGI-Database Connectivity 49
10. User Interface to EnviroSearch 54
11. Meta-Search Engine Screen 57
12. Steps followed by an agent 62
13. JDBC in two tier model 67
14. JDBC in three-tier model 68
15. Future Work 75

CHAPTER 1

OVERVIEW

In my last two semesters of this thesis period, I have mainly tried to learn Software Agents in detail. Before the start of the thesis, it was a new concept for me; even though research has been going on in this field for a long time and similarly technologies related to agents like Java, Aglets, Telescript, KQML (Knowledge Query and Manipulation Language [UMBC_KQML]), AOP(Agent Oriented Programming [Shoham92]) and CGI (Common Gateway Interface) Programming were also new to me. The idea for this subject was given to me by Dr. Franz Kurfess. In 1997, two of his students, Klaus Holthaus and Felip Miralles, tried to use Intelligent and Mobile Agents [Cockayne98] in an application to monitor distributed applications [Klaus97]. That experiment was still in design and prototype phase, and was very good candidate for further research. At the same time there were also some doubts about any possible progress in that. Klaus and Felip noted in their project report that there were some design flaws and resource problems. That idea, their work and related problems led me to form following objectives at the beginning of my thesis.

- Learning Internet technologies and becoming familiar with Agent concept.

  As I acknowledged before, I did not know anything about Agents and related technologies so acquiring required skills was one of the goals itself.

- Trying to extend the work done by Klaus Holthaus and Felip Miralles; to put Intelligent Agents technology in Distributed Application Management Tasks.

- In the case where I can not pursue the earlier objective, I also kept fall back objective of finding an application or domain where I can put my newly acquired skills.
- Analyze the developed application in terms of its usefulness, performance, throughput and user friendliness.

At NJIT, Dr. Marcus Healey from Environmental Engineering Department, Dr. Franz Kurfess and Dr. Jason Wang from Computer Information and Science Department, are involved with NJPIES program (New Jersey Program for Information Ecology and Sustainability) which allows users to access information (data) related to different pollutant chemicals, factories using those chemicals, their emission data for the year, etc. NJPIES also intends to become a large repository of data containing environment related documents and research. NJPIES is introduced in a separate chapter along with its objectives. That solved the problem of finding an application domain for Agent technology.

Agents are popular in Information Retrieval and Filtering applications. When user searches for some information in the Internet, he is lost most of the time in the piles of data thrown to him. Agents can learn user's choice over the time and can use that knowledge to search information which is very specific to the user. Unnecessary data can be filtered by that agent. To achieve a similar goal, I have designed, developed, implemented and tested a software agent kind program which can download lot of interesting articles, research papers, news articles, etc. on Environment Science from the Internet and then store those URLs in the local database at NJIT after filtering useless information or URLs. I have named this application as "EnviroSearch" to remain consistent with other products of NJPIES.

This documentation basically describes all my efforts, experimentation, research and learned skills which I came across during this thesis period. My initial efforts were in learning the application "Monitoring Distributed Applications with Intelligent Agents" and then I focused on "EnviroSearch". Besides these two projects; I have gathered a lot of valuable information relating to existing trends of Intelligent Agents research and related work in the industries.

Chapter two and three are dealing with formal introduction of these projects, related work and my contribution. Chapter four describes the design and architecture of my search engine "EnviroSearch"; which I developed for NJPIES. EnviroSearch engine allows user to search environment related documents from the Internet and also from NJIT's local database (EnviroDB). Special agent program has been developed which feeds this database with related documents and URLs. Chapter five has all details about the implementation efforts of this search engine. It describes all related technologies involved, user's interaction details, problems encountered during the implementation phase, integration efforts and some sample resulted screens. Chapter six is dedicated to 'Evaluation' of my implemented program (the search engine). It checks the validity of the result and integrity of the overall design. Chapter seven is for conclusions and summary of overall activities. For the benefit of next batch of students who would like to extend my work on this package, I have identified some missing things and enhancement possibilities. That is listed as future work in chapter eight. This document ends with

References section which lists all books, articles and web sites, I have used in my implementation and in this documentation.

CHAPTER 2

INTRODUCTION

Agents are emerging technology that is making computer systems easier to use by allowing people (computer users) to delegate work back to the computer. They act on users' behalf and achieve users' goals. They help doing things like finding and filtering information which users are interested in. They also customize the presentation of that information depending on users' liking. They also help automating some of the activities. This thesis was targeted to explore this technology in detail and then use that research to come up with some useful product. This chapter introduces term "Agent" in some detail and then introduces "NJPIES" program referenced earlier; and its objectives in more details.

2.1 Agents - Definitions and Concept

In a real life, we, human beings often use services from different agents. Real estate agents provide services related to buying and selling homes whereas travel agents provide services like booking a vacation, reserving flight tickets, booking hotel accommodations, etc. These agents provide these services by charging us some fees and then try to achieve our objectives. In computer world also; similar concept is becoming popular in which software programs try to do what humans (users) were doing manually earlier. These agent bear different names depending on properties they exhibit while achieving user's goals. They work as mobile agents, intelligent agents, autonomous agents or Internet agents. There are several definitions in the industry which describe these agents concept from simplest to complex words. Simplest being the following one.

"An agent is a software thing that knows how to do things that you could probably do yourself if you had the time."

- *Ted Selker of the IBM Almaden Research Centre [IBM95]*

The definition described above doesn't convey the complexities which can be handled by agents in a changing environment. Following definition describes it in that context.

"A piece of software which performs a given task using information learnt from its environment to act in a suitable manner so as to complete the task successfully. The software should be able to adapt itself based on changes occurring in its environment, so that a change in circumstances will still yield the intended result."

- *G.W. Lecky-Thompson [Stan96]*

Workers involved in agent research have offered a variety of definitions, each hoping to explicate his or her use of the word "agent." Possibility is that each of them grew directly out of the set of examples of agents that the definer had in mind. Here is some more of these definitions.

The AIMA Agent [Russell and Norvig 1995]: "An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors." AIMA is an acronym for "Artificial Intelligence: a Modern Approach," a remarkably successful new AI text that is followed in lot of universities and schools.

The IBM Agent [http://activist.gpl.ibm.com:81/WhitePaper/ptc2.htm]: "Intelligent agents are software entities that carry out some set of operations on behalf of a user or another program with some degree of independence or autonomy, and in so doing, employ some knowledge or representation of the user's goals or desires."

Agents are in research area for a long time. Earlier they were considered a part of just Artificial Intelligence. Nowadays they are just not in research area anymore. Commercial agent based applications are existing in the market because of the popularity of different technologies and toolkits.

2.2 Environmental Information System and NJPIES

NJPIES (New Jersey Program for Information Ecology and Sustainability) is associated with holistic approach to environmental data collection, compilation, integration and provision that puts people, not technology, at the center of the environmental information world. NJIT intends to create through NJPIES the Virtual Environmental Enterprise (VEE) at New Jersey Institute of Technology (NJIT). NJPIES is a program dedicated to improving access to environmental information and increasing its usefulness through the advanced computer tools at New Jersey Institute of Technology, Newark, New Jersey. Dr. Marcus Healey, Dr. Franz Kurfess and Dr. Jason Wang are the key persons involved with this program. One can use services offered by NJPIES at http://njpies.njit.edu (Figure: 1)

The objectives of NJPIES are:

1. Improve access to Environmental Information through development and implementation of advanced computer based search tools.
2. Inform academic, industrial, government and individual users about efficient and effective ways to access and apply these environmental information resources for increased understanding, better approaches to addressing

environmental problems and a more effective ability to predict future environmental challenges and opportunities.

3. Train individuals in Information Ecology and Sustainability.

NJPIES intends to provide following products to the user community. Some of them are already existing whereas remaining are in their prototype phase: [NJPIES]

NJ Enviro Databases

Application includes information on air pollution, toxic releases, hazardous waste, and water discharge permits. Through NJ EnviroDB database, you can get lists of which facilities in your neighborhood are releasing pollutants.

EnviroDaemon

Its a Search Engine which puts your search for Pollution Prevention information on the right track.

Shopping Cart For Pollution Prevention Documents

The idea of this prototype is to make the research papers, industry information, technical studies etc. easily available to the industry, researchers and students through the World Wide Web.

Environmental Lifecycle Cost Analysis Model (ELCAM)

The ELCAM application tool attempts to identify, measure, and quantify social costs of human activities such as manufacturing that are not considered with traditional accounting systems. The application developed will quantify, monetize, and rank the damage or external costs to the environment of certain types of emissions.

Conversion Tool

The conversion tool is a UNIX based application that converts flat files to Oracle format. The tool is used to move data from a flat file into an Oracle table. Once the data is in Oracle, it can be segmented and queried as needed by the user. Having files stored in an orderly fashion always improves readability and accessibility.

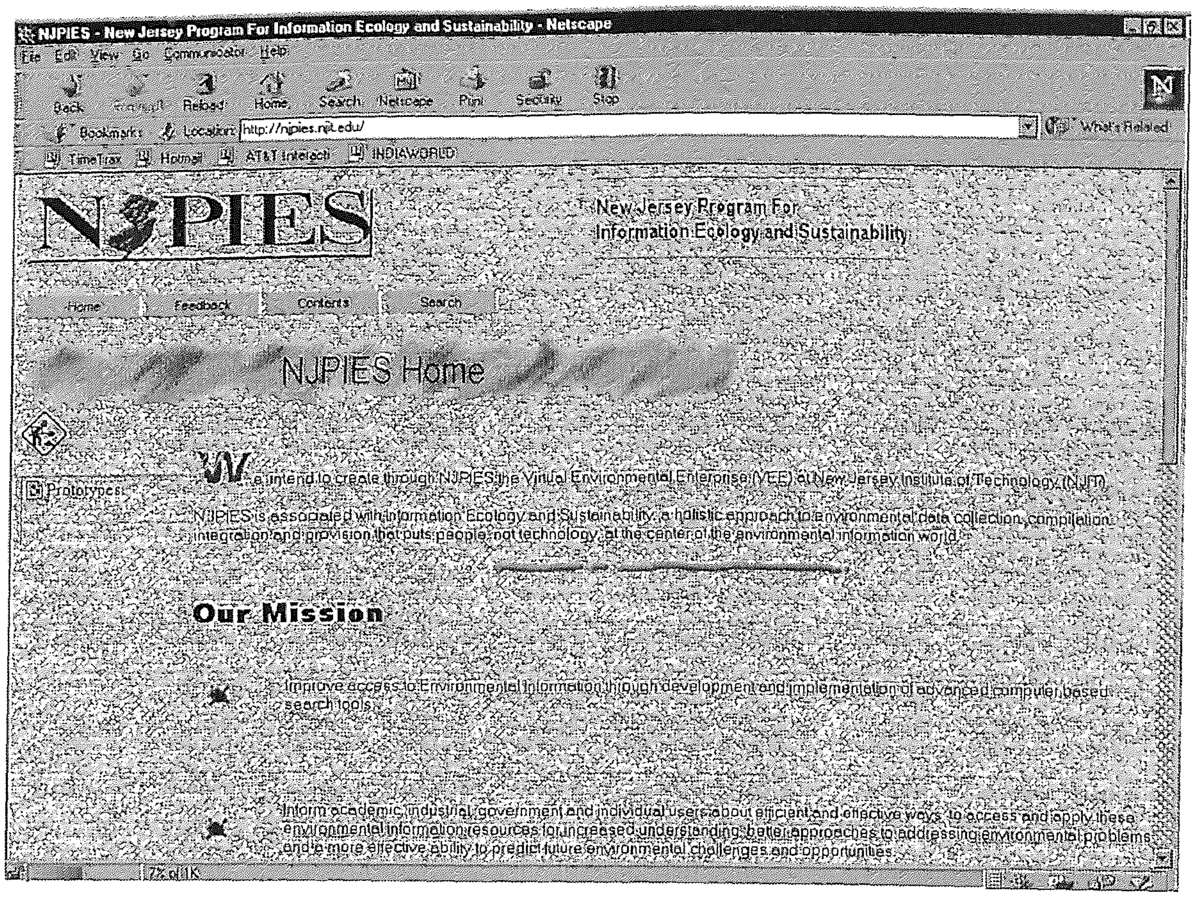
Fate-Transport Expert System Utility

This Utility uses if-then rules based on chemical constants to determine probable fate-transport in the environment.

From the product offerings of the NJPIES, it is obvious that NJPIES intends to become large collector of environment related documents. The current focus is a repository of all possible documents related to "New Environment Wave" which includes following topics.

- Pollution Prevention
- Life Cycle Analysis
- Industrial Ecology
- Environmental Accounting

To collect these documents, articles, technical studies and research papers on "New Environment Wave", some kind of intelligent utility is required which can visit different URLs (Uniform Resource Locators), analyze the content of those URLs and if something interesting related to pollution prevention, life cycle analysis, etc. found on those sites, that program should be collecting that document or article for NJPIES database. Same actions will be very costly if done using humans working on these activities. Further details about the needs of such a program and proposed solution is available in chapter 4.


NJPIES - New Jersey Program For Information Ecology and Sustainability - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Stop
Bookmarks Location: http://njpies.njit.edu/
What's Related
TimeTrax Hotmail AT&T Intelecti INDIAWORLD
NJPIES
New Jersey Program For Information Ecology and Sustainability
Home Feedback Contents Search
NJPIES Home
Prototypes
7% of 1K

We intend to create through NJPIES the Virtual Environmental Enterprise (VEE) at New Jersey Institute of Technology (NJIT).

NJPIES is associated with Information Ecology and Sustainability, a holistic approach to environmental data collection, compilation, integration and provision that puts people, not technology, at the center of the environmental information world.

Our Mission

- Improve access to Environmental Information through development and implementation of advanced computer based search tools.
- Inform academic, industrial, government and individual users about efficient and effective ways to access and apply these environmental information resources for increased understanding, better approaches to addressing environmental problems and a more effective ability to predict future environmental challenges and opportunities.

