Domain-dependent Information Gathering Agent

Aleksander Pivk*, Matjaz Gams

Jozef Stefan Institute, Jamova 39, 1000 Ljubljana, Slovenia

Abstract

A universal agent should be capable of gathering information from arbitrary heterogeneous sites and offer intelligent information services on its own based on information so gathered. We present a domain-dependent agent for information gathering. It can visit an arbitrary domain-related site by observing a user perform the first query. By understanding key concepts of the first query, the agent performs subsequent queries autonomously. When a user asks the agent about a particular item, the agent gathers relevant information from various sites. The major advantage of the agent is a semi-automatic creation of a wrapper around a particular site with few human interventions. We have implemented two versions of such information-gathering agents: ShinA (SHoppINg Assistant) for e-trading tasks, and EMA (EMployment Agent), which performs employment and job related functions over the Internet.

Keywords: Intelligent agent; Comparison shopping; Information extraction; Inductive learning; Wrapper

1. Introduction

Internet services offer several advantages over the classical ones, and are already changing the way companies, customers, and users perform their activities. But the rapid growth of the Internet has resulted in an explosion of available information and online services. To make things worse, the Internet is poorly structured, and disorganized (Lewis, 1999). There are millions of heterogeneous Internet sites offering various services, which makes the Internet unintelligible for any single user conducting a non-trivial task. Therefore, serviceability of the Internet depends to a large degree on the amount of automated uniform user-friendly services, systems, and tools (Levy, 2000; Pazienza, 1997).

The negative side of the Internet success is evident when one wants to gather relevant information. One of the first solutions to cope with information overload were search engines. Typically, a user provides a couple of keywords, and a search engine finds links to relevant sites.

This simple approach has several drawbacks. A search engine proposes only simple links to hopefully relevant sites. A user must still follow the links manually, and further browse for the desired information (Balabanovic & Shoham, 1995). Some of the links point to vast databases with specialized interfaces. This forces the user to get acquainted with new interfaces, and slows down the user even more. In addition, this approach favors largest information providers due to name-branding and time optimization. As a result, this largely reduces the basic advantage of the Internet. To further complicate matters, a user is unaware of changes in already visited sites unless he or she revisits the sites frequently.

A rather simple solution is to provide alert services when a relevant site changes. E.g., several e-commerce sites already allow shoppers to order price alerts that notify a shopper when the price of a product changes, or falls below a specified amount. But some of the services require lengthy surveys to be filled out before they can be used, while at the same time they may provide little or no personalization. It is also undesirable that such services weaken user privacy (Jakobsson, 1998).

Another approach involves voluntary ratings and reviews of sites by users, resulting in personalization and recommendations. Again, this takes a lot of user time and introduces problems with privacy. In addition, recommendation systems effectively reduce the size of the marketplace and introduce bias, as it is difficult to obtain a sufficient number of ratings for every existing vendor, and to control the reliability of the sources (Pivk, 2001; Basu, Hirsh & Cohen, 1998; Schafer, Konstan & Riedl, 1999).

In this paper we present solutions of the problem of information gathering. We propose a domain-dependent information-gathering agent that extends the existing approaches by providing automatic uniformity, user autonomy, and privacy. We implemented two systems: a semi-uniform intelligent agent for employment tasks (Gams, 2001) and for e-commerce (Pivk & Gams, 2000). Our principal goal was to improve accessibility and expand the benefits of existing information-gathering approaches.

The paper is organized as follows: In Section 2 we describe agents, in Section 3 wrappers, in Section 4 the algorithm, in Section 5 the implementation for employment tasks, and in Section 6 for e-trading.

* Corresponding author. Tel.: +386-1-447-3380; fax.: +386-1-425-1038.
E-mail address: aleksander.pivk@ijs.si

Related work is described in Section 7. We summarize our ideas in Conclusion.

2. Agents

In information gathering, one of the most attractive new approaches are software agents. They help automate a variety of activities, most of which are time consuming. Software agents differ from "traditional" software in the sense that they are personalized, social, continuously running and semi-autonomous (Gutmann, Moukas & Maes, 1998). In this way, the Internet is becoming more user-friendly, semi-intelligent and human-like.

