Comparison Shopping System
Using
Multiple Agents and auto-extraction

By
Baba U. Anem
CONTENTS

1. Introduction ................................................................. 3

2. Automatic Information Extraction ................................. 6
   2.1 Details of Auto Extraction System .............................. 8
   2.2 Learning and Extraction Process ............................... 9
       2.2.1 Providing training samples .............................. 10
       2.2.2 Generation of extraction rules ......................... 11
   2.3 External Interface .................................................. 12

3. Cooperative Agents ..................................................... 13
   3.1 Agent ........................................................................ 14
   3.2 Intelligent Agent ..................................................... 15
   3.3 Multi-agent System .................................................. 16
   3.4 Agent Communication .............................................. 17

4.0 Supporting Technologies .............................................. 19
   4.1 Communication Protocols ......................................... 19
   4.2 Parallel Computing .................................................. 20
       4.2.1 Multiprogramming .......................................... 21
       4.2.2 Multithreading ............................................... 22
       4.2.3 Distributed Computing Environment ................. 22

5. Implementation .......................................................... 24
   5.1 Experimental Results ............................................. 32

6. User Manual .............................................................. 35

7. Conclusion ............................................................... 39
1. **Introduction**

The main idea of shopping in World Wide Web is to reduce the travel time going to shops, and to get products at reasonably good prices. As many users prefer to do shopping on the Web, the WWW is becoming an important channel for retail e-commerce. Today we can buy virtually anything on the Web. With so much choice available, the customer these days finds himself rather overwhelmed and can spend significant amount of time finding a good deal. Although there is increasingly more information available via the Internet to make educated buying decisions, there are still computational limitations on gathering, filtering, and analyzing such data. Shopping activities require a large effort from a user and include searching for parties interested in selling or buying what the user wants to buy or sell, comparing prices and other features of the good or service to help make an optimal purchase decision. To help the customer in narrowing down his choices there are a number of sites available, which fall in the category of shopping agents. They are programs, which traverse the Web to find the best deals being offered by different Web sites. A comparative shopping agent is needed to automate several of the most time consuming stages of the buying process.

*According to Intelligent Agents Group (IAG)[1], an agent is a computational entity which*

- acts on behalf of other entities in an autonomous fashion
• performs its actions with some level of proactivity and/or reactivity

• exhibits some level of the key attributes of learning, co-operation and mobility.

Software agents (often simply termed agents) are software systems that loosely conform to the above definition and can be described as inhabiting computers and networks, assisting users with computer-based tasks.

We need software agents because
• more and more everyday tasks are computer-based,
• the world is in a midst of an information revolution, resulting in vast amounts of dynamic and unstructured information,
• increasingly more users are untrained,
• and therefore users require agents to assist them in order to understand the technically complex world we are in the process of creating.

Problem specification. The goal is to develop a system that should be able to extract product information from various Web pages automatically using multiple agents working parallel. The system is aimed at helping shoppers perform comparison of product prices from various competing online retailers. This system will crawl to different online vendor sites on behalf of the user and fetch price information for different products. Once the system gets the prices of a product from
different sites then user can compare the prices and buy a product. This system substantially reduces Web shopping time.

In this project we have integrated multiple agents and automatic extraction of data features to an existing comparison shopping system. The idea of implementing automatic extraction is basically to reduce the problem of changing the code for extraction of the required data from Web pages whenever there is any change in the target Web pages. We have. We have introduced a new feature called Auto-Fetch, which would fetch the book prices automatically when the system-load is low. This comparison shopping system will search different online vendor sites, help users to decide what product to buy, which store offers the best price for a given product and to substantially reduce Web shopping time.

In this report we discuss about the features Information Extraction System, Multiple Agent System, Auto Fetch, which we integrated to comparison shopping system. We will give an overview about the technologies used in this project. We will give the user manual, which gives information about how to use the system. Finally we will wrap up this with conclusions and future work.
2. **Automatic Information Extraction**

We have integrated the comparison shopping system with automatic information extraction system. This system extracts the necessary information automatically without using the pattern matching technique which previous comparison shopping system was using. One can ask the question why should we use Automatic Extraction system.