Figure 1: The Home Page of NJPIES [NJPIES]

CHAPTER 3

AGENTS AND THEIR APPLICATION IN INSM

This chapter addresses agents, related technologies etc. in detail. Properties attached with an agent program which make it different than a normal program are listed in following section. Some of the important toolkits which are popular to write agent kind of programs are also described. IBM's ASDK (Aglet Software Development Kit [IBM_AGLET]) and General Magic's Telescript are the main ones. KQML (Knowledge Query and Manipulation Language [UMBC_KQML]) is another standard based technology which facilitates integrating intelligence or knowledge in the agent program. AOP (Agent Oriented Programming [Shoham92]) standard is another technology which makes it easier for developers to put different aspects of agent systems in to programming.

Klaus's and Felip's work [Klaus97], where we were trying to put Intelligent agents in INSM (Integrated Network and System Management) is also described as a separate section.

3.1 Agents - A Broad View

Agents are introduced briefly in the earlier chapter so here we will deal with them in a broader context. An autonomy is desired property of an agent program. An autonomous agent is a system situated within and a part of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to effect what it senses in the future. Autonomous agents are situated in some environment. Change the environment and we may no longer have an agent. A robot with only visual sensors in an environment without light is not an agent. Systems are agents or not with respect to some environment. The AIMA agent discussed earlier requires that an agent "can be viewed" as sensing and acting in an environment, that is, there must exist an environment in which it is an agent. For example, a payroll program in a real world environment could be said to sense the world via it's input and act on it via its output, but is not an agent because its output would not normally effect what it senses later. A payroll program also fails the "over time" test of temporal continuity. It runs once and then goes into a coma, waiting to be called again. Most ordinary programs are ruled out by one or both of these conditions, regardless of how we stretch to define a suitable environment. All software agents are programs, but not all programs are agents. In other words, a normal program is generally a set of pre-programmed actions, usually to be executed in a linear, deterministic fashion. they tend to be written in procedural languages - a procedure is generally a directive which assumes some preconditions, performs an action, and returns a result. Procedural languages are fine for many tasks, especially those which have a well-defined set of solutions. Unfortunately, they do not scale well to problems involving distributed interaction such as occurs among multiple negotiating agents. In such situations the representation of contingencies, plans and goals using a declarative language tends to be a more effective approach. Agents are non-deterministic automatons which can employ anthropomorphic attributes such as emotion, belief, and commitment. Some agents even emulate real people! Given the same set of conditions and the same goal, a script will always produce the same result. An agent, on the other hand, may produce different results based on its current emotional state, beliefs, and so on. Agents also learn and adapt to new situations in a manner that clever scripts can only approximate.

Following are some of the important properties which agents possess which are not seen within normal programs.

- **Autonomy:** agents operate without the direct control or intervention of humans or others, and have some kind of control over their actions and internal state;
- **Social ability:** agents interact with other agents and sometimes with humans via some kind of Agent Communication Language.
- **Reactivity:** agents perceive their environment (which can be a physical world, Internet, a user through GUI, a collection of other agents or may be combination of some of these entities), and respond in a timely fashion to changes that occur in this environment.
- **Goal orientedness:** an agent is capable of handling complex tasks. They are more goal oriented Vs result oriented.
- **Mobility:** the ability of an agent program to roam around in the network. Agent also travels sometimes to other machines in order to achieve the goal.
- **Collaboration:** Sometimes user might give instruction in ambiguous manner or may make mistakes while instructing agents. Agents do not unthinkingly accept all such instructions. Agent is capable of asking questions to users and also sometimes they refuse to execute certain tasks.

The characteristics listed above are common to most of the agents but at the same time it is not always necessary to have all these properties existing in a agent. If an agent is helping a user about how to operate a PC, it might not need to travel to another computer in a network to do its job and hence mobility property can be missing from it. What makes an agent "intelligent" is very hard to define. Intelligence is the degree of reasoning and learned behavior: the agent's ability to accept the user's statement of goals and carry out the task delegated to it. At a minimum, there can be some statement of preferences perhaps in the form of rules, with an inference engine or some other reasoning mechanism. Higher levels of intelligence will involve a user model. Further on the scale will be systems that learn and adapt to their changing environment.

Suitability of agents in different applications:

In [IBM95] eight application areas are identified where now (or in the near-future) agent technology is (or will be) used.

These areas are:

1. Systems and Network Management:

Systems and network management is one of the earliest application areas to be enhanced using intelligent agent technology. The movement to client/server computing has intensified the complexity of systems being managed, especially in the area of LANs, and as network centric computing becomes more prevalent, this complexity further escalates. Users in this area (primarily operators and system administrators) need greatly simplified management, in the face of rising complexity. Agent architectures have existed in the systems and network management area for some time, but these agents are generally "fixed function" rather than intelligent agents. However, intelligent agents can be used to enhance systems management software. For example, they can help filter and take automatic actions at a higher level of abstraction, and can even be used to detect and react to patterns in system behavior. Further, they can be used to manage large configurations dynamically. Klaus and Felip tried to use Agents in similar application which monitors distributed applications in distributed environment. Mobile agents collect process and communication load level details from the managed hosts and will provide them to system and network administrators on a client system.

2. Mobile Access / Management:

As computing becomes more pervasive and network centric computing shifts the focus from the desktop to the network, users want to be more mobile. Not only do they want to access network resources from any location, they want to access those resources despite bandwidth limitations of mobile technology such as wireless communication, and despite network volatility.

Intelligent agents which (in this case) reside in the network rather than on the users' personal computers, can address these needs by persistently carrying out user requests despite network disturbances. In addition, agents can process data at its source and ship only compressed answers to the user, rather than overwhelming the network with large amounts of unprocessed data;

3. Mail and Messaging:

Messaging software (such a software for e-mail) has existed for some time, and is also an area where intelligent agent function is currently being used. Users today want the ability to automatically prioritize and organize their e-mail, and in the future, they would like to do even more automatically, such as addressing mail by organizational function rather than by person. Intelligent agents can facilitate all these functions by allowing mail handling rules to be specified ahead of time, and letting intelligent agents operate on behalf of the user according to those rules. Usually it is also possible (or at least it will be) to have agents deduce these rules by observing a user's behavior and trying to find patterns in it.

4. Information Access and Management:

Information access and management is an area of great activity, given the rise in popularity of the Internet and the explosion of data available to users. It is the application area that this thesis will mainly focus on. Here, intelligent agents are helping users not only with search and filtering, but also with categorization, prioritization, selective dissemination, annotation, and (collaborative) sharing of information and documents;

5. Collaboration:

Collaboration is a fast-growing area in which users work together on shared documents, using personal video-conferencing, or sharing additional resources through the network. One common denominator is shared resources; another is teamwork. Both of these are driven and supported by the move to network centric computing. Not only do users in this area need an infrastructure that will allow robust, scaleable sharing of data and computing resources, they also need other functions to help them actually build and manage collaborative teams of people, and manage their work products. One of the most popular and most heard-of examples of such an application is the groupware packet called Lotus Notes.

6. Workflow and Administrative Management:

Administrative management includes both workflow management and areas such as computer/telephony integration, where processes are defined and then automated. In these areas, users need not only to make processes more efficient, but also to reduce the cost of human agents. Much as in the messaging area, intelligent agents can be used to ascertain, then automate user wishes or business processes.

7. Electronic Commerce:

Electronic commerce is a growing area fueled by the popularity of the Internet. Buyers need to find sellers of products and services, they need to find product information (including technical specifications, viable configurations, etc.) that solve their problem, and they need to obtain expert advice both prior to the purchase and for service and support afterward. Sellers need to find buyers and they need to provide expert advice about their product or service as well as customer service and support. Both buyers and sellers need to automate handling of their "electronic financial affairs". Intelligent agents can assist in electronic commerce in a number of ways. Agents can "go shopping" for a user, taking specifications and returning with recommendations of purchases which meet those specifications. They can act as "salespeople" for sellers by providing product or service sales advice, and they can help troubleshoot customer problems.

8. Adaptive User Interfaces:

Although the user interface was transformed by the advent of graphical user interfaces (GUIs), for many, computers remain difficult to learn and use. As capabilities and applications of computers improve, the user interface needs to accommodate the increase in complexity. As user populations grow and diversify, computer interfaces need to learn user habits and preferences and adapt to individuals. Intelligent agents (called interface agents) can help with both these problems. Intelligent agent technology allows systems to monitor the user's actions, develop models of user abilities, and automatically help out when problems arise. When combined with speech technology, intelligent agents enable computer interfaces to become more human or more "social" when interacting with human users.

3.2 Agent Technologies

Aglets and Aglets Software Development Kit

As I discussed in Section 3.1, Java is now preferred language for writing mobile agent applications. Lot of tools are available in industry which provide framework to develop mobile agents. IBM's Aglet Software Development Kit (ASDK) [IBM_AGLET] is major one which has become popular in industry. ASDK is the product of IBM's Tokyo Research Lab. Before describing ASDK in detail, we need to know what aglets are. The aglet is the next step in the evolution of executable content on the Internet, introducing a program code that can be transported along with state information. Aglets are Java objects that can move from one host on the Internet to another. That is, an aglet that executes on one host can suddenly halt execution, dispatch itself to a remote host, and resume execution there.

Aglet Vs Applet: Applet is another popular technology in which programs (executable content) are transferred from server to client system in form of a class and then that applet is executed within the framework of a browser which downloaded that applet. Browsers normally use "ClassLoader" methods to get these referenced classes whenever they come across <APPLET> tag in the HTML document. Applets have some restrictions though about their capabilities. Applets normally can not access local file systems and they can not even open network connections to any other machine except the server from where they were downloaded. Applets are transferred over the Internet in form of classes and then they get executed on client machines. Aglets are having advantages that they can also transfer their state along with their data. This way an aglet can execute on one machine and then can stop executing on that machine and then resume its execution on another machine from the point where they had left their execution on earlier machine. This is the same characteristics which we need in mobile agents. Mobile agents need to traverse in the network to achieve their goals. Mobile agents normally also visits different systems (places in terms of General Magic's telescript) and meet other visiting agents over there where they exchange information in order to achieve their individual goals. Aglets by that nature provides these capabilities. In our work of monitoring distributed applications, we used Aglets as our mobile agents which travel to managed hosts and collect process information from those machines and then they send that information to originating machine in form of messages. The way applet needs some kind of context (browser or applet viewer), similarly Aglet also needs special context in which they can do their job. This context is provided by Aglet Software Development Kit which needs to be existing on all systems where aglets intend to travel.

Aglets Software Development Kit

ASDK is a special toolkit developed at Tokyo Research Laboratory of IBM Corp. and it provides environment for programming mobile internet agents in Java language. ASDK itself is developed using 100% Java language so this kit is platform independent. One can download this kit from IBM's URL http://www.trl.ibm.co.jp/aglets/ [IBM_AGLET].

When one downloads this kit, appropriate user manuals and sample programs (aglets) come with the kit. The ASDK from IBM also offers a first-of-its-kind visual environment for building network-based applications using mobile agents. ASDK includes the following packages:

- The Aglets Framework for mobile Java agents
- The Agent Transfer Protocol (ATP)
- JDBC [SUN_JDBC]
- Tahiti - a visual agent manager
- Fiji - an agent Web launcher

Aglets Framework for mobile Java agents: ASDK provides users with the Java Aglet API (J-AAPI) to develop aglet based applications. The Java Aglet Application Programming Interface (J-AAPI) is a standard for interfacing aglets and their environment. J-AAPI defines the methods necessary for aglet creation, message handling in the aglet, initialization, dispatching, retraction, deactivation/activation, cloning, and disposing of the aglet. J-AAPI is simple, flexible, and stable. Internet agent developers can write platform independent aglets and expect them to run on any host that supports J-AAPI [Lange98].

Fiji - an agent Web launcher: Fiji is a Java applet; based on the Aglets Framework and therefore capable of creating an aglet or retracting an existing aglet into the client's Web browser. The Fiji applet simply takes an agent URL as its parameter and can easily be embedded in a Web page by using HTML, like any Java applet. And just as with other applets, all the required software will be dynamically downloaded to the browser as it is needed. The client will not have to explicitly download and install Aglets Workbench in order to use your agents. This is a way one can empower Web pages with mobile agents.

Tahiti - a visual agent manager: Tahiti is a visual agent manager based on the Aglets Framework. Tahiti uses a unique graphical user interface to monitor and control aglets executing on your computer. Tahiti's User friendly menu provides following options:

Aglet : for handling aglets

Create... : Create an aglet.

Dialog... : Sends a request to an aglet to open its dialog panel.

Dispose... : Destroys the agent.

Clone... : Make a copy of the agent.

AgletInfo : Shows the properties of the agent.

Kill : Destroy the agent without calling onDisposing().

Exit : Shutdown the server.

Mobility :

Dispatch... : Send the aglet to another server.

Retract... : Retract a dispatched aglet from another server.

Deactivate... : Deactivate the aglet with time.

Activate... : Activate a deactivated aglet

View:

Options:

Tools:

Help:

Sample Aglet Program:

ASDK comes with following sample aglet programs along with the source code.

- examples.hello (HelloAglet.java)

  HelloAglet is a mobile aglet that goes to a remote host to say "Hello World" and then returns home and dies after displaying a message.