There are many definitions of what the term "agent" denotes, based on different approaches, expectations and visions (Bradshaw, 1997). Shoham (1997) describes a software agent as a software entity which functions continuously and autonomously in a particular environment often inhabited by other agents and processes. The requirement for continuity and autonomy derives from human desire that an agent be able to perform activities in a flexible and intelligent manner, responsive to changes in the environment, and without constant human supervision. An agent that functions over a long period of time should be able to adapt from its experience (Liebowitz, 1999). There are two types of agents, closely related to our research: Information/Internet agents and shopping agents.

Information/Internet agents perform the role of managing, manipulating, or collating information from many distributed sources (Maes, 1994; Maes, Guttman & Moukas, 1999). The motivation for developing information agents is at least twofold. Firstly, there is an increasing need for tools that manage the information explosion. Secondly, there are also vast financial opportunities to be gained.

Shopping agents find products (the following terms: product/object/item/entity represent synonyms in this paper) under the best terms among different e-commerce sites (Dastani, Jacobs, Jonker, Catholyn & Truer, 1999; Gutmann, Moukas & Maes, 1998). A shopping agent queries multiple sites on behalf of a shopper to gather pricing and other information on products and services.

However, basic comparison-shopping agents (Greenwald & Kephart, 1999) still lack several desired properties. They introduce a marketplace that is biased in favor of those e-commerce sites that collaborate with the shopping agents. In addition to a limited number of e-commerce sites to choose from, those participating sites often do not offer the best prices. Typically, e-commerce sites that are included into agent's repertoire follow specific protocols or include hand-coded wrappers to transform data into agent-readable forms (Hammer, 1997; Muslea, Minton & Knoblock, 1998).

An ideal agent would extract contents of heterogeneous Internet databases online, and present the data to a user in a uniform way (Cowie & Lehnert, 1996). While this task is feasible by humans, current computer systems are not intelligent enough to perform it successfully as the type, amount, and organization of the information provided by databases differ from site to site (Baek, Liebowitz, Prasad & Granger, 1999). For an autonomous agent, the desired properties are uniformity, user autonomy and privacy.

Uniformity refers to the ability of an agent to automatically enter various sites and adapt to the type, amount, and organization of the information provided by various companies. Since this property demands human-level intelligence, the task is beyond current computer systems. In reality, we accept limited uniformity, i.e. *semi-uniformity*.

Autonomy refers to the idea that an agent provides the best possible service by remaining as independent as possible from both customers and vendors. Autonomy from vendors implies that the service is to remain unbiased by performing wide searches (as opposed to only searching the databases of a few "preferred" vendors). This can be achieved by progress in making interfaces more uniform, and by improved methods for interpreting potential hits. Autonomy from the customer means that users can be relieved from the tedious task of searching for information and of needing to adjust to different sites.

Additionally, our research addresses the *privacy* of the shopper by concealing the identity and behavior of the user. However, we note that the privacy provided is conditional, and should be selectively revoked if abuse is suspected. These aspects of our proposed agent provide for an unbiased marketplace where the user benefits in many respects, and safely stays hidden behind the agent who performs anonymous searches on user's behalf.

3. Wrappers

There is a classical solution for dealing with heterogeneous information sources (in our case domain-specific databases), where each source has a unique way of providing information to the user. Namely, a program may be able to integrate information from the sources if it has at its disposal a communicator-translator, generally referred to as *wrapper* (Yang, Seo & Choi, 2000), which produces a translation rule for each site.

Formally, a wrapper is a program or a rule that understands information provided by a specific source and translates it into a regular form that can be further reused. A wrapper is an essential component of a mediator system (Fig. 2), which accepts queries from users, translates each one into the appropriate query for the individual source, fetches the relevant pages from that source, extracts the requested information from the retrieved pages, and returns the result to the user. Essentially, wrappers make the Web sources look like databases that can be queried through the mediator's query language (Kusmerick, Weld & Doorenbas, 1997; Cowie & Lehnert, 1996).

Wrappers can be either hand-coded or created automatically by software agents (Muslea, 1999).

Hand-coding wrappers is a tedious and time-consuming task. In addition, a comparison agent with these manually written wrappers is not scalable because new stores are not automatically integrated.