Due to the dynamic nature of the Web, the layout of the information on the page can change very often. This dynamic nature poses huge problems for such agents. If the agent relies upon the programmer to detect the changes in the layout and to change the information extraction algorithm accordingly, the agent’s efficiency and accuracy is compromised. Additionally, the process of manually changing code can become cumbersome, if the number of sites that the agent involves in its comparison-shopping is large. Another problem, with such an agent is, it is domain (product category) dependent. An agent built with hard-coded logic, for extracting information, specific to each Web site, would work only for that domain. To make it work for another domain, would mean writing code specific to the Web sites of that domain. Even adding a new Web site for the same domain would mean adding more code.

This Auto Extraction system automates the process of information retrieval and make the agent domain independent, the process of comparison-shopping can be made fast, more efficient and reliable. Auto Extraction system is a GUI based system, which enables the agent to
extract product information from the Web page. The algorithms use machine learning concepts. The idea behind using learning algorithms is that it will help make the agent generic and will make it possible to easily adapt to various domains.

The pages on the Web are dynamic. Everyday, several new pages are added on the Web and some existing ones discontinued, resulting in stale links. Due to this very flux of the Web, a comparison-shopping agent faces the need of quick adaptability and scalability. Its problem is further compounded due to the fact that the format of an existing product page could change anytime and very often. All the above factors warrant that many of the processes of the agent be automated. One of the processes that will improve the performance significantly, is the process of extracting product information from the Web page.

The previous system used pattern matching techniques to extract the necessary information from Web pages. Whenever there is any change in the Web site then the system would fail. We need to make a lot of changes to the code in order to handle the changes. It is really a very tedious job. To handle these kinds of problems we integrate the comparison shopping system with an automatic extraction system. The automatic extraction system uses machine learning approach, for automation of the extraction process. A typical machine learning process involves, developing algorithms that are first trained on some training set. From the training set the algorithms develop rules, which
can then be applied to the target set. The system incorporates multiple agents, which would go to different web sites and bring the corresponding web pages related to a particular product. Once the agents bring the web page auto-extraction module is applied to extract the information of interest.

2.1 Details of Auto Extraction System

This chapter describes an approach developed to extract information from Web pages, primarily suitable for comparison-shopping. This system attempted to automate the task of extraction as much as possible. The following steps [2] are involved in extracting relevant information from Web pages.

- **Structure Definition**: The first step involves defining a structure suitable for the relevant information on the Web page.

- **Providing training samples**: The second step in the process is to provide the learning engine with the samples that fit the structure defined in step one.

- **Generation of extraction rules**: The learning engine would generate extraction rules from the training samples provided.

- **Applying extraction rules**: The extraction rules generated in the previous step will be applied to the Web page to obtain all the relevant information. The effectiveness of the rules will be determined based upon the extraction results. These results will also
determine if more training samples are necessary for the learning engine.

- **Rule refinement**: The rules learnt can be fine-tuned manually if the learning engine is unable to capture all the details. We have to use the GUI based tools that enables us to carry out these steps. As shown in the figure, the process begins with a definition of the record structure contained within the Web page. The definition is stored in the database. The learner will try to learn to extract records of the specified structure from target Web pages.

![Figure 2.1. Learning and Extraction Process](image)

2.2 Learning and Extraction Process

Figure 2.1 [2] shows the overall process that we use, to learn rules and extract data from a Web page. The Learner and Extractor together
make up the GUI tool. Both these programs interface with a common database. The Learner has modules for Structure definition, providing training samples and Extraction rule generation. The Extractor handles Application of extraction rules and Rule refinement.

First we need to train the learner to identify records. Inputs to the learner are Web pages for whom the record structure has been defined. The learner is shown sample records on the Web page. It tries to infer rules called extraction rules from the sample records. This learning approach makes use of the inherent structure of tags and syntactic properties of plain text, to infer rules.

This system converts the entire page into a document tree. The tree is made up of tags and plain text nodes. The plain texts of the Web page end up as leaf nodes of the tree. The learner tries to identify a node of interest by exploiting some properties of this tree and the plain text nodes. The rules learnt by the learner for a particular type of a page are stored in the database under that page type. The extractor uses these rules to extract records from target Web pages. We have used extractor to check if the extractor is able to extract records properly.