- examples.simple (DisplayAglet.java)
- examples.server (ServerApp.java)
- examples.client (WatcherClient.java)
- examples.talk (TalkMaster.java, TalkSlave.java, TalkWindow.java)

Following is a sample aglet program which consists of basic required methods. Users can change those methods and add their own in order to write their own agents.

```
import aglet.*;
public class typicalAglet extends aglet.Aglet {
  private String anInstanceVariable = "I aint born yet man!";
  public typicalAglet(){
    System.out.println("Entering typicalAglet.typicalAglet()");
    System.out.println("Leaving typicalAglet.typicalAglet()");
  }
  public void onCreation (Object init) {
System.out.println("Entering typicatAglet.onCreation(..)");
System.out.println("anInstanceVariable isnow:"+anInstanceVariable);
     System.out.println("Weee, I am being created!!");
```

```
        anInstanceVariable = "Jumpin!";
        System.out.println("Leaving typicatAglet.onCreation(..)");
    }//EOF onCreation
    public void run() {
       System.out.println("Entering  typicatAglet.run()");  System.out.println("Where  is
theparty!!");
       System.out.println("anInstanceVariable is now: "+anInstanceVariable);
       System.out.println("Leaving typicatAglet.run()");
    }//EOF run()
}//EOF class
```

There are three methods that are looked for in an aglet when the Context creates it:

1. anAglet() which is the constructor,
2. void onCreation() which is a method which is called by the context as soon as the constructor has finished its work
3. Finally, it calls public void run() at which point the aglet is autonomous, which means what it can go about doing it was programmed to do.

Once user writes an aglet program, following steps need to be followed on PCs with Windows environment in order to compile and execute that program.

1. Define your aglet to be included in the package my.aglets. (my.aglets is a package name for example.)

2. Define a corresponding subdirectory path my\aglets under a directory DIR (possibly the original c:\.....Aglets1.0.3\public).
3. Include DIR in AGLETS_PATH and AGLETS_EXPORT_PATH (if it is not there already). Note: There is NO NEED TO MODIFY the classpath directory.
4. Compile your aglet files into DIR\my\aglets, using the javac "-d" and "classpath" options. Learn more about packages and javac, if necessary.

   As the result of compile, let us assume, you have a file newAglet.class
5. Run Tahiti and create your aglets. Use my.aglets.newAglet as the aglet name in the create aglet panel.

More precise descriptions is available in the Installation Guide.

ABE (Agent Building Environment) by IBM:

IBM also has released a new version of their Agent Building Environment - a toolkit for software developers that makes it easy to build an application based on agents, or to add them to an existing application. In the alpha version, the intelligent agent watches for a certain condition, decides what to do based on the rules you've given it, and triggers an action as a result. This developer kit comes with a number of pre-built parts which make it easy for you to add agent technology to applications. The "central intelligence" brain or the agent is based on reasoning engine and adapter technologies from IBM's T.J. Watson Research Lab. "Adapters" or interfaces allow the agent to interact with the rest of the world. The HTTP adapter, for instance, interfaces with the world-wide-web. The NNTP adapter interfaces with Internet USENET news services, and the timer adapter allows events to be triggered based on time. One can write his own adapters, as well, and guidelines and a sample adapter are provided as a part of that kit. Custom adapters can be written in either C++ or Java. A simple full-screen interface is also provided to allow the user to specify the rules for the agent's behavior.

Ara :

"Agents for Remote Action" is a platform for the portable and secure execution of mobile agents currently under development at the University of Kaiserslautern. Mobile agents in this sense are programs with the ability to change their host machine during execution while preserving their internal state. This enables them to handle interactions locally which otherwise had to be performed remotely. Ara's specific aim in comparison to similar platforms is to provide full mobile agent functionality while retaining as much as possible of established programming models and languages. Various interpreted programming languages like Tcl, Java can be adapted to Ara, making them usable for mobile agent programming. Ara is intended as a general system platform on top of which specific applications such as information mining, mobile device support, etc. can be built. Version 1.0 alpha of the Ara platform (for Solaris, Linux and SunOS) has been released free for non-commercial purposes, including the complete source code, extensive documentation, and a number of example agents.

ATP:

Agent Transfer Protocol is an application-level standard protocol for distributed agent-based information systems. Aimed at the Internet and using Universal Resource Locators (URL) for agent resource location, ATP offers a uniform and platform-independent protocol for transferring agents between networked computers. While mobile agents may be written in many different languages and for a variety of vendor-specific agent systems, ATP offers the opportunity to handle agent mobility in a general and uniform way. For example, any agent host machine will have a single and unique name independent of the set of vendor-specific agent systems it supports. ATP also provides a uniform agent transport mechanism and allows a standard agent query facility to be used throughout the network.

AgenTalk:

AgenTalk is a coordination protocol description language for multi-agent systems. In the distributed artificial intelligence area, many coordination protocols such as the contract net protocol have been proposed, and many application-specific protocols will be required as more software agents start to be built. AgenTalk allows coordination protocols to be defined incrementally and to be easily customized to suit application domains by incorporating an inheritance mechanism. AgenTalk is being co-developed by NTT Communication Science Laboratories and Ishida Laboratory, Department of Information Science, Kyoto University. The Agent-oriented Programming project aims to develop a new programming paradigm that exploits the computational advantages of attributing mental states to machines, employs ideas from speech act theory, and relies on computational versions of social laws.

AOP:

The Center for the Study of Language and Information (CSLI) is an Independent Research Center founded in 1983 by researchers from Stanford University, SRI International, and Xerox PARC to further research and development of integrated theories of language, information, and computation. AOP (Agent-oriented Programming) is a part of it and is a programming paradigm in which computer agents are assumed to have a formal version of mental states---states which dictate agents' actions and which are affected by messages they receive. Agent-oriented programming [Shoham92] provides a new approach to programming distributed systems, emphasizing explicit representation of time, beliefs, and commitments, including speech-act-like communicative commands. At CSLI (Center for the Study of Language and Information), they work on the theoretical investigation of agents with mental states, interpreter design and implementation, and applications such as information retrieval, personal software assistants, and robotics. They are also investigating the compilation of such a language into a neutral, agentless process language.

KQML:

KQML or the Knowledge Query and Manipulation Language is a language and protocol for exchanging information and knowledge. It is part of a larger effort, the ARPA Knowledge Sharing Effort which is aimed at developing techniques and methodology for building large-scale knowledge bases which are sharable and reusable. KQML is both a message format and a message-handling protocol to support run-time knowledge sharing among agents. KQML can be used as a language for an application program to interact with an intelligent system or for two or more intelligent systems to share knowledge in support of cooperative problem solving. In short, it is an agent communication language that is used in systems ranging from research experimental systems to real business production systems [UMBC_KQML].

KQML focuses on an extensible set of performatives, which defines the permissible operations that agents may attempt on each other's knowledge and goal stores. The performatives comprise a substrate on which to develop higher-level models of inter-agent interaction such as contract nets and negotiation. In addition, KQML provides a basic architecture for knowledge sharing through a special class of agent called communication facilitators which coordinate the interactions of other agents The ideas which underlie the evolving design of KQML are currently being explored through experimental prototype systems which are being used to support several testbeds in such areas as concurrent engineering, intelligent design and intelligent planning and scheduling.

JKQML:

JKQML [ALPHA_JKQML] is a framework and API for constructing Java-based, KQML-speaking software agents that communicate over the Internet. JKQML allows the exchange of information and services between software systems, creating loosely coupled distributed systems. Common attributes of agents include reactivity, autonomy, collaborative behavior, communication ability, mobility, and so on. Software agents must communicate with other agents in order to work flexibly and autonomously. JKQML provides flexibility for the extension of the framework, and it supports the following three protocols: [ALPHA_JKQML]

- KTP (KQML transfer protocol): a socket-based transport protocol for a KQML message represented in ASCII.
- ATP (agent transfer protocol): a protocol for KQML messages transferred by a mobile agent that is implemented by Aglets.
- OTP (object transfer protocol): a transfer protocol for Java objects that are contained in a KQML message.

JKQML is based on the 1996 proposal for a new KQML specification.

3.3 Monitoring Distributed Applications with Intelligent Agents

3.3.1 Background

In Fall 1997 semester, Felip Miralles and Klaus Holthaus did a project with a goal of developing some prototype which will monitor distributed processes in a distributed environment. In a distributed environment different heterogeneous computers act together and perform a complex functionality as a unit. Felip and Klaus proposed a platform independent application to monitor the distribution of processes of an application or a set of applications among different processors in a distributed environment. That application will be a tool intended for system administrators and will be helpful to optimize the distribution of different processes in different processors.

The idea was to utilize intelligent agent technology to perform tasks for this purpose. They proposed a distributed architecture consisting of Java applets, utilizing IBM's Java Aglets API (Aglet Workbench (AWB) which is now known as Aglets Software Development Kit (ASDK)) and intelligent agent technology provided by IBM's Agent Building Environment (ABE).

Key objectives of that project were :

- Explore the technology of mobile and intelligent agents
- Examine the application of that technology to the management of distributed environment.
- Develop, implement and test a prototypical agent architecture for monitoring distributed processes

- Evaluation of this prototype with regard to issues such as performance. portability and scalability.

In the beginning, I started working on this unfinished project [Klaus97]. I spent lot of my time learning the required skills and technology. Here I am trying to explain that work briefly. With the guidance of Dr. Franz Kurfess, I also wrote a paper which recently got published in IEEE Computer Society organized ECBS (Engineering of Computer Based Systems) 99 Conference and Workshop. I also got an opportunity to present this paper in that conference. Remaining section describes that work in detail.

3.3.2 Intelligent Agents for Systems Management

Our idea is to utilize intelligent agent technology as an integrated architecture for INSM. This distributed architecture will consist of Java applets and applications, utilizing the following frameworks for intelligent agent technology:

Mobile agents, implemented in IBM's Java Aglets API (Aglet Workbench, now known as ASDK - Aglets Software Development Kit).

Intelligent agent technology, provided by IBM's Agent Building Environment.

There are several reasons why we think intelligent agents are useful for our application area:

1. Networks are distributed, systems are distributed - therefore, management has to be distributed as well.
2. Considering the growth of networks and the number of end systems in companies, it would be a great relief for administration staff to have a management architecture that

performs tasks in parallel and autonomously, which is exactly what a herd of intelligent agents roaming the company's networks could do.

3. Networks and systems are arbitrarily heterogeneous - up to the point where they support Java, which is to some extent the language of choice for agents.
4. Management policies and goals are most crucial for network and systems management - agents provide the mechanisms to implement these goals more directly and still more flexibly than ever before, due to the intelligent capabilities of agents.

3.3.2.1 Design

Task Specification

Since our experiment is concerned with distributed applications monitoring as a specific subtask of INSM, an agent serving this purpose would use its sensors to monitor CPU load and network communication of distributed processes. Its effectors would relocate those processes based on its goals, e. g. the premise that communication between two processes on different hosts may not exceed a certain limit. The intention of our experiment is to provide the monitoring part of the above task by retrieving information about processes and displaying that information in a graphical way. By doing this our tool gives a systems administrator the information needed to optimize the performance, network load, etc. of distributed applications.

Following information per process and per processor needs to be retrieved as a basic set:

Processor = IP address, host name, idle time, communication time, I/O time, processing time

Process = ID#, process name, type (thread / process), scheduling time, sleeping time, Kernel- running time, Process-running time

Both will be implemented as classes. Also, each processor object will contain a list of the above described process objects. Besides that, another class Communication will represent communication load between two different processes:

Communication = Process1 of Processor1, Process2 of Processor2, Communication load

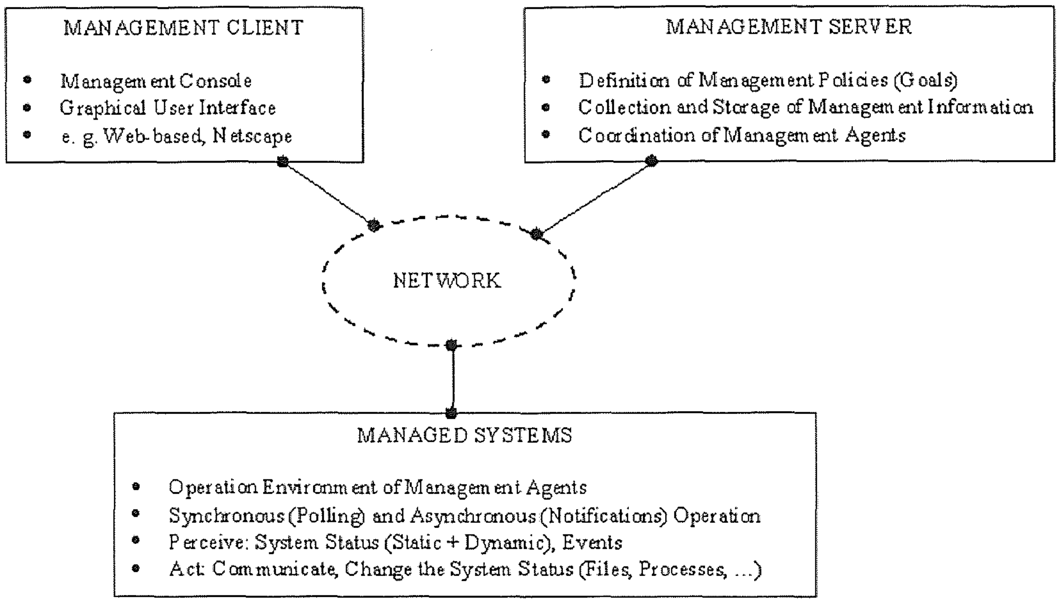

MANAGEMENT CLIENT
Management Console
Graphical User Interface
e. g. Web-based, Netscape
MANAGEMENT SERVER
Definition of Management Policies (Goals)
Collection and Storage of Management Information
Coordination of Management Agents
NETWORK
MANAGED SYSTEMS
Operation Environment of Management Agents
Synchronous (Polling) and Asynchronous (Notifications) Operation
Perceive: System Status (Static + Dynamic), Events
Act: Communicate, Change the System Status (Files, Processes, ...)