Automatic construction of wrappers by agents for online stores is a challenging issue, mainly because HTML documents on the Web are not agent-friendly (Liu, Pu & Han, 2000). Another difficulty for automatic wrapper-generation comes from the heterogeneity of scores, in the sense that different stores employ different mechanisms for manipulating customer queries, and different styles for displaying product information. Fortunately, domain-dependent sites such as online stores are mostly semi-structured (Atzeni, Mecca & Merialdo, 1997), and we can extract at least some meaningful information without the help of semantic-based modules.

4. Information-gathering algorithm

We present our algorithm for an automatic information-gathering domain-dependent agent.

Task: Gather information from heterogeneous Internet databases with as little human intervention as possible.

Domain-dependent constraint: Heterogeneous Internet databases have to contain common concepts like job definition in employment tasks, or product description in e-commerce.

Before presenting the algorithm, we describe the basic data structures used.

4.1. Data structures

A **record** represents a product/object/item/entity, e.g., a job offer or a product offer. Here is an example of a product offer from Amazon, in this case a book:

Special Edition Using Java Server Pages and Servlets, by Mark Wutka, **Our Price: $27.99, You Save: $12.00** (30%), Used Price: $15.06, **Availability:** Usually ships within 24 hours, **Paperback** - 754 pages 1st edition (January 15, 2000), Que; ISBN: 0789724413; Dimensions: 1.70 x 9.09 x 7.36, **Amazon.com Sales Rank:** 69,177.

A **reply** usually consists of records of items in HTML. The items are stored as lists of records of text, or records in XML format.

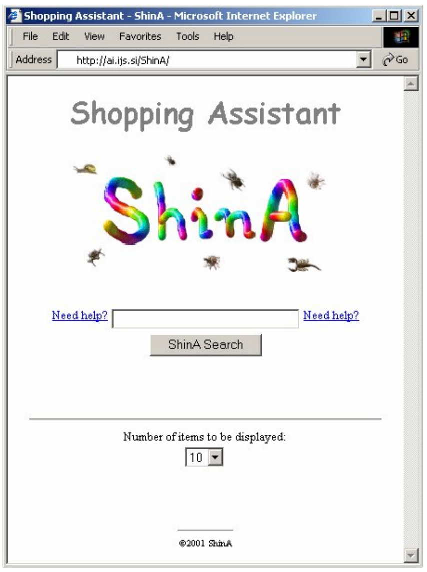
A **form** corresponds to an HTML form. There are two variations of this data structure: *empty* – corresponding to the HTML source, and *filled* – with data filled in. An example of a form is presented in Fig. 1. The relevant part of the HTML source is:

```
<form name="shinaForm" method="post"
action="shinaServlet" onSubmit="doSubmit()">
<center><table border=0 ><tr>
<td align=center><font size=-1>
<a href="Help.html">Need help?</a>
</font></td><td><center>

<input type="text" value="" name="searchStr"
size="30" maxlength="80" onChange=
"this.value=validateQuery(this.value);">

<td><input type="submit" value="ShinA
Search"></td></table></center>

<hr size=2 width="380" align="center">
<center><table cols=1 width="63%" ><tr>
<td align=center><font size=-1>
Number of items to be displayed:
</font></td></tr><tr><td align=center>
<select name="noItems" size="1" >
<option value="5>5&nbsp;
<option value="10" selected>10&nbsp;
<option value="20"> 20&nbsp;
<option value="50">50
</font></select></td></tr></table></center>
</form>
```


Shopping Assistant - ShinA - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://ai.ijs.si/ShinA/
Go
Shopping Assistant
ShinA
Need help?
Need help?
ShinA Search
Number of items to be displayed:
10
©2001 ShinA
Fig. 1. ShinA's main interface.

The first paragraph of the HTML code introduces the form name (*shinaForm*), the method (*post*), and the action (*shinaServlet*) that is performed upon query submission, a couple of parameters, and a link to Help. The second paragraph presents an input field for a text query, which gets validated by the *validateQuery* method. The third paragraph describes the ShinA's submit button. When pressed, the entered text – query, is sent to the *shinaServlet* program. The last paragraph lets a user choose the number of items per output page.