2.2.1 Providing training samples.

The training samples that are provided to the learner are the records contained in the Web pages. We need to take special care while selecting the training samples. We have to consider the variations in length of Title, Author and Price. We need to consider variations in
depth of record fields. The Learner and Extractor modules behave differently on Linux machine when compared to Windows systems. I thought of training the different sites on Windows 95 and then unload the tables using mysqldump to load it in MYSQL database on Linux machine. It did not work as the system works differently on Linux. The ability of the learner to learn every nuance of the record structure that the Web site is capable of producing depends on thoughtful and careful selection of sample pages.

2.2.2 Generation of extraction rules.

The record samples stored in the database are used to generate the extraction rules. The rules that are learnt are also kept in the database. The extraction rules are learnt for every element of the record structure. Several key properties of the document tree are utilized to formulate key rules. The system gave good results for repetitive pattern and Uniqueness of nodes [2]. While creating the rules the Extractor module considers the following factors

- Depth of the Node
- Tag Sequence of each field
- Relative position
- Keywords
- Omitwords
- Value of the entire text
Every time a new record is shown it uses information from all the previous records and the new one to re-generate the rules. The record extraction algorithm has a time complexity of $O(n \log n)$, where $n$ is the number of nodes in the document tree.

2.3 External Interface

We have used an external interface module (exInterface.pl) to extract records from Web pages. We have also made use of templates in this module. Templates help to convert records from similar Web sources but different record structure, into those with same record structure. They make it very easy to do comparison-shopping. We have used Title, Author and Price in record structure. A record means a group of information relevant to some entity. In our case the record structure (Title, Author and Price) is for a Book. For creating templates Extractor tool has been used. Extractor module provides an easy way to define templates and associate similar record structure, to one template. The records that are extracted by the externalInterface module, are converted to some standard template, and finally stored in the database. These stored records are queried using standard SQL, and are shown to user to perform comparison-shopping.
3. **Cooperative Agents**

We have implemented cooperative agents in our comparison shopping system. The cooperative system was developed to solve complicated problems based on distributed problem solving. The main motivation is that using distributed resources concurrently can allow a speed-up of problem solving. In fact the possible improvement, due to parallelism, depends on the degree of parallelism inherent in problem. One problem that permits a large amount of parallelism during planning is a classic toy problem from the AI literature: the Tower of Hanoi (ToH) problem.

There are several distributed problem-solving strategies. The Cooperative agent system used the concept of “task sharing”[6] or “task passing”. When one agent has too many tasks to do it should enlist the help of agents with few or no tasks. The main steps in “task sharing” are:

1. **Task decomposition**: Generate a set of sub-tasks to potentially pass to others. This could generally involve decomposing a large task into sub-tasks that could be solved by different agents.

2. **Task allocation**: Assign sub-tasks to appropriate agents.

3. **Task accomplishments**: Each of the appropriate agents accomplishes their sub-tasks, which could include further decomposition and sub-task assignment, recursively to the point that an agent can accomplish the task alone.
4. Result synthesis: When an agent accomplishes its task, it passes the result to the appropriate agent (usually the original agent), since it knows the decomposition decisions and thus is most likely to know how to compose the results into an overall solution.

Before we go into Cooperative agent system we need to understand the concepts related to this system. We provide the information about agents, intelligent agents, agent architecture, multi-agent system definition and agent communication.

3.1 Agent

There is no universally accepted definition of the term agent, and there is a great deal of ongoing debate and controversy on this very subject. An agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objectives [4]

![Figure 3.1](image.png)

Figure 3.1
In the figure 3.1, the agent takes sensory input from the environment and produces as output actions that affect it. The interaction is usually an ongoing and non-terminating one. The figure gives an abstract, top-level view of an agent. In this diagram, we can see the action output generated by the agent in order to affect its environment. In most domains of reasonable complexity, an agent will not have complete control over its environment. It will have at best partial control, in that it can influence it. From the point of view of the agent, this means that the same action performed twice in apparently identical circumstances might appear to have entirely different effects, and in particular, it may fail to have the desired effect.

Any control system can be viewed as an agent. An example of such a system is a thermostat. Thermostats have a sensor for detecting room temperature which is embedded within the environment, and it produces as output one of two signals: one that indicates that the temperature is too low, another which indicates that the temperature is OK. The actions available to the thermostat are “heating on” or “heating off”.