Figure 2: General Architecture [Klaus97]

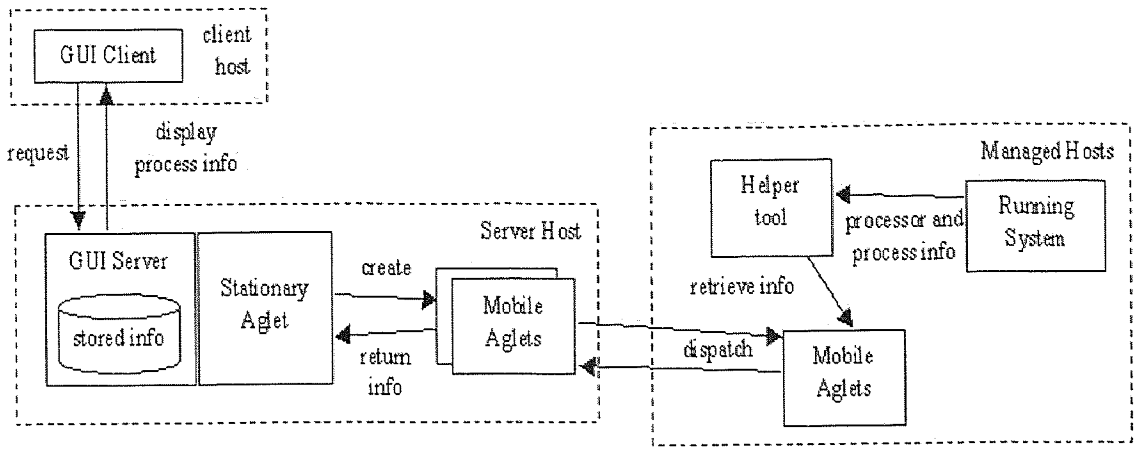


Figure 3: System Components [Klaus97]

The General Architecture

On one hand INSM architectures need to be distributed; on the other hand however it will be necessary to have a certain degree of centralization: Agents that access managed systems are restricted in terms of size and CPU usage, therefore they can only contain the very essential functionality needed at the point of action. Another aspect in this matter is not to overload the network by transferring oversized agents from system to system, thus wasting bandwidth and CPU time of the systems to be managed. Besides that, the herd of agents roaming a company's systems must be coordinated, information must be stored, and the systems administrator responsible for the systems management function must have versatile access to what the agents are doing, e. g. to prevent serious disasters caused by malfunctioning agents, which requires some degree of centralization as well. The overall architecture of our system is shown in Figure 2.

The server component is stationary, dispatching agents, collecting information and persistently storing the current status of all systems and networks. (For the network management community in the Internet, this could be compared to the Management Information Base (MIB), being accessed via SNMP. Another task of the server component is to provide access to the GUI client when launched from a (remote) machine. The systems administrator should be able to access the management system from any machine that provides Web access and supports Java, thus the client component is a Java applet being launched remotely and connecting back to the server. The intelligent agents are dispatched by the server and access the systems that are to be managed. The management server defines the goals for the agents (management policies, see above); the individual agents should have enough intelligence to achieve these goals without any further direct control. The agents should even be able to explore the systems in the network on their own (auto-discovery is an essential function of today's network management systems), monitor them autonomously, classify events into marginal or relevant, and take appropriate action, in order to achieve the goals that were set before. All this happens (for the most part) without interference and maybe not even notion of either users or system administrators. An agent only initiates notification or interaction with either one when situations occur that can not be handled in an automated way.

The Components of the Architecture

A more detailed view of the system is shown in Figure 3. It shows the essential components together with their interactions.

The GUI client: The applet provides menu choices to open and save process configurations stored in files that were created either by the monitoring system itself or by the user who changed the process distribution for simulation purposes. The most important option is the one that allows beginning the analysis of an application or set of applications running on a set of different of processors. The monitoring is done off-line and the results are stored in a file as mentioned above. Once such a file is opened, the user can display the information in either the distribution window, showing the processes in every processor and the communication between processes, or in the cost window, showing the utilization of the different processors (processing, I/O, communication), and the portion of these times dedicated to the application(s) in question. The cost window will show the time every processor is idle and the network traffic between processors. Figure 4 shows a distribution window where circles represent different processors (systems) and a single process is shown as a filled square whereas thread is shown as a hollow square. When pointing with the cursor on a process or a thread the information of the execution features of that specific process or thread is displayed. Communications between processes are depicted with arrows, with different widths representing the amount of network traffic that is generated.

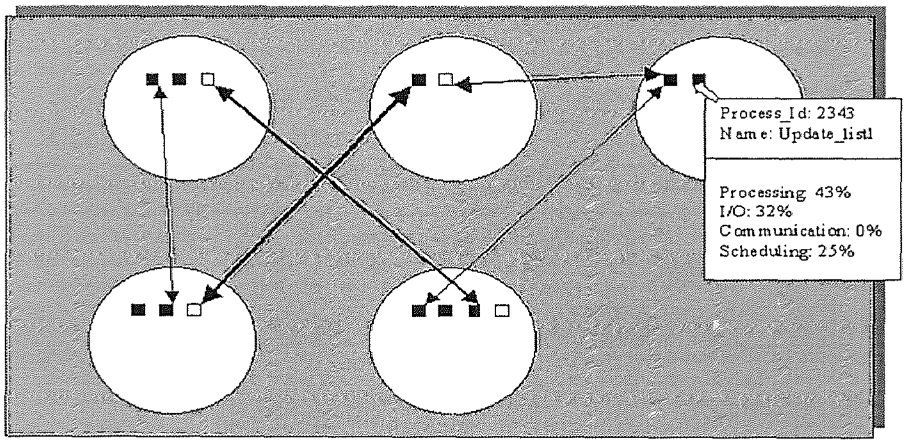


Figure 4: Distribution Window [Klaus97]

The Server Component: This component provides access to the GUI applet. Once the applet is launched, it connects back to the server and starts the monitoring request formulated in the monitoring options dialog. The design of the server component also includes the inference engine / rule base that provides intelligence to our architecture. This can be accomplished using IBM's Agent Building Environment (ABE): A rule-based forward chaining inference engine. Rules are stored in the KIF format and can be edited via a graphical rule editor, besides direct ASCII editing. A number of adapters, some are provided, some can be written in C++ or Java, provides sensor / effector capabilities to the inference engine and actually start the inferencing process. The user can write adapters himself, possibly in Java. The server basically provides a "home base" for the mobile agents and stores the information they gathered.

The Mobile Agents: The server has an important component that provides the ability to fetch the necessary information about processors, processes and communications in the specified remote hosts. This component is based on mobile agents technology in form of mobile Java Aglets, using IBM's JAAPI, the Aglets Software Development Kit. Aglets are little Java applets that are able to autonomously travel over the network, clone themselves, create other aglets, send messages to other aglets, serialize themselves for persistent storage, etc. While they travel they maintain their state of execution and resume execution upon arrival. This component of the server has the capacity to create aglets, dispatch them and communicate with them. In our case these aglets are dispatched to remote hosts in order to retrieve information about processes / applications that are running on them. They send messages back to their server component containing the information they retrieved. There will be two types of agents, a stationary one, that communicates with the rest of the server application and manages the other type, the mobile agents that actually travel to the managed hosts.

The Helper Tool: We still need an additional 'helper tool' that is installed locally on each managed host and retrieves information about processes. It provides cumulative / average values that can be picked up by the incoming aglet. Thus the aglet will not be concerned by the specific system platform of the host it was dispatched to, increasing their platform-independence. Only the helper tool is system-dependent. Using this tool can be considered a work-around for the problem that Java does not provide access to the "guts" of a system as we would have needed.

3.3.2.2 Implementation

Working Environment

We have implemented the server component on a UNIX platform. The client component can be launched from any computer that supports Java. The Aglets Workbench is entirely written in Java and is thus platform-independent. Each monitored host needs to have a copy of the Aglets Workbench installed and the Aglets daemon running in order to allow aglets to reside on them. Applets are a peculiar kind of programs as they are only intended to be executed in the context of an HTML file. This places some rather severe restrictions on what you can do in an applet to protect the environment in which they are executed. The process of storing and retrieving objects in an external file is called serialization. Writing an object to a file is referred to as serializing the object, and reading an object from a file is called deserializing an object. The way that components are arranged in a container is usually determined by an object called a layout manager. This layout manager for a container determines the position and size of all the components in the container. The Vector class defines a collection of elements of type Object that works rather like an array, but with the additional feature that it can grow itself automatically when you need more capacity. Because it stores elements of type Object, and Object is a superclass of every object, you can store any type of object in a Vector. This also means that you can use a single Vector object to store objects that are instances of a variety of different classes. We have not used this tremendously flexible feature.

The Back End –Concepts and Technologies

The back end of our architecture consists of two components:

Mobile Java Agents, which provide remote access to the processors on which monitoring is to be performed

Local tool(s) which are necessary to gather the low level information from inside the running system

We implemented the mobile part of our architecture by using IBM's Java Agent API, ASDK. Aglets are a concept built on Java Applets by extending applet capabilities to create a somewhat autonomous behavior, which is what makes them appear as software agents. The core of the architecture is the Aglet Transfer Protocol (ATP), provided by the aglets daemon (agletsd), that has to be operational on every host that is supposed to utilize aglets. These actions are facilitated via Java RMI (Remote Method Invocation) and Java's Object Serialization. The activities of an aglet are mainly implemented in methods that are derived from the Aglet base class and that form the lifecycle of an aglet. E. g. when created, an aglet's OnCreation method is invoked, and when dispatched to a remote context, first the OnDispatching method is invoked, then the aglet is transferred, and as soon it arrives, the OnArrival method is invoked. The overall concept is a callback model: Upon certain events the ATP daemon calls the appropriate methods in the aglets classes provided by the programmer. The autonomous behavior of aglets is created by having aglets perform these actions on themselves or other aglets. An aglet context provides proxies to aglets by identifying a particular aglet with a unique aglet ID. Once these "handles" are given, aglets can perform the above actions on any aglet they have access to. Since aglets are basically given all the functionality that can be implemented with Java, security becomes a very important issue when implementing actions that a host allows an aglet to perform. Along with the ATP daemon comes a GUI to manage aglets interactively, called Tahiti, in which security options can be set. Aglets are distinguished into trusted and untrusted aglets, and each of them is given selective read or read/write access to the host's file system. The ATP uses a new type of URL to locate an aglet in a context: atp://some.host.name:<port> . The code base of an aglet (that is the compiled class code of the aglet) is supposed to be located in a directory within the path variable AGLET_PATH, expecting a Java package tree in this directory. Usually AGLET_PATH equals AGLET_HOME/public, where AGLET_HOME is the directory where the Aglets Workbench was installed. With regard to our architecture it is important to note that the Aglets Daemon has to be running on every host on which process monitoring is supposed to take place.

3.3.2.3 Experimentation:

The following problems came up during our experimentation while putting this technology in a prototypical implementation.

Design Issues

As we tried to develop our application up to the point where it can retrieve actual "live data" about processes on remote machines, we encountered a set of design issues that revealed a problem complexity significantly higher than we expected. These are the major design issues that need to be addressed:

1. One application consists of several process classes and there are several instances of one process class at a point of time. It is not trivial to determine what process classes are to be monitored concerning one application. The name of a process does not necessarily reveal this association. If we had to monitor X-Windows processes running in a system, there would be a number of process instances of the class "xterm" – do we need accumulated information of all xterms in general or more detailed for every single instance?

2. One process instance appears at several "snapshot" times, but not necessarily all of them. Back to the X-Windows example, xterm processes appear and disappear frequently over time. How do we account for processes at times inside our monitoring period when they do not exist in the system?

3. Several instances of either the same or a different process class communicate with each other. According to how the number of entities increases that we are monitoring, it figures that monitoring their communication will be even more complex.

Implementation Issues

Regarding UNIX platforms as a restriction for the applicability of our tool, there are several points to consider when implementing the helper tool that retrieves the process data from a running system:

1. Process identification. In a UNIX system processes are identified by their process ID, whereas our tool looks for processes associated with a certain application by their name.

2. Process State. Whenever our tool takes a snapshot of the process table, it will never find any processes running, because only one process is running at any given time, and in this particular moment it will always be the tool itself!

4. Process communication. In our experiments we could not really figure out, how to measure communication between two processes. First of all, there are several ways how this happens, e. g. pipes, sockets, shared memory, etc. Regarding sockets for example, it would be necessary to know the ports that the processes in question are using for their communication. Then our tool would have to listen on all these ports.

CHAPTER 4

DESIGN AND ARCHITECTURE OF ENVIROSEARCH

In World Wide Web (WWW) of interconnected computers, a system is identified as a hostname and a document which is published on the Internet is addressed as Uniform resource Locator (URL). Usually URL contains a full hostname including the domain in which it exists, port number if a web server is listening on a different port than the standard one (80) and a document including full path. This document usually is written in HTML language which will be interpreted by browser which reads them in. The HTML document will have a head section and a body section. Body section is the one which contains an information which owner of that document wants to publish to Internet community. HTML document also have some protocol specific tags represented in form of a <TAG > string. The current URL of NJPIES - http://njpies.njit.edu provides users with the capability of looking for some historical data using different categories. User can search by facility/factory emitting pollutants or by chemicals which are emitted by different factories in NJ state. These data have not been updated for long time though. The tool which I am providing to NJPIES (which will be also referred as EnviroSearch) will allow user to search for environment related documents. That tool is using Java technology and can be accessed by user friendly GUI screens. This section describes about requirements, system design and related data models.

As objectives suggested in first chapter, we have a need of a search program which will download environment related documents, articles, research papers etc. on its own without having human interaction so that users of NJPIES can get up-to-date information. The current focus being on "New Environment Wave". Following issues led to this need.

(1) At present, there are lot of environment engg. Related web sites but most of these sites collect general purpose environment related documents. A knowledgeable user might not get latest and complex information from these sites.

(2) Even these general purpose information related to different environment issues are existing on different sites or links. Users need to browse through multiple websites in order to get all related information. These websites may not be connected through links within each other so users need to know full URL by heart. Only couple of sites in USA are there which really keep large collection of documents under same site.