A **query** is a string of text submitted to a server through an HTML form. An example of a query searching for "Java Servlets" is:

http://ai.ijs.si/shinaServlet?searchStr=Java+Servlets (3/1)

The query consists of the address of the server (*http://ai.ijs.si*), the name of the program (*shinaServlet*), the parameter name (*searchStr*), and the entered value. Our system also attaches statistics to each query in the form of the success rate (3 attempts / 1 failure).

HTML commands are parts of a form. The agent uses them as patterns. An example of an HTML command as a form field is:

<input type="text" value="" name="query" size="30" maxlength="80" onChange= "this.value=validateQuery(this.value);">

A **query-reply** consists of a query and a corresponding reply. With it the system records questions sent by the user together with reply generated.

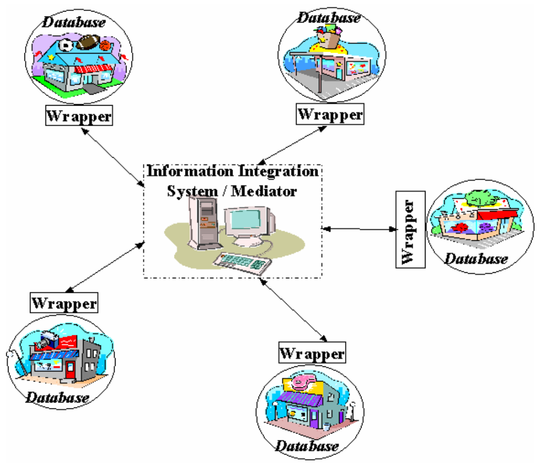


Fig. 2. An information integration system / mediator accesses databases through wrappers.

Extraction rules enable parsing of the output of a particular database in order to get a uniform output by the mediator system. Extraction rules represent the syntactic structure of database output (Ashish & Knoblock, 1997). Our implementations are rather straightforward, e.g. a rule searches for a text description inside HTML documents, separated by specified delimiters. Examples of extraction rules are:

text := string of characters
number := string of digits 0 .. 9
…
price := html.body.td[i].number
WHERE html.body.td[i].b.text = "EUR"
…
jobDescription := { text1 jobDefinition text2 delimiter}
productDescription := { ID name price text }

The first two lines are examples of basic syntactic structures like *text* and *number*. The middle part is an example of an extraction rule for *price*. It consists of a number, which is followed by a currency type, in this case EUR. The last two lines represent top-level extraction rules, describing major concepts like *jobDescription* or *productDescription*. A *jobDescription* consists of some text before an actual *jobDefinition* and some text afterwards, where a delimiter separates each *jobDescription*.

Schematically, the system communicates with heterogeneous databases using extraction rules for wrapping (Fig. 2).

4.2. Algorithm

The algorithm consists of two major phases:

1) *the initial query to an unknown database*
   The agent observes a user enter and query a new Internet database (*ObserveUser* – names of procedures are in italics). Then the agent records and parses communication between the human and the database, and extracts sufficient content (HTML *RuleExtraction*, concept matching). If necessary, the agent consults the user to get the desired level of understanding of the communication. This enables nearly error-free performance of the system.
2) *subsequent queries (MappAndMatch, Retrieve)*
   The agent performs a new query to the Internet database by simulating what a user did. If the query is successful, the agent updates the statistics for the query and presents information through a uniform interface to the user. If the query is not successful, the system updates statistics, and discovers whether the query has become invalid. If the query is found to be invalid, the system demands new rules to be extracted for a particular site. The user is notified accordingly.

When connecting to a new database for the first time, the *ObserveUser* procedure is applied:

```
procedure ObserveUser (query, commands, reply);
{A user starts communicating with a database through the agent interface. The agent remembers user commands and HTML commands without active participation. The user is in principle unaware of agent's presence.}
begin
  start observing a communication between a user
    and a database interface;
  repeat remember query and commands;
  until session is finished;
  if session finishes with error
    then forget the whole session;
    else RuleExtraction(query, commands,
      extractRules, reply);
end;
```

```
procedure RuleExtraction(query, commands,
    extractRules, reply)
{The agent stores three basic types of information:
- database description - name, URL address, title,
  country and other available information.
-  query and extraction rules - to perform another
   query in future; the query is either stored for the
   first time or modified accordingly
-  a list of HTML commands - the agent parses
   HTML commands, especially those that perform
   some action with the input/output fields. Other
   commands are most often skipped.}
begin
  store query into appropriate data structure;
  find a delimiter in the reply;
  repeat  {parse commands and create extract rules}
     find the next HTML command in the reply;
     if this command fits a rule
         then add the rule to a list of extractRules;
         else ignore the command;
   until all commands are parsed;
   set pointers to establish query-reply structures;
  {end parsing commands, start parsing the reply in
   order to extract concepts}
   repeat
      find the first concept/pattern/item (e.g., a job, a
        product ... );
      add the concept/pattern  into the corresponding
        list of entities {mapping};
   until document is parsed and transformed into a list
      of entities for the mediator system;
end;
```