3.2 Intelligent Agent

We are not used to thinking of thermostats or UNIX daemons as agents, and certainly not as intelligent agents. An intelligent agent is one that is capable of flexible autonomous action in order to meet its design objectives, where flexibility means three things:
Reactivity: intelligent agents are able to perceive their environments, and respond in a timely fashion to changes that occur in it in order to satisfy their design objectives;

Pro-activeness: Intelligent agents are able to exhibit goal-directed behavior by taking the initiative in order to satisfy their design objectives;

Social ability: intelligent agents are capable of interacting with other agents (and possibly humans) in order to satisfy their design objectives.

3.3 Multi-agent System

Agents operate and exist in some environment, which typically is both computational and physical. The environment might be open or closed, and it might or might not contain other agents. Although there are situations where an agent can operate usefully by itself, the increasing interconnection and networking of computers is making such situations rare, and in the usual state of affairs the agent interacts with other agents.

Multi-agent system is a system in which several interacting, intelligent agents pursue some set of goals or perform some set of tasks [5]. Agents may be affected by other agents or perhaps by humans in pursuing goals and executing their tasks. They communicate in order to achieve better goals of themselves or of the society/system in which they exist. Communication can enable the agents to coordinate their actions and behavior, resulting in systems that are more coherent.
A multi-agent system has the following major characteristics:

- Each agent has incomplete information and is restricted in its capabilities.
- System control is distributed.
- Data is decentralized.
- Computation is asynchronous.

Why should we be interested in distributed systems of agents, when anything that can be computed in a distributed system can be computed on a single computer with at least the same efficiency. There are many reasons for this. Distributed systems are sometimes easier to understand and easier to develop, especially when the problem being solved is itself distributed. There are also times when a centralized approach is impossible, because the systems and data belong to independent organizations that want to keep their information private and secure for competitive reasons.

### 3.4 Agent Communication

Coordination is a property of a system of agents performing some activity in a shared environment. The degree of coordination is the extent to which they avoid extraneous activity by reducing resource contention, avoiding livelock and deadlock, and maintaining applicable safety conditions. Cooperation is coordination among nonantagonistic agents, while negotiation is coordination among competitive or simply self-interested agents. As a team, cooperating agents try to accomplish
what the individual cannot, hence, fail and succeed together. Competitive agents try to maximize their own benefit at the expense of others, so the success of one implies the failure of others. In this comparison shopping system agents are cooperative.
4.0 Supporting Technologies

In Comparison Shopping System we have used many technologies, which are related to communication, parallel computing, and Web programming. Without having the basic knowledge of these technologies it is little bit difficult to understand Comparison Shopping System. In this part we provide a brief idea about the technologies mentioned above.

4.1 Communication Protocols

Communication protocols are typically specified at three levels. The level 1 of the protocol specifies the method of interconnection; the level 2 specifies the format or syntax of the information being transferred; the top level 3 specifies the meaning or semantics of the information. In this project we have use TCP/IP protocol. TCP or Transmission Control Protocol/Internet Protocol is the basic communication language or protocol of the Internet. It can also be used as a communications protocol in a private network. It is specifically designed to provide a reliable end-to-end byte stream over an unreliable internetwork. An internetwork differs from a single network because different parts may have widely different topologies, bandwidths, delay, packet size, and other parameters. TCP is designed to dynamically adapt to properties of the internetwork and to be robust in the face of many kinds of failures.

TCP service is obtained by having both the sender and the receiver create an end point called Socket. Each socket has a socket number (address) consisting of the IP address of the host and the 16-bit
number local to the host, which is called port. A port is the TCP name for a TSAP. To obtain a TCP service, a connection must be explicitly established between a socket on the sending machine and a socket on the receiving machine.

The main reasons for choosing TCP/IP are[3]:

a. TCP/IP is now standard into most popular operating systems, such as Unix, Lynx, MS-Windows, and NT.

b. Most programming languages such as C/C++, Java and Perl[7] support Socket programming based on TCP/IP.

The communication protocols should be shared by all agents in a system. They should be concise and have only a limited number of primitive communication acts. There are several speech acts, KQML[5], KIF[5], and ICL[5] that are invented for communication purpose among agents in a system. None of the above languages are used in this project. This project is dedicated in trying a new language, XML[8], as one of communication protocols.

4.2 Parallel Computing

There are three kinds of parallel computation that are known.