(3) "New Environment Wave" deals with some of the advanced topics like pollution prevention, life cycle analysis of different pollutant chemicals, green accounting and social cost related to hazardous emissions. There isn't any known URL at present which has documents related to "New Environment Wave". Interesting articles or documents can be existing on remote URLs which need to be discovered. That requires some kind of intelligent program which will find these unknown sites on its own without needing interaction of human users.

(4) http://www.envirosources.com and http://www.epa.gov - these sites are well known to environmentalists but they most of the time deal with issues related to USA or the world as a whole and state level information is still missing from these sites. By expanding NJPIES program, New Jersey state will be benefited in some sense.

(5) Student studying in environment engineering faculty at NJIT, need to know about different URLs (WWW Addresses) where they can look for certain kind of documents of their interest. NJIT needs to provide them some kind of platform or entry point from where they can explore the Internet world further. NJPIES and this search tool will provide them that convenient facility.

(6) Existing search engines like Yahoo!, AltaVista, Lycos are basically general purpose search engines. They ask users to enter a keyword or a phrase (set of keywords) and then they list documents containing those keywords. Most of the time, they list lot of unrelated and confusing documents and sometimes users get frustrated by unorganized and irrelevant information. Some kind of special search engine is required which deals with only environment related issues. That can be extended further by incorporating environment engineering related artificial intelligence.

Issues described in earlier paragraph resulted in following requirements:

4.1 Requirements

(1) NJPIES Web Users should have facility to search for documents based on any keyword/phrase similar way as current standard search engines are doing. Since these search engines are already existing in industry, there should not be duplicate efforts at NJIT. Using those engines in integrated fashion should be a preferable approach.

(2) Since standard search engine will not be able to filter the documents in intelligent manner and since environment engineering students will be looking for most of the time environment related documents; this search engine should have some basic

intelligence which can help them to decide whether a document is related to environment related or not.

(3) Some kind of scoring points associated with a URL should help users to find relevance of the contents.

(4) Also standard search engines might not list all related documents at all time since articles and URLs keep getting changed over the period of time. That requires to have a some kind of local storage where we can store these documents on permanent basis. This feature will make sure that even if a document exists on a specific URL for just certain limited time (e.g. monthly magazine might keep a pollution related document for maximum a month or two), we should be able to have its copy on permanent basis in our database.

(5) Most of the search engines employ thousands of human operators who will manually search for documents then analyze their category and subsequently store these documents in different tables, files. Some kind of automated approach is desirable since university can not do these kind of job with limited resources.

4.2 System Design

The basic philosophy behind the design is resulted from Java's capabilities: to connect to a given URL and download the source document of that URL. In order to make search automated, agent program needs to visit thousands of sites and then evaluate those sites based on some pre acquired knowledge. If that site (URL) is worth our interest, then agent can download that source document from that URL to local system. That requires agent to have knowledge of thousands of different URLs to visit ('visit' verb here and now onwards will be used more in its logical meaning than in its dictionary meaning where we normally visualize entity to move physically from one place to another. Agent program actually downloads source document from remote system to local system in order to do further processing on that document). Initializing that agent program with those thousands URLs is next to impossible task and that's where a practical assumption helps the design. Most of the time; a document at particular URL will have links (reference to other documents on other URLs) as a part of its content. So agent program needs to be fed some 5-10 URLs in the beginning and program can then start accumulating more and more URLs for future visits. This way, for example; 5 different URLs can have total 20 referenced URLs, those 20 referenced URLs may subsequently have references of 80 URLs and so on. Figure 5 shows a sample HTML page (Source Document of a URL).

```
<HTML>
<HEAD>
<TITLE> This is a Title for the sample page </TITLE>
<! This is a comment tag in HTML document>
</HEAD>
<BODY bgcolor="#ffffff">
<H1> Sample Header </H1>
This is a body of this document. Main content/text of this page is found here. <br>
This line will be printed in next line
<hr>
We just had a horizontal line break <br>
<IMG SRC="./picture.gif" HEIGHT=30 WIDTH=30> This tag appends a picture from the file on this
webpage. <br>
<A HREF=" HTTP://www.referencedsite1.com "> Hit here to browse a new link </A> will put a link in
this homepage. User can click and retrieve some more information from www.referencedsite1.com site.
<br>
<A href="http://www.referencedsite2.com/doc2.html"> Click here to browse a new link </A> puts a link
where users can search further. <br>
Now there will be end of a text or body
</BODY>
</HTML>
```

Figure 5: A Sample HTML Document

Figure 6 shows the same page after removing all HTML tags from the page. This is what users see when they browse that URL. In other words, that is an actual content of that site. Figure 7 shows referenced URLs on this sample HTML page which would be collected by an agent program for further search.

Sample Header

This is a body of this document. Main content/text of this page is found here.
This line will be printed in next line

We just had a horizontal line break
 This tag appends a picture from the file on this webpage.
Hit here to browse a new link will put a link in this homepage. User can click and retrieve some more information from www.referencedsite1.com site.
Click here to browse a new link puts a link where users can search further.
Now there will be end of a text or body

Figure 6: Actual content of a HTML page after removing HTML tags.

www.referencedsite1.com
www.referencedsite2.com/doc2.html

Figure 7: Retrieved links (URLs) from the sample HTML page

Figure 8 shows the basic elements of an overall search tool system.

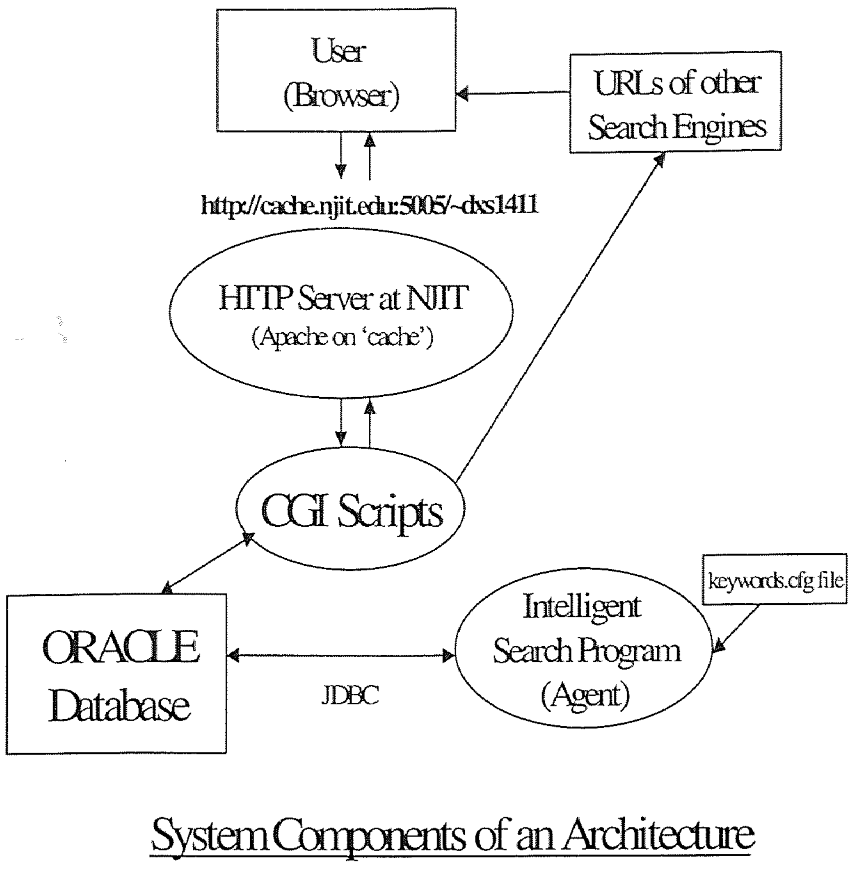


Figure 8

Architecture consists of following five entities:

(1) GUI Client & HTTP Server: It is an entry point or an access point for users within a homepage of the NJPIES (http://njpies.njit.edu). Web Server will be listening all incoming requests on one of the ports.

(2) Database: Database stores all interested documents and most of the time just URLs which contain them.

(3) CGI Programs. These scripts help Web Server on NJPIES to do specific functions. When users want to use Meta-Search Engine; specific Perl script will be invoked by a server. Similarly when users are interested in looking for documents from Oracle database, different Perl script will be invoked by the server. This decision will be made by a server based on user's selection on the NJPIES homepage. DBI (Data Base Interface) module is available for all kind of relational databases so one needs to write a CGI script which talks to DBI. Perl has facility to talk to any database through this DBI interface. That allows us to change back end database as and when required without changing our CGI programs. Figure 9 demonstrates overall architecture in this context.

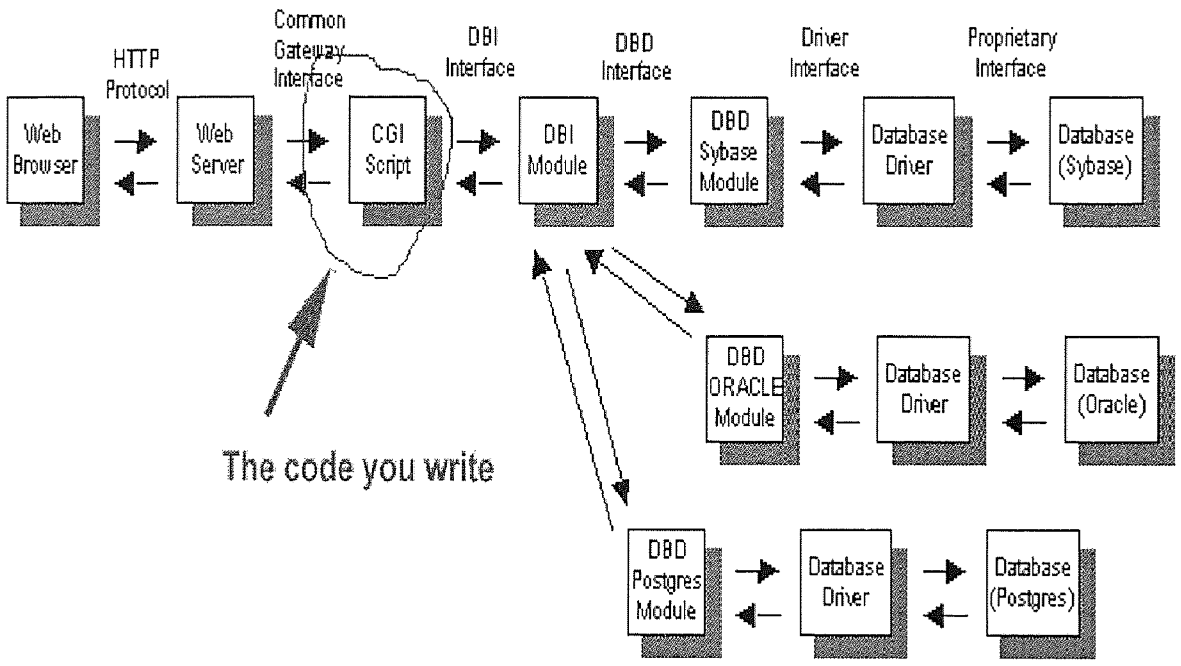


Figure 9: CGI-Database Connectivity [Selena98]

(4) the Internet: The Internet is basically source entity for most of our functions. Meta-Search Engine script as well as Java program (agent as described below) will be accessing the Internet all the time to look for environment related documents. All the systems which are part of this overall design need to be connected to the Internet through TCP/IP protocol.

(5) Intelligent Search Program (Agent) : This is a program running on 'cache' system. This program will be invoked by System Administrator and will then search thousands of different URLs to download interesting documents and listing of those documents on different URLs. This program uses keywords and its appropriate weight defined in a separate file. Based on the content of a document at a given URL, agent program tries to rank that document while comparing all existing strings in the document with the keywords listed in a file.

First four components of the system design are self explanatory and will be discussed further in Implementation chapter. Agent program requires some attention though.

Agent program needs to communicate with different URLs and hence can be implemented two ways:

(1) It can be a C/C++ program which will then open a socket connection with different URLs. After opening a socket connection, program needs to adhere to HTTP protocol to communicate with the server effectively. In most of the systems, HTTP Server runs on default port# 80 or alternatively on port# 8000 or 8080. We need to know wither IP address or domain name of the system to open a socket connection.

(2) Use Java language to implement 'agent' program so that connection to URLs can be easily done using Java's in built url.openStream() function where url represents a class of type URL.

It is also important that agent program doesn't waste its time visiting the same URLs again and again and collecting same list of referenced links (URLs) for yet another revisit. This can definitely happen in a practical world where a same article/document/URL is referenced in multiple sites. This is very important to avoid since in that case, agent program might hit a loop where for example URL A can have references for URL B and C. URL B might have references for URL D and E and URL E has reference for URL A or B. This situation will put 'agent' program in the loop and then agent will not be able to achieve its goal of collecting lot of different URLs.

To avoid the issue raised in the previous paragraph, we need to have a way of finding out whether a URL is visited before or not and that adds another complexity. Over the longer duration of agent's run chances are there that agent will visit thousands of different URLs and that will keep increasing over the period of time. If agent program keeps recording visited site in some kind of database, that database will be exhausted at some point of time. Similarly, time which program will take to compare a newly retrieved URL with those visited ones will also increase exponentially over the period of time. A middle approach balancing two problems and solutions should be taken.

Similarly, as we identified in requirements section, our goal is to store all URLs and their documents which are up-to-date. If a URL which used to point to an interested article before; doesn't contain that information later then that should be removed from our local database. Users should not be seeing outdated information at any time. That requires to have a separate agent program (agent# 2) which kicks off every month or so and revalidates all URLs stored in our database. Basically that program will visit only those URLs which are listed in EnviroDB database and will re-rank them. If needed, links will be then deleted from the database. This program which is supposed to be running periodically, can also solve our previous problem that is 'storing thousands visited sites' by deleting them periodically. This way, we are solving two problems.