The procedure for creating extraction rules establishes connections between the parsed HTML document and the domain-dependent concepts. In this way, background knowledge in the form of ontologies is used to capture the meaning of essential parts in the HTML document. For example, in the domain of employment tasks, ontologies can be used to identify professions.

If the RuleExtraction procedure fails at any stage, it calls for a supervisor intervention. On demand, it shows all intermediate results, e.g. a list of extracted rules, which the supervisor can modify at will. Therefore, if the system decomposes the process well, no human help is needed. In any case, at the end of the session the system assumes that the parsing and mapping procedures are flawless. This does not mean that no mistake can occur in the future, because a page can change.

The "Retrieve" procedure connects to databases of previous queries, where queries are modified to reflect user’s question. The "Retrieve" module connects to the database in two ways:

- if exact query was previously successfully performed, the agent repeats it;
- otherwise the system tries to create a query, based on a similar query. The best way to do this is to “understand” by ontologies, which queries qualify as promising candidates. For example, if a user asks for a specific type of shoes, all shoe-related shops and all shoe-related previous queries seem to be a good basis for a new query. The new query is constructed by changing previous entities with the specific type of shoes.

```
procedure Retrieve(query, replies);
{get info from a query database and write it into
replies, consisting of a list of entities with
corresponding parameters and their values}
begin
  find all queries with identical input fields to the one
     typed by a new user;
  if no queries found then CreateQuery (queries);
  for all queries do
    extract database address from a query and connect
     to the Internet address and obtain lastReply;
    if an error is obtained then
      if query is invalid then begin
        RuleExtraction(query, null, extractRules, null);
        notify the user;
        if RuleExtraction not successful then
             notify the supervisor;
      end;
      else notify the supervisor;
    else begin
     MappAndMatch (lastReply, replies);
     update statistics;
   end;
   show extracted replies to the user through the
     mediator system;
end;
```

```
procedure CreateQuery (queries)
{create a new list of queries based on previous
similar ones}
begin
  apply an ontology to create a list of new queries;
  for all previous queries that correspond to similar
     items
  do begin
    replace items with new ones;
    set statistics for new queries;
  end;
end;
```

There are different types of ontologies, some of them hard-coded into the system and others consisting of simple or complex data structures. For example, if a query corresponds to a form with just one input field for a database search, and the user enters just one input field in the mediator form, then this is a perfect match.

Another ontology is based on clustering of databases based on the entities they provide (i.e., kinds of products).

Other ontologies consist of a list of entities that can be entered into a specific input field. For example, practically all job definitions are defined in a national catalog.

Ontologies play a substantial role in construction of modifications from a list of queries and commands. For example, a full "looking-for-jobs" query can be described in a form with a flexible frame consisting of many subfields. The query contains information about job profession and additional specifications of area, income, type of work etc. The routine checks that all common major items, such as profession, are filled in and that modified commands are (hopefully) related to the same job specification.

```
procedure MappAndMatch (lastReply, replies)
{based on extracted rules, parse the lastReply and
add each extracted entity with corresponding values
into replies, i.e. into the mediator system}
begin
  repeat
    match each extraction rule to the lastReply;
    when the first rule triggers, delete the matched top
      of the lastReply;
    if the extracted transformed text corresponds to a
      part of the entity
      then add it into replies;
  until end-of-lastReply;
end;
```

Advanced successful performance of the system is based not only on ontologies but also on memory-based learning. The ontologies are more or less static and sometimes even hand-coded. They represent basic knowledge about a specific problem domain. Some ontologies are dynamically created or instantiated, e.g. those describing shops, which makes it possible to search similar shops. Namely, each shop typically presents its product catalogue, basic orientation, and structure.