- Multiprogramming
- Multithreading
- Distributed parallel computing
4.2.1 Multiprogramming

Early computers ran one process at a time. While the process waited for servicing by another device, the CPU was idle. In an I/O intensive process, the CPU could be idle as much as 80% of the time. Advancements in operating systems led to computers that load several independent processes into memory and switch the CPU from one job to another when the first becomes blocked while waiting for servicing by another device. This idea of **multiprogramming** reduces the idle time of the CPU. Multiprogramming accelerates the throughput of the system by efficiently using the CPU time.

Programs in a multiprogrammed environment appear to run at the same time. Processes running in a multiprogrammed environment are called **concurrent processes**. In actuality, the CPU processes one instruction at a time, but can execute instructions from any active process. We have implemented this concept of multiprogramming using the function `fork()`. `fork()` is a very powerful function in Unix system that creates a child process of a process. The child process has the same parameters and running environment with its parent process. The child process can run as an independent process. Once its task is finished, the child process will be destroyed automatically.
4.2.2 Multithreading

Multithreading is the ability of a program an operating system process to manage its use by more than one user at a time and to even manage multiple requests by the same user without having to have multiple copies of the programming running in the computer. Each user request for a program or system service (and here a user can also be another program) is kept track of as a thread with a separate identity. As programs work on behalf of the initial request for that thread and are interrupted by other requests, the status of work on behalf of that thread is kept track of until the work is completed. Here we introduce this technology to make it possible so that the agent can execute more than one task in one process.

4.2.3 Distributed Computing Environment

In network computing, DCE (Distributed Computing Environment) is an industry-standard software technology for setting up and managing computing and data exchange in a system of distributed computers. DCE is typically used in a larger network of computing systems that include different size servers scattered geographically. DCE uses the client/server model. Using DCE, application users can use applications and data at remote servers. Application programmers need not be aware of where their programs will run or where the data will be located.
The SSCA is a software-distributed system based on a distributed system where all machines (PCs or work-stations) are connected by Internet. From an external view, all members of SSCA work on the same task to increase the efficiency by parallel computing. In the internal view, the relationship between members is a client-server relationship. In detail, one agent might be a client, server or both. The client agent asks service from another agent. The requested agent provides the service that the client needs. All member agents of SSCA might be running in different machines, but they can cooperate on one task by interaction over internet. For the purpose of this project multiprogramming and distributed computing are used.
5. Implementation

All the modules for this system have been developed using Perl. We have also used mySQL[12] database for storing the data. As mentioned in the previous chapter we have used XML as a communication protocol between agents. There is main html page where user enters the key word. The application domain chosen for the project is a book. The database design used this project is shown below. In this we did not show the PRICECOMPARISON and RESULTS tables, which are independent of these tables.

Fig 5.1. Database Schema

```
SAMPLEVALUES
SITEID #
RECNBR
FLDID #
DEPTH
TAGSEQ
RELPoS
KEYWORDS
OMITWORDS
VALUE

EXTRACTEDRECS
RECNBR
SITEID #
FLDID #
VALUE

TEMPLATEFIELDS
TEMPLATEID #
TEMPLATEFLDID
TEMPLATEFLDNAME

TEMPATES
TEMPLATEID
TEMPLATENAME

FINALEVALUES
SITEID #
FLDID #
DEPTH
TAGSEQ
RELPoS
MINLEN
KEYWORDS
OMITWORDS
RELPoSADJ
MINLENADJ

SITEFIELDS
SITEID #
FLDID
FLDSEQ
FLDNAME
FLDTYPE
MANDATORY_FLAG

SITES
SITEID
SITE_NAME

TEMPLATEASSOCIATION
SITEID
TEMPLATEID #
FLDID #
TEMPLATEFLDID

Legends
# - Foreign Key
Red Fields - Primary Key
```
Database schema. A database has been used to interface between the Learner, Extractor and External interface modules. The database primarily stores the rules learnt for the Web documents. We have also introduced additional tables to support interface with a comparison-shopping agent. We have used the concept of templates, which helps to comparison shop between records from different Web sites. In this project we stored the extracted records in PRICECOMPARISON table. The arrows between the various tables in the database schema indicate a foreign key constraint.

There are many modules developed for this project. We give information for most of the modules that play crucial role in this project. The important modules and their details given below ( some information taken from [2], [3]):

**Bookbot Module**

The application starts when user accesses the page search.html. This page is basically for searching the books based on title, author and ISBN. Once user enters the word then Bookbot module is invoked. These days books can be purchased from lot of websites, like the primary vendors for any product, the primary book sites are available for the books, which maintain books from all the publishers. The bookbot module crawls the list of selected websites and searches for the book and displays the list if the book is available. The bookbot module combines the results of the selected websites and displays the results back to the
web browser in the alphabetical order. The results web page provides a link for each book to compare the books based on price from the variety of websites.