(1) Our list of visited URLs don't increase exponentially

(2) In a period from the last run to this run, visited URLs can have changed and might be containing articles or documents of our interest now. By deleting that list periodically, we enable our 'agent' program to revisit them and re-rank them.

CHAPTER 5

IMPLEMENTATION

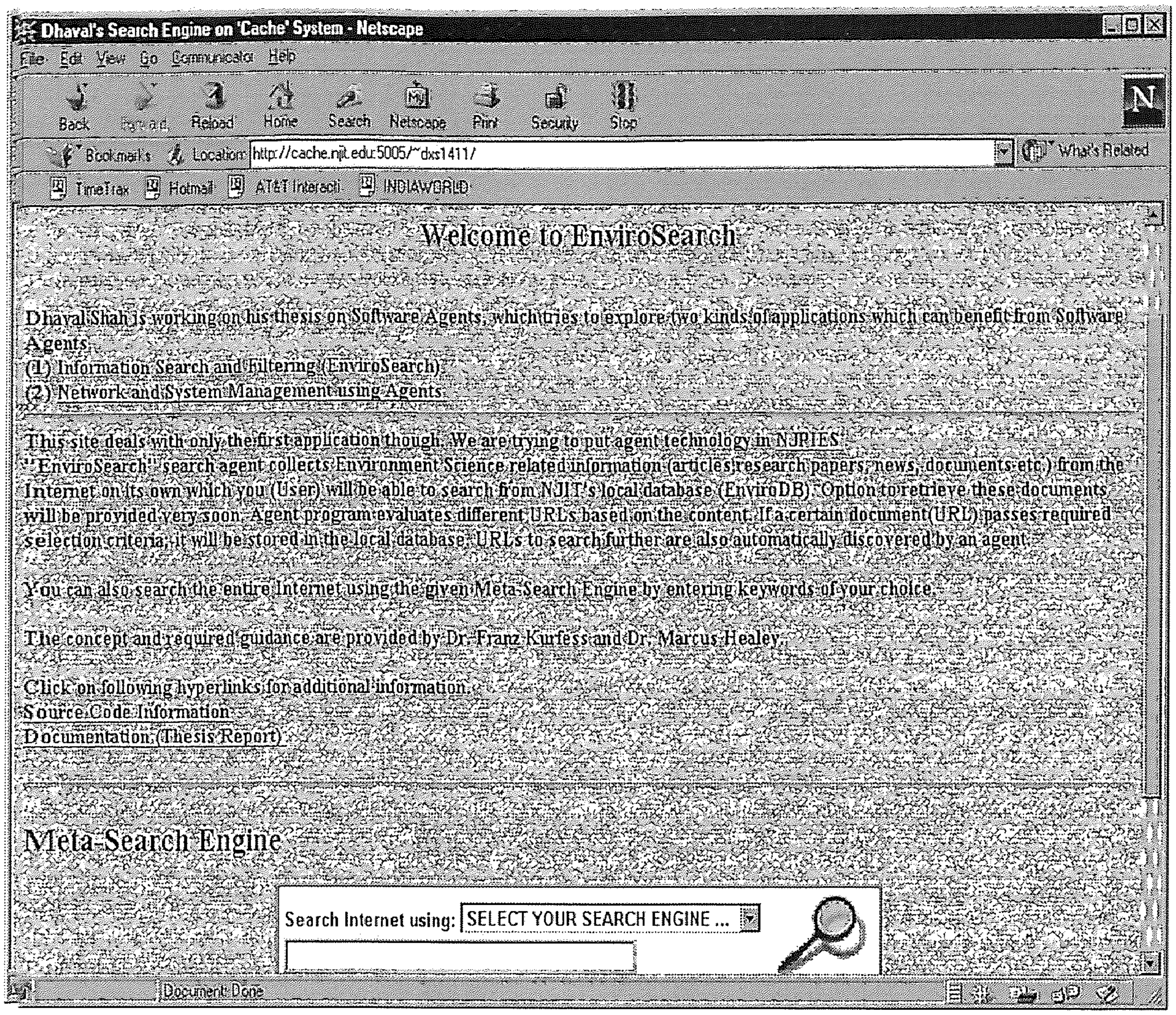
The five entities described in a system are described in detail in this section along with technology and tools used to implement each of them. Some design issues are also elaborated here. For the clarity purpose; implementation details are divided in two parts:

(1) Front End Tools

(2) Back End Tools

5.1 Front End Tools

Users access this search engine through Netscape or Internet Explorer browser. In the integration phase; a hyperlink will be established for this search engine on current homepage of NJPIES (http://njpies.njit.edu). At present, I have kept this tool separate from the NJPIES site. A dedicated Apache Web Server is running on 'cache' system (Solaris OS) at NJIT and it listens on port # 5005 for the requests from users. Users need to use URL http://cache.njit.edu:5005/~dxs1411 through their existing browsers to access this search engine. Once users launch this URL through the browser, homepage of this search engine will be displayed as shown in Figure 10. This screen will serve as an entry point for users. This homepage is designed using HTML language. Users are given a choice to select a method. Users can search for environment related documents form NJIT's local database or users can search for any documents from the Internet using keyword search like we normally do while using standard search engines.


Dhaval's Search Engine on 'Cache' System - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Stop
Bookmarks Location: http://cache.njit.edu:5005/~dxs1411/
What's Related
TimeTrax Hotmail AT&T Interacti INDIAWORLD
Document: Done

Welcome to EnviroSearch

Dhaval Shah is working on his thesis on Software Agents, which tries to explore two kinds of applications which can benefit from Software Agents.
(1) Information Search and Filtering (EnviroSearch)
(2) Network and System Management using Agents.

This site deals with only the first application though. We are trying to put agent technology in NJRIES.
"EnviroSearch" search agent collects Environment Science related information (articles,research papers, news, documents etc.) from the Internet on its own which you (User) will be able to search from NJIT's local database (EnviroDB). Option to retrieve these documents will be provided very soon. Agent program evaluates different URLs based on the content. If a certain document(URL) passes required selection criteria, it will be stored in the local database. URLs to search further are also automatically discovered by an agent.

You can also search the entire Internet using the given Meta-Search Engine by entering keywords of your choice.

The concept and required guidance are provided by Dr. Franz Kurfess and Dr. Marcus Healey.

Click on following hyperlinks for additional information.
Source Code Information
Documentation (Thesis Report)

Meta-Search Engine

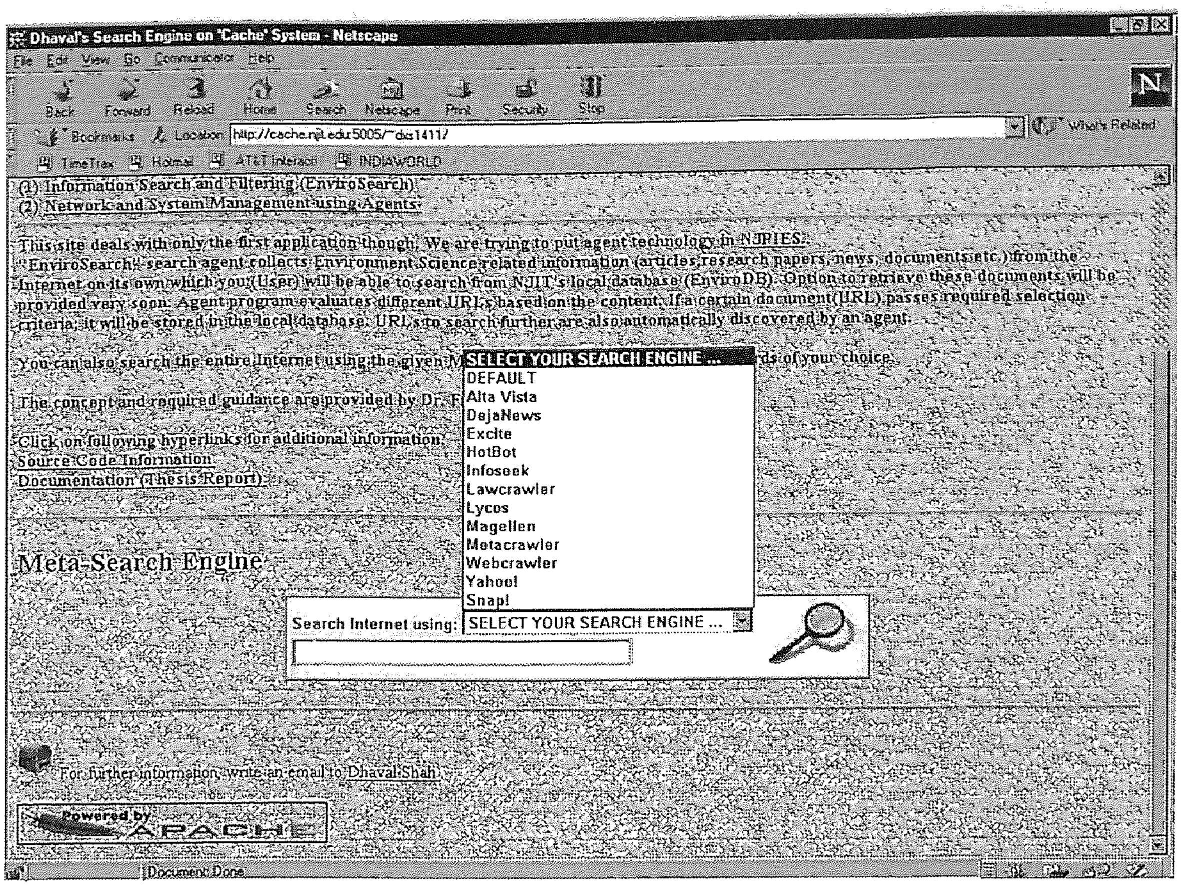
Search Internet using: SELECT YOUR SEARCH ENGINE ...

Figure 10: User Interface to EnviroSearch

User is given this choice in form of form processing. When user enters his/her choice in this form, that form along with data is given to a special program on server side. These programs are called CGI (Common Gateway Interface) programs. CGI is a standard way of processing user's data on server side. That way Web Server (http server) can concentrate on just communicating with users over TCP/IP network. CGI scripts do add some of the securities issues since that allows programs or processes to be executed depending on user's data which can be malfunctioning. Perl is very powerful language to do pattern searching and string manipulations. CGI scripts can be also written in C,C++, AppleScript, etc. languages but because of powerful features of Perl, I have used Perl scripts where necessary. HTTP server passes user's data to CGI program in a standard fashion. Depending on the method used while writing a form, different approach is taken. When 'GET' method is used; data is given to CGI scripts in a special environment string called "QUERY_STRING" and "QUERY_LENGTH" parameter defines the length of the input. When "POST" method is used, data is given to CGI scripts through standard I/O facilities. Different data fields of the form are connected with a special character '&' to form one string.

5.1.1 Meta-Search Engine

When user wishes to search the whole Internet using a keyword or a phrase, this option will be selected. One can also do that by using existing standard search engines like Yahoo!, AltaVista, InfoSeek, Lycos, HotBot, WebCrawler, etc. This meta-search engine provides a capability to use any of them using a single interface. These search engines have their own homepages and normally they throw a screen asking for a keyword or set of keywords in form of a phrase. Once user enters keywords, their database will be searched and all documents containing those keywords will be listed as a response. To achieve this, these standard search engines keep large index files and these index files are daily updated by their human operators or by their special programs. These search engines have well known URLs and also well known syntax of query string which is passed to their CGI programs. These strings are used by my CGI program (Meta-Search Engine) which is written in Perl. Once the user selects a Meta-Search Engine to do a further search, script displays the screen as shown in Figure 11. User will select his choice of search engine and will enter keyword(s) to search the Internet. Depending on what user selects for his preference of search engine, corresponding string is then used along with entered keyword(s) for further processing. Now another beauty of HTTP (HyperText Transmission Protocol) is used to actual search on the selected search engine. Instead of following steps like contacting that search engine, passing the keywords, retrieving resulted document/screen and then passing it to NJPIES user; my script just throws that well known string back to user along with 'LOCATION' tag which causes user's browser to access the URL passed by my meta-search engine script.


Dhaval's Search Engine on 'Cache' System - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Stop
Bookmarks Location http://cache.njit.edu:5005/~dxs1411/
What's Related
TimeTrax Hotmail AT&T Interact INDIAWORLD
SELECT YOUR SEARCH ENGINE ...
DEFAULT
Alta Vista
DejaNews
Excite
HotBot
Infoseek
Lawcrawler
Lycos
Magellen
Metacrawler
Webcrawler
Yahoo!
Snap!
Search Internet using:
Document: Done

(1) Information Search and Filtering (EnviroSearch)
(2) Network and System Management using Agents

This site deals with only the first application though. We are trying to put agent technology in NJPIES.
"EnviroSearch" search agent collects Environment Science related information (articles, research papers, news, documents etc.) from the Internet on its own which you (User) will be able to search from NJIT's local database (EnviroDB). Option to retrieve these documents will be provided very soon. Agent program evaluates different URLs based on the content. If a certain document (URL) passes required selection criteria, it will be stored in the local database. URLs to search further are also automatically discovered by an agent.

You can also search the entire Internet using the given M ... rds of your choice.

The concept and required guidance are provided by Dr. F

Click on following hyperlinks for additional information:
Source Code Information
Documentation (Thesis Report)

Meta-Search Engine

For further information, write an email to Dhaval Shah

Powered by APACHE

Figure 11: Meta-Search Engine Screen

5.1.2 Database Search Script

Once user selects option to retrieve existing documents related to environment from NJIT's local database, following screen will be thrown to a user by a CGI program which will be then connecting to database (Back End) and then retrieve documents. Using this screen, user will be able to specify keywords representing his interest. All documents having those keywords and which are stored in the local database (EnviroDB) will be then listed in a similar fashion that standard search engines use. The script which does this job will be also written in Perl language. Perl also provides database access support so that CGI program can contact any standard database software and can retrieve/update tables within the database by using SQL (Structured Query Language).

5.2 Back End Tools

The tools described so far deals with users directly and hence shares front end processing. The tools described in this section will be transparent to users of NJPIES. These tools/entities are not directly visible to users. In the previous section of front end tools, we said that all documents existing in local database (EnviroDB) will be listed by the script to users. We didn't specify though who will feed this database and what kind of database will be used. These questions will be answered in this section.

5.2.1 Database

NJPIES is using Oracle 8.0 database to store all important data. Data are stored in different table spaces and again within those table spaces in different tables. EnviroDB is NJIT's database which is used for NJPIES activities. SQL (Structured Query Language) is standard interface to relational database. One can also interact with Oracle database by implementing program logic in Pro*C. In EnviroSearch, we have used JDBC facility to talk to Oracle database. During the integration phase; documents which my search engine is going to store; will be a part of EnviroDB but at present for the security and integrity purpose, I have my own tablespace in Oracle database on 'cache' system. Following tables are stored under that database. Definition of each table is shown below.

Table : URLLIST

URLLINK	VARCHAR(250)

Actual Uniform Resource Locator which holds the document of our interest.

SCORE	NUMBER(4)

Points scored by this URL in terms of its content

CONTENT	VARCHAR(500)

Document which is hold by that URL. We store here first 500 characters of that document for listing purpose.

5.2.2 Agent Program

This will be a separate program with a goal of just collecting different URLs which might have information related to Environment topics. This program will be invoked by a authorized person and it will be running on 'cache' system. This program has a capability to run in infinite loop but in order to have our control on overall operation, a special terminating conditions are provided to an authorized person. These termination conditions allow program to stop at the end of specified period of execution or it can stop after visiting certain number of URLs. Initially this program takes a set of URLs to start searching with; but then this program keeps getting new URLs from the visited URLs. Logic here is that most of the URLs which will be initially given to an agent program, will also have some more URLs listed in form of referenced sites or hyperlinks. These can be easily found by viewing a source of the URL which agent is processing. Referenced URLs will have tag of <HREF="HTTP://referenced.url.site>. This way, implementation of this program will be equivalent to so called spider or warm programs which keep visiting different URLs in the Internet and then download the content and a link information of interest.

This whole activity is done as a series of steps defined below (Figure 12):

(1) Authorized person starts the program (let's give a name 'Agent' to this program). Agent is fed a set of URLs to visit and keywords of our interest. These keywords are in a separate file named "keywords.cfg" so that they can be reconfigured dynamically.

(2) Agent visits a URL and downloads the source document. (i.e. it connects to the remote URL and retrieves a source document)

(3) The downloaded document will have lot of unnecessary HTML tags. At the same time, these tags will have a list of other URLs which are referenced in that document and those URLs might have some more interesting articles related to "New Environmental Wave". Agent adds these list of URLs into its itinerary which is implemented by a well known feature in Java called Vectors. (URLs to be visited).

(4) Agent removes all HTML tags which are mainly there for formatting purpose. Interestingly this task saves our ranking algorithm from evaluating that site in a wrong way. At present, ranking procedure, checks only text content of a document. This is done after removing all HTML tags. In other words, we are evaluating the URL based on its real content. In real world, lot of URL owners advertise their sites in a false manner by putting lot of unnecessary keywords in meta tags and that too number of times. What do they achieve by doing this is that their sites are picked up and ranked higher by number of search engines who just check the whole structure of a URL source document for the keywords. These false keywords are not seen by

viewers when they view these pages in their browsers. These meta tags help these sites to be picked up easily even though actual content doesn't contain interested information for a viewer.

(5) Now 'agent' has an actual text document. Still agent doesn't know whether that document is related to environment or not.

(6) Now agent's intelligence plays a roll. It checks the content of this document; keyword by keyword and will try to match them against the set of keywords already supplied to it in a file. This file has list of keywords along with weight assigned to that keyword. Based on the occurrences of these keywords and their weigh, agent assigns a score to that document (or a link) . Details of this algorithm is described in a separate subsection.

(7) Let's assume that at NJPIES; we are interested in collecting a document with the link information if a document scores above 50 points. Agent has just ranked the site based on its content and it scores 78 points. Agent will copy the link information and a content (or part of it) in our database. Agent will also assign some keywords to that document within the database so that the same document can be extracted even by some different keywords when user wants to search a local database. If that URL scores less than our passing score of 50, agent interprets that link being useless link and discards that document.

(8) Agent already has an itinerary (URLs to visit) and has recently put some more URLs from the previous document, agent picks up the next URL from that list and repeats all steps from 1 - 7. Agent also increments its counter of number of URLs visited so far.

(9) If a counter reaches a limit specified at the time of the invocation of this agent program, agent stops doing further search. At this time system has a new static file /database table available which contains URLs already visited so far. Agent doesn't repeat that when invoked next time.

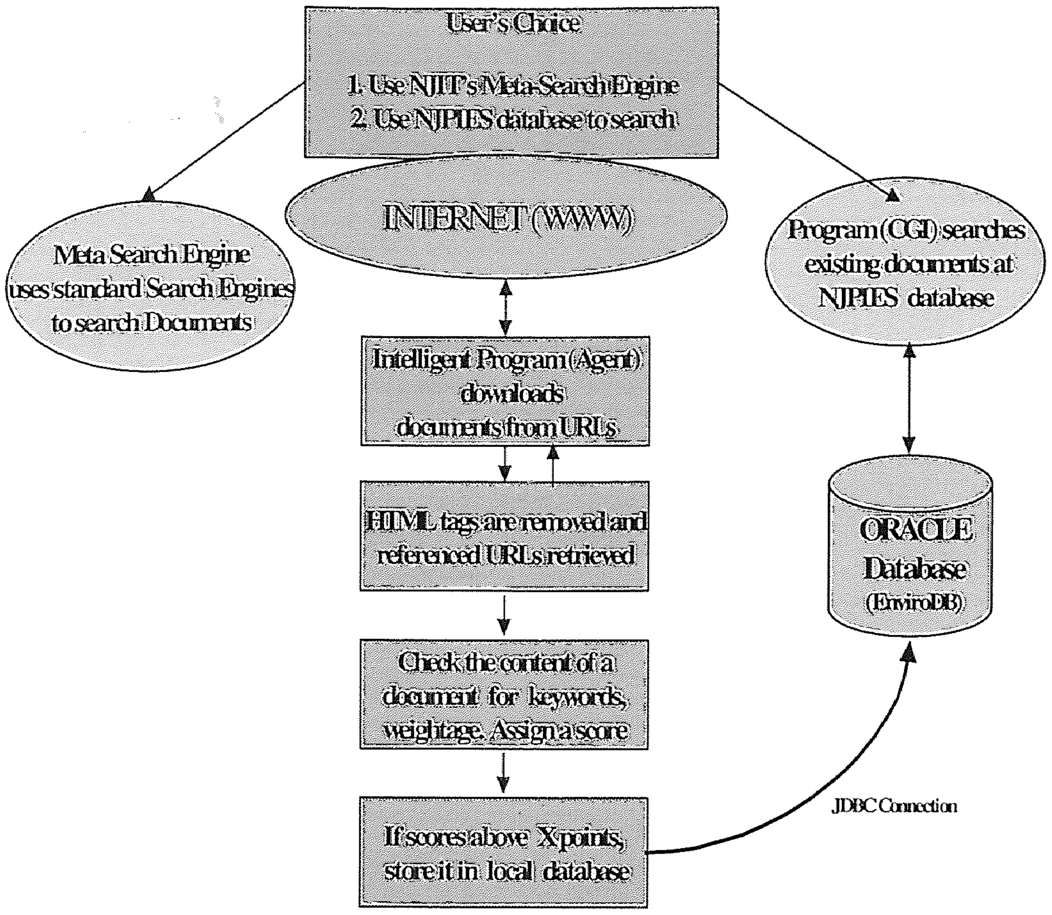

User's Choice
1. Use NJIT's Meta-Search Engine
2. Use NJPIES database to search
INTERNET (WWW)
Meta Search Engine uses standard Search Engines to search Documents
Program (CGI) searches existing documents at NJPIES database
Intelligent Program (Agent) downloads documents from URLs
HTML tags are removed and referenced URLs retrieved
Check the content of a document for keywords, weightage. Assign a score
If scores above X points, store it in local database
ORACLE Database (EnviroDB)
JDBC Connection

Figure 12: Steps followed by an Agent

Configurable Parameters:

Following parameters need to be configured. These parameters can be configured using a standard configuration file or through command line arguments.

(1) PASSING_SCORE: If a document scores above this score, agent treats this document or URL which contains this document as an important site and will download that document or name of a link (URL) in local database.

(2) URL_FILE: It is a name of a file which contains initial set of URLs which agent should be using for the search. Once agent visits these URLs, it already has new set of URLs for further search. This is set to "unvisited.lst" at present.

(3) URLS_TO_VISIT : This number represents the terminating condition of a program. As soon as agent visits these many URLs, it stops its further execution and reports on its progress.

(4) ORACLE_USER_LOGIN/ORACLE_USER_PASSWORD: This login and password identifies a user in ORACLE database. Table which is storing URLs and documents belongs to this user.

(5) TEMP_FILE_NAME: Temporary Filename which is used by an agent program to store source document of a URL. This is set to "output.txt" at present.

(6) SA_PASSWORD: It is not implemented yet, but this password will hold the password for a system administrator who can execute this program.

(7) KEYWORDS_FILE: This is a filename of a configuration file which holds all our interesting keywords along with appropriate weight. Keyword and weight is separated by a special character '*'. This parameter is set to "keywords.cfg" at present.

(8) URLS_TO_STORE: This is alternative terminating condition representing number of URLs to store before agent stops the execution.

(9) TERMINATION_METHOD: This variable will represent type of method used for terminating a program.

5.2.2.1 Scoring Algorithm:

(1) Store all keywords and weight in one ASCII file which will be read by a search engine. Weight will be in range of 1 to 10 for each keyword/phrase, higher the weight, more valuable is that keyword. For example if we want to download all documents related to "Pollution Prevention", "Pollution Prevention" keywords will get a weight of 10 points whereas a word 'Environment' will get a weight of just 1 point since it is more common. On the same line of thinking, simple "Pollution" word will get a weight of let's say 4 points.

(2) If a document contains three occurrences of "Pollution prevention" keywords (3 * 10 = 30 points), 8 times word "Pollution" in other context than prevention (4 * 8 = 32 points) and contains 12 occurrences of a word "Environment" (1 * 12 = 12 points), agent will assign a following score to that document or URL.

```
  30 ( for "Pollution prevention" 3 times)
+ 32 (for "Pollution", 8 times)
+ 12 (for "Environment", 12 times)
----------------
  74 (for content)
+ 23 (for number of occurrences of all our keywords 3 + 8 + 12)
```

Final:97 points

(3) If we have decided to store a document which scores above let's say 50 points, this document is very much valuable to us and we will copy it to our database.

5.2.3 JDBC™ (Java™ DataBase Connectivity)

JDBC™ is a Java™ API for executing SQL statements. (As a point of interest, JDBC is a trademarked name and is not an acronym; nevertheless, JDBC is often thought of as standing for "Java Database Connectivity".) It consists of a set of classes and interfaces written in the Java programming language. JDBC provides a standard API for tool/database developers and makes it possible to write database applications using a pure Java API [SUN_JDBC].

Using JDBC, it is easy to send SQL statements to virtually any relational database. In other words, with the JDBC API, it isn't necessary to write one program to access a Sybase database, another program to access an Oracle database, another program to access an Informix database, and so on. One can write a single program using the JDBC API, and the program will be able to send SQL statements to the appropriate database. And, with an application written in the Java programming language, one also doesn't have to worry about writing different applications to run on different platforms. The combination of Java and JDBC lets a programmer write it once and run it anywhere.

In short, JDBC makes it possible to do three things:

- establish a connection with a database

- send SQL statements
- process the results.

The following code fragment gives a basic example of these three steps:

```
Connection con = DriverManager.getConnection ("jdbc:odbc:dbname", "login",
"password");
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery("SELECT a, b, c FROM Table1");
while (rs.next()) {
    int x = rs.getInt("a");
    String s = rs.getString("b");
    float f = rs.getFloat("c");
}
```

JDBC is a "low-level" interface, which means that it is used to invoke (or "call") SQL commands directly. It works very well in this capacity and is easier to use than other database connectivity APIs, but it was designed also to be a base upon which to build higher-level interfaces and tools. A higher-level interface is "user-friendly," using a more understandable or more convenient API that is translated behind the scenes into a low-level interface such as JDBC. As interest in JDBC has grown, more developers have been working on JDBC-based tools to make building programs easier, as well. Programmers have also been writing applications that make accessing a database easier for the end user.

The JDBC API supports both two-tier and three-tier models for database access. In the two-tier model (Figure 13), a Java applet or application talks directly to the database. This requires a JDBC driver that can communicate with the particular database management system being accessed. A user's SQL statements are delivered to the database, and the results of those statements are sent back to the user. The database may be located on another machine to which the user is connected via a network. This is referred to as a client/server configuration, with the user's machine as the client, and the machine housing the database as the server. The network can be an intranet, which, for example, connects employees within a corporation, or it can be the Internet.