**CompareISBNPrices Module**

Once user clicks on the compare prices link then CompareISBNPrices Module is invoked. The purpose of the compareISBN module is to determine the best price for the given book. Normally, if the book has to be bought from the bookstores outside, the customer has to visit and check each and every shop. The compareISBN module performs the same functionality by crawling various Web sites based on the ISBN number and, gets the price information of the book and its URL (clicking this would take to respected site for the book). This module first checks the data for the ISBN in the table PRICECOMPARISON and deletes the data if it is more than one day old. If the data for the ISBN is not more than one day old then this module fetches the data from table and shows it to the user without going crawling various Web sites.

If there is no data or data is older than one day data in the PRICECOMPARISON table this module creates the instance of ssca_r_agent. It then calls the function assign_task() of ssca_r_agent. Finally it calls the function dbGetBUYSITE(), which fetches the data, inserted by ssca_search.pm, from PRICECOMPARISON table. The part of code executed, when there is no data or data is older than one day in the table , is shown below. $searchFor contains the ISBN number.
my $client = new sscav_r_agent();
$client->assign_task($ISBN_NBR);
dbGetBUYSITE( $searchFor );

This module behaves like requesting agent as we have added requesting agent functionality to this.

We describe the functionality of requesting agent below:

1. Take order from Web page;
2. Transform orders into Shopping Task (XML format);
3. Send the Shopping Task to Manager agent then wait for the result;
4. Display the results to the user

Class sscav_m_agent

This is a derived class of sca_server_1. It is management agent of this comparison shopping system. Its main functions are

- keeping track of search agents
- decomposing tasks
- synthesizing results

It has a member variable registration table, which hold information about search agents, which are registered and listening to its requests. It has another member variable called task table, which basically keeps track of tasks.

It has a function called second_deco( ) that handles the decomposition of tasks and assigning different tasks to listening search agents. While handling the task decomposition this function checks the number of
listening search agents. If we have eight search agents and eight Web
sites then each agent would get the task of getting the price from single
store. If we have four search agents and eight Web sites then each agent
would get the task of getting the price from two stores. If we have seven
search agents and eight Web sites then each agent would get the task of
getting the price from once store but the first search agent would get an
extra task of getting price from remaining store.

Class ssca_s_agent

This is a derived class of *sca_server_1* or *sca_server_2*. It depends on
the environment where the system runs, because the *sca_server_1*
is
only supported by Unix. It is the searching agent of this system.

SSCA_S_AGENT is the module used by *s_agent_1.pl* program for creating
the search agents. Actually search agents go to different Web sites and
get the price for books. This module has function for creating the
objects of *ssca_s_agent_1*. Once we create the object then we can call
the function registration( ) for registering the search agent with
management agent.

The functions of searching agents:

1. Receive the sub shopping task from management agent;
2. Executing the task;
3. Send the result back to management agent.
The current search agent gets the price from Web sites using extractRecords function in exInterface.pl module. The sample code is shown below.

```perl
$siteSelected = "POWELS";
$siteName = $content;
($price) = &extractRecords($dbh,$siteSelected,$siteName,$tablename);
if ($price){
    $price = getDecimalPrice( $price );
    dbInsertBUYSITE($searchFor,$buySite,$price);#Inserts an entry into database
}
```

$siteSelected variable has the name of the site ( from SITES table ) for which we gave training. $siteName has the content of the result page for the ISBN from the book site. extractRecords function extracts the price from the $siteName using the extraction rules. We send the database handle ( $dbh ) to this function.

The database handle is created as shown below:
```
$dbh = DBI->connect("DBI:mysql:$dbname", $user, $passwd)
    or die "Can't connect: ". DBI->errstr;
```

We need to specify database name for variable $dbname , $user and $passwd which has access to the database. This class uses the module sscasearch.pm. In this module we have functions, to extract the book
information, for eight sites. We can add any number of new sites. If we want to introduce new Web site then we should add a new function in `ssca_search.pm`. We have to make little changes in the code in management agent (`m_agent_2.pm`) and search agent (`ssca_s_agent_1.pm`).

**Class Socket**

This is a library module that supports Socket programming. We can get it from Perl lib. It also can be obtained from C/C++, Java library. It is based on the TCP/IP communication protocol.

**Class sca_listener_1**

This is a class whose instance can listen at local host and a given port number. Once it receives a message, it calls `fork()` to generate a child process to do something. The main process keeps listening. It is used in Unix environment.

**Class sca_listener_2**

This is a class whose instance can listen at local host and a given port number. Once it receives a message, it does something and then goes back to listen. It is used in windows and NT, which does not support `fork()`.

**class sca_listener_3**

This is a class whose instance can send out a message then listen at local host and a given port number. It can be used in both Unix and Windows.
class sca_message_sender
This is a class whose instance can send a message to a given address.

class sca_client
This is a derived class of sca_listener_3 and sca_message_sender
whose instance can send a request first then listen at a local host and a
given port number. Once it receives the result corresponding to its
request, it displays the result.

dbInterface.pl
This module has all the database related functions. It has functions
dbSiteCreate, dbtemplateCreate and dbInsertTemplateAssoc for creating
sites, creating templates and associating templates. These functions
are used by Learner and Extractor modules. In extractRecords function
in exInterface.pl program we used many database functions.
They are mentioned below:
dbGetMandatoryFldIds($dbh, $siteName) : # Gets the mandatory field ids
for the site
dbGetFldIds($dbh, $siteName); # Gets all field ids for this site
dbGetFldDef($dbh, $siteName, $fldId); # Gets the field definition from
the INALVALUES table

AutoFetch.pl
This module fetches the book prices automatically when the system-load
is low. In this module the system-load is checked by calling function
getLoad. The getLoad function returns $sysLoad value. If the $sysLoad is
“High” then the program sleeps for ten minutes. It recursively checks for system-load for every 10 minutes until it is “Low”. Once the $sysLoad is “Low” then we get the ISBN list from RESULTS table. We fetch the price information automatically for each ISBN number if the data corresponding to the ISBN number in PRICECOMPARISON table is more than one day old. Once we get the price information for all ISBN numbers in ISBN list then the program will sleep for 24hrs time.

While calculating the system load we have considered the values userCpu, systemCpu and idleCpu.

5.1 Experimental Results

For finding the efficiency of the software I ran several test cases. I ran the comparison shopping system for same five books with different number of agents running on the system. The results are shown in the table 5.1 below:

<table>
<thead>
<tr>
<th>Test No./ # Agents running</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>21</td>
<td>10</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>18</td>
<td>11</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>21</td>
<td>9</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>19</td>
<td>11</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>18</td>
<td>11</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>22</td>
<td>13</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>21</td>
<td>11</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>8</td>
<td>19</td>
<td>12</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>9</td>
<td>20</td>
<td>11</td>
<td>9</td>
<td>6</td>
</tr>
<tr>
<td>10</td>
<td>22</td>
<td>13</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>Average</td>
<td>20</td>
<td>11.2</td>
<td>7.6</td>
<td>5.7</td>
</tr>
</tbody>
</table>

Table 5.1 Time consumption While All Searching Agents Are Running on the Same Machine ( Uses training approach to extract records )
From the results it is pretty clear that single agent takes a lot of time when compared to the time taken by four or eight agents running parallel. The time taken by 4 agents is almost same as time taken by 8 agents. It may be because the load is less (i.e., we are fetching information from 8 sites only). If the load is more then probably we could see the difference in times taken by 4 agents and 8 agents.

I checked the results for SSCA[3], which used pattern-matching techniques to extract records from the Web pages. The results are shown in the table 5.2 below:

<table>
<thead>
<tr>
<th>Test No./ # Agents running</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>33</td>
<td>16</td>
<td>9</td>
<td>14</td>
</tr>
<tr>
<td>2</td>
<td>28</td>
<td>27</td>
<td>54</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>31</td>
<td>18</td>
<td>10</td>
<td>104</td>
</tr>
<tr>
<td>4</td>
<td>29</td>
<td>15</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>5</td>
<td>27</td>
<td>19</td>
<td>9</td>
<td>7</td>
</tr>
<tr>
<td>6</td>
<td>28</td>
<td>20</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>7</td>
<td>35</td>
<td>21</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>29</td>
<td>18</td>
<td>10</td>
<td>7</td>
</tr>
<tr>
<td>9</td>
<td>30</td>
<td>21</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>10</td>
<td>34</td>
<td>15</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td><strong>30.4</strong></td>
<td><strong>19.0</strong></td>
<td><strong>9.2</strong></td>
<td><strong>8.8</strong></td>
</tr>
</tbody>
</table>