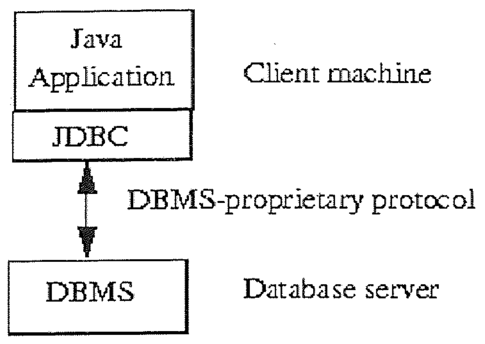


Figure 13: JDBC in two-tier model [SUN_JDBC]

In the three-tier model (Figure 14), commands are sent to a "middle tier" of services, which then send SQL statements to the database. The database processes the SQL statements and sends the results back to the middle tier, which then sends them to the user. MIS directors find the three-tier model very attractive because the middle tier makes it possible to maintain control over access and the kinds of updates that can be made to corporate data. Another advantage is that when there is a middle tier, the user can employ an easy-to-use higher-level API which is translated by the middle tier into the appropriate low-level calls. Finally, in many cases the three-tier architecture can provide performance advantages.

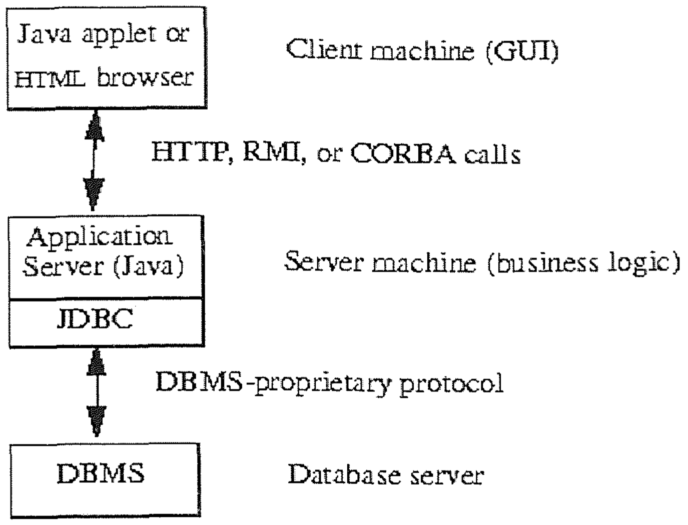

Java applet or HTML browser
Client machine (GUI)
HTTP, RMI, or CORBA calls
Application Server (Java)
JDBC
Server machine (business logic)
DBMS-proprietary protocol
DBMS
Database server

Figure 14: JDBC in three-tier model [SUN_JDBC]

CHAPTER 6

EVALUATION

EnviroSearch product combines Agent's capabilities of searching and filtering the content over the Internet. NJPIES is interested in collecting URLs which are related to only Environment and nothing else. The major objective is achieved by the implemented "EnviroSearch" product. However some interesting issues are involved with the implementation of this application and also some of the enhancement or modification possibilities are also there. Evaluating the application against its objectives; following observations can be made:

(1) Meta-Search Engine is a complete product in its own and can be useful to users of the NJPIES homepage. They will be able to use it to get documents/URLs containing keywords of their choice. This doesn't limit users to enter just environment related keywords. Users can enter any keyword to search URLs.

(2) Agent program which feeds URLs, environment related documents, articles, papers, etc. to ORACLE database needs to be enhanced. It uses first order ranking algorithm which is not that sophisticated. The problem being, this algorithm checks just the source document of the URL for keywords and not the subsequent links on that URL. Hence, ranking of that document doesn't depend on how many interesting links exist on that URL. Because of this limitations even if a URL is a bank of lot of useful environment related documents which are existing as separate links on that site, that site is not considered important by this agent program because it doesn't contain keywords which agent is looking for; as a part of its source document..

(3) At present, agent program is started manually by a system administrator who needs to log in on 'cache' system and provide some of the required parameter values. That whole procedure could be automated or at least capability should be there so that administrator can start that program from the same website where users normally access for using this search engine. In order to have some control over this operation/activity, password protection is must.

(4) Agent program evaluates URL's source document and hence it checks HTML document which is existing at that URL. Also reference links will be checked in subsequent run. If a source document has a link to WORD document or PDF Document then agent is not able to check the content of that document. For example http://www.epa.gov site contains lot of environment related documents but most of them are in form of PDF attachments which usually require reader to have Adobe's Acrobat Reader program (plug-ins). Writing these kind of plug-ins is anyway out of the scope of this thesis.

(5) The main functionality or the purpose of the implementation of this search engine in NJPIES is really achieved. "EnviroSearch" can be really helpful in providing the list of URLs and documents to "EnviroDaemon" search engine which is hierarchy based search engine. EnviroDaemon assumes the availability of all documents which are then searched for particular keywords at a specific places within the documents. User can specify looking for documents which contain a given keyword in main header or sub header level. Combining EnviroDaemon with EnviroSearch is a powerful idea and they should definitely complement each other to give a useful end product to end user.

CHAPTER 7

CONCLUSION

Development of agent programs are now not in just research areas. There are lot of commercial applications available in the market which are powered by agents. Comparing the required properties of an agent system; it can be realized that 'autonomy' and 'mobility' are desired properties of an agent program which makes them work on their own to achieve user's goals. With Java's popularity; it is easy to deploy mobile agents easily. Lot of toolkits are available which make that task really easy. Aglet Software Development Kit is one of them. With the wide spread popularity of the Internet it can be assumed that future trends will be more in developing web based agents. E-Commerce is one of the hot areas nowadays and lot of companies are trying to build their business on Agent's capability to search a specific information from the Internet. Personal Assistants or Virtual Assistants are new names for agent programs. In terms of summarizing two projects on which I got a chance to work, both projects tried to use different properties of agents. The purpose of the project; which Klaus and Felip did and I joined later was the development of a platform-independent tool to monitor the distribution of processes of an application on different processors in a distributed environment. Mobile agent technology seems to serve pretty well for the purpose of distributed process monitoring, due to the parallel operation and mobility of agents. Most of the problems we had with this work are caused by the project setting, i. e. limitations on time resources, and by technological complications, i. e. lack of a proper development environment and the need to use an agent framework that was not mature at the time of implementation.

EnviroSearch demonstrates agent's searching and filtering capabilities. Of course, it limited itself to just a text based document search. In real world, there are different type of objects ranging from images to multimedia clips.   A software discovering new URLs on its own and that too after evaluating contents of that URL based on user's search criteria is not a small wonder. Technology keeps changing and more and more useful tools will be coming out very soon and there will be always a need of enhancements and modifications but because of modular design of my agent program, I am sure that the program will be able to go through all changes with lot of ease and also without sacrificing any major functionality.

CHAPTER 8

FUTURE WORK

For the benefit of next batch of students who would like to research further in this area, I have identified some of the enhancement or modification possibilities. Because of emphasizing on research more than the implementation, I did not get enough time to implement some of the small utilities. These missing items are also listed in following list.

(1) Agent program uses very simple or first order scheme to rank a source document of a given URL. Some more trial and error practice is required to check the effectiveness of that scheme. One can put more intelligent scheme or algorithm and it can be easily integrated in overall architecture by replacing one of the subroutine processing.

(2) I could not implement CGI Script which will retrieve Environment related documents form the Oracle database. A major step of feeding that database with documents is already implemented though using JDBC within an agent program. Somebody with expertise in Perl can easily do that in one week time. Alternatively, similar JDBC code can be used in a program to retrieve those documents. In that case, code will be acting slightly in reverse directions.

(3) Associative Memory and Wordnet are two projects undergoing at NJIT, which deal with similarity based search. EnviroSearch search engine can be altered to have direct interface with these projects so that agent can search for documents with similar keywords to ones which defined earlier. In this case, agent will use standard words from its configured file and then will use these products to identify similar possible

words which could be existing in the document under search. Figure 15 shows possible lay-out after that integration.

(4) At present, agent program is invoked at command prompt on UNIX machine. We can put it under the same URL(homepage) from where administrator can execute this program. Of course, administrator needs to identify himself by providing a password.

(5) George Chang, a Ph. D. student has developed a hierarchy based search engine (EnviroDaemon) which lists documents after comparing keywords at different hierarchy of a document. This is very important work to make NJPIES a popular destination for end users. Capability of retrieving more URLs from EnviroSearch and relevance searching capability of EnviroDaemon can complement each other.

(6) This agent can be enhanced to allow it to use standard search engines on its own so that it can find out more URLs using these standard search engines. Meta Search Engine can help there and ready-made query strings can be considered a URL for that purpose.

(7) Some more testing and enhancement are always welcome. Similarly Java program is foolproof only when you have caught all possible exceptions which might occur in this situations, I might not have done that at some places or may be I might have caught general purpose exceptions where I could have caught specific ones. This work will be helpful to one who needs to understand the architecture of my agent.

(8) As discussed in design and architecture section, we still need a separate program (agent #2) which will be executed periodically to check integrity and validity of all URLs stored under local database (EnviroDB). This program will also delete existing list of visited URLs which could grow in thousands. This will solve lot of

problems related to memory requirements, processing time and effectiveness of overall operation.

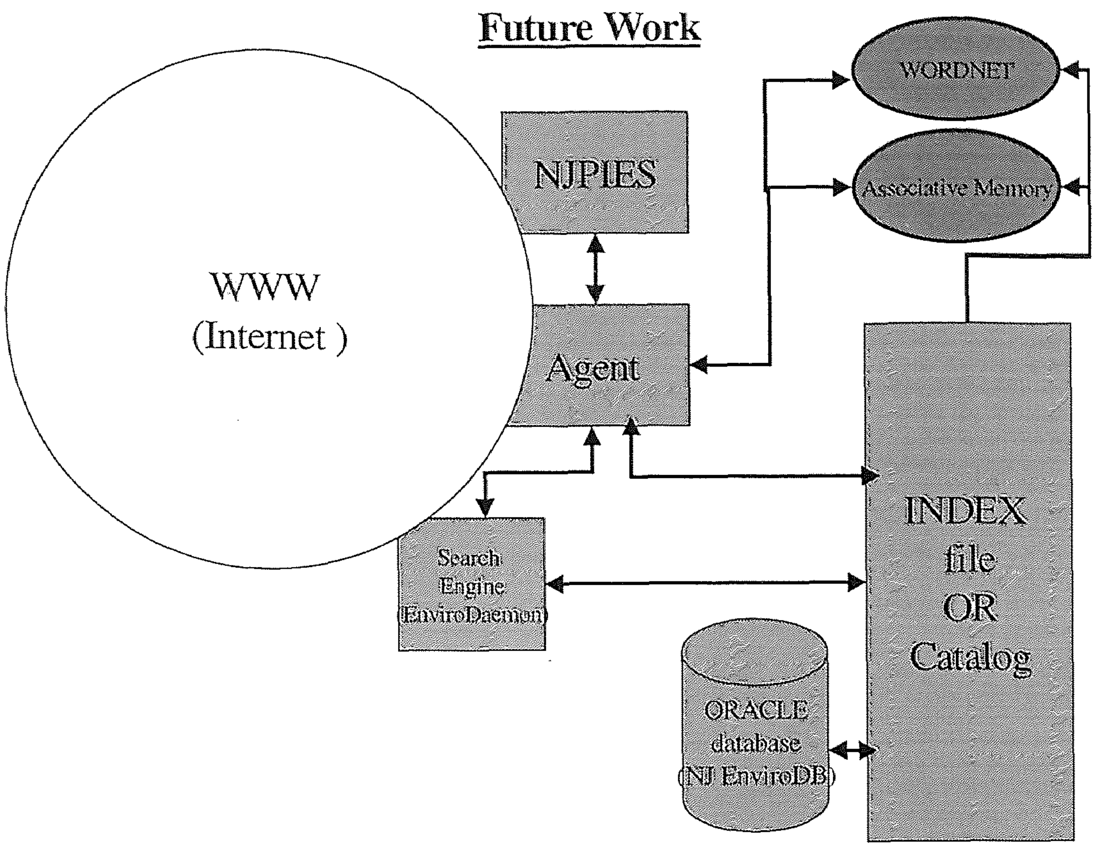


Figure 15: Possible lay out after integrating EnviroSearch with EnviroDaemon, WORDNET and Associative Memory projects

REFERENCES

- [Sloman94] Morris Sloman ed., *Network and Distributed Systems Management*, Addison-Wesley, Reading, MA, 1994.

- [IBM_AGLET] IBM's Web site for Aglets, *Aglets Software Development Kit*, http://www.trl.ibm.co.jp/aglets/

- [Lange98] D. B. Lange, M. Oshima, *Programming and Deploying Java Mobile Agents with Aglets*, Addison-Wesley Pub., MA, 1998.

- [Cockayne98] W. R. Cockayne, M. Zydd, *Mobile Agents*, Manning Publications, CT, 1998.

- [NJPIES] The home page for NJPIES, NJIT, NJ.
  http://njpies.njit.edu

- [Klaus97] Klaus Holthaus and Felip Miralles, *Monitoring Distributed Applications with Intelligent Agents*, Project Report, CIS Dept., NJIT, NJ, Fall 1997.

- [Stan96] Stan Franklin and Art Graesser, *Is it an Agent, or just a Program?,* A Taxonomy for Autonomous Agents. Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages, Springer-Verlag,1996.

- [IBM95] Gilbert, Aparicio, et al. *The Role of Intelligent Agents in the Information Infrastructure,* IBM, United States, 1995.

- [UMBC_KQML] *Information on KQML,* University of Maryland Baltimore County, MD.
  http://www.cs.umbc.edu/kqml/

- [ALPHA_JKQML] *Information on JKQML,* IBM's alphaWorks Mission, NY.
  http://www.alphaworks.ibm.com/formula/jkqml

- [Shoham92] Shoham, Yoav. 1992. *Agent Oriented Programming: An Overview and Summary of Recent Research,* In Proceedings of the Workshop on Distributed Artificial Intelligence, 1992.

- [Selena98] Selena Sol, *Using Perl 5 and the DBI Module to Communicate With Databases,* October 19, 1998.
  http://wdvl.com/Authoring/DB/Intro/dbi_intro.html

- [SUN_JDBC] *JDBC^{TM} Guide: Getting Started*, Sun Microsystems, CA. http://java.sun.com/products/jdk/1.1/docs/guide/jdbc/getstart/introTOC.doc.html