Table 5.2 SSCA : Time consumption While All Searching Agents Are Running on the Same Machine ( Uses Pattern matching techniques )

From these two tables it is clear that Our system, which uses training approach for extracting the records, takes less time for extracting the results. Even when one agent running our system took 20 seconds on an average to get the results where as SSCA took 30.4 seconds for the same results. From the observation we can say that our system is more efficient.
6. **User Manual**

In order to use this system we need to have modules described in the previous chapter. First we need to run the management agent \((\text{m\_agent\_2.pl})\). We have to run the search agents \((\text{s\_agent\_1.pl})\). The screen prints of when these processes running are shown below:

**Fig 6.1** screen print of management agent (m\_agent\_2.pl)

**Fig 6.2** screen print of management agent (m\_agent\_2.pl)
Once we are done with this process we have to access search.html from http://pikespeak.uccs.edu/~Project/search.html. The page is shown below:

Fig 6.3 search.html

In this html page we have to enter the search words (For example: “perl” as shown in the page), select any of the radio buttons and then click the “Search for Best Book Price” button. This triggers perl script bookBot.pl, which generates result page for search word. When I used the search word as “perl” the result page generated is shown below.
Once we get the results page then we need to click on **COMPARE PRICES FOR THIS BOOK** link. This triggers `compareISBNPrices.pl` program that would bring us the book prices at different stores. The result of
compareISBNPrices.pl program is shown below.

Fig 6.5 result page generated by compareISBNPrices.pl

We need to keep the bookbot.pl and compareISBNPrices.pl modules in `cgi-bin` directory and give the users the read and execute authority. We need to keep address.xml file in public_html directory and give the access of read and write access to users. If the software is not able to extract price from some sites then we have to train for those sites using new training samples. If the site is completely changed then we have to take special care in finding the good training samples with lot of variations in length, depth of mandatory fields. We have to make sure that the MySQL server is running all the time.
7. Conclusion

The comparison shopping system, with multiple agents running, using auto-extraction technique is more efficient than comparison shopping system with single agent. The speed and memory of the machine limit the enhancement of efficiency if all searching agents run on the same machine, so agent’s performance is machine dependent. When number of tasks is less then we cannot see the advantage of “task sharing”. If the number of tasks is more then multiple agent system with “task sharing” will be always better than single agent system.

Future Work. We have implemented automatic extraction of records from Web documents. This saves a lot of coding time when there is any change in the Web page. We need to keep track of the changes in the Web page. We can have some scripts that go to the Web sites once in a month or fortnight to check if there is any change in the page. If there is any change then it should notify us with the site name so that the Learner would be trained with new sample pages. Some of the areas that we would like to address in our future work are as follows:

Integration with a Recommendation System

Our system basically extracts the price information from different book sites for a particular book. It would be a better idea to add personalization to the system by implementing the recommendation feature.
Recommendation systems apply statistical and knowledge discovery techniques to the problem of making product recommendations during a live customer interaction and they are achieving wide spread success in E-commerce. Most recommendation systems use collaborative or social filtering methods that base recommendations on other users’ preferences. This approach assumes that a given user’s tastes are similar to another user of the system and a sufficient number of user ratings are available to make correct recommendations. If a user is looking for a particular book, then the system should recommend related books that user might be interested. This can be achieved from the knowledge recommended system gained from previous buying patterns.

**Auto-detection of change in Target pages:** The target web sites keep on changing format of their Web pages. If there is any major change then the current system cannot extract records from these pages. We need to find out, if there are any changes in these pages, regularly. It is very difficult to manually got to the sites and check the changes. It is better we create a program that goes to the sites and gets the recent page and compares with the trained page. If there is any change then it should inform someone who maintains this software. Once we know that the page is changed then we can re-train the Learner with the new page. In this way we can keep the Comparison Shopping System more stable.
REFERENCES

[1] Intelligent Agents Group (IAG), Computer Science department, Trinity College, Dublin
http://www.cs.tcd.ie/research_groups/aig/iag/pubreview/chap2/chap2.html