While ontologies are domain-specific, the memory-based learning is a plain uniform implementation of the basic technique published elsewhere (Kitano, 1993). The algorithm was only slightly modified according to the problem domain characteristics. The algorithm performs memory-based learning by storing queries and statistics. The system learns each time a pattern is successful or not by changing its statistical success rate. In this way, the system adapts to the desires of the users and to changes in the domain-related sites on the Internet. Our algorithm is described in (Gams 2001). Additional learning in ShinA is described in Section 6.

The system was implemented in two problem domains: employment and e-trading. While the two implementations vary a lot in terms of programming languages and technical details, the algorithm is practically the same. Both implementations offer information about desired objects, which differ in properties, benefits and costs. Users browse through a large number of potential candidates, and compare several similar possibilities to finally choose specific objects.

5. Employment information gathering

EMA (EMployment Agent) is an intelligent employment agent for providing employment information in a way similar to human employment agents (Gams, 2001). An Internet-based intelligent employment agent should perform like a human agent, provide basic information about available jobs and available workers, and try to match these two databases as efficiently as possible (Fig. 3).

In employment tasks, users typically want to get a job - any job, a good job, better than a current job. In this way, a typical task of the agent is to find relevant information about a specific job offer (analogy – a specific e-commerce item) from various information sources.

EMA is an intelligent agent that provides user-friendly information on demand, or when it notices relevant information on the Internet. EMA can store search patterns, perform search on its own or following direct commands, and it can send online replies or offline emails regarding vacant jobs or available workers. EMA can store and observe interesting Web sites chosen by users, and match available jobs and workers.

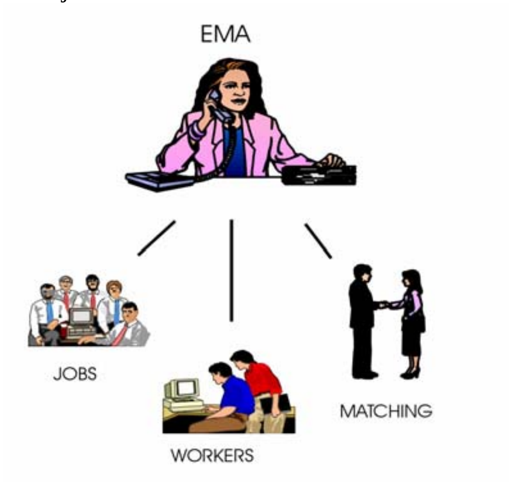


Fig. 3. The basic task of the EMA Internet-based intelligent employment agent is to provide employment information as human agents do.

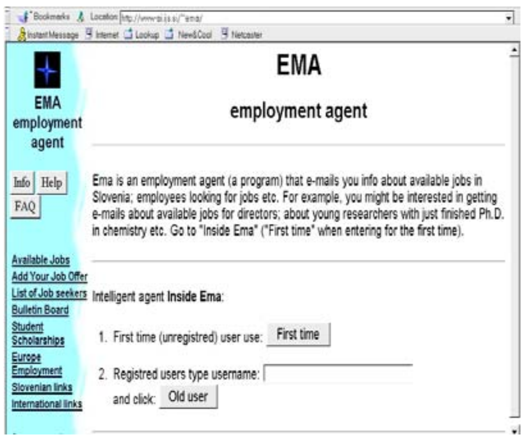
EMA has been implemented a couple of years ago with several modules added subsequently. Ema also incorporated some language and translation capabilities, such as Slovene text-to-speech interface. It also included ontologies for profession definitions, and the ability to connect to arbitrary employment sites on its own.

Natural language processing.

A common input to EMA is a partially constrained Slovenian domain-dependent text, often consisting of phrases and form inputs. In bulletin boards, the language is a matter of choice, but titles and information are either in Slovene or in English. Consequently, majority of text is either Slovene or English, with a couple of exceptions. No censorship is performed regarding language or specific details as long as the input is dedicated to the desired employment task and inside minimal decency requirements.

The system is capable of translating Slovene text into English. The translation is based on a dictionary consisting of up to four words observed previously in the employment data. In the worst case, new combinations are translated word-by-word and stored for further overview by humans. Stored combinations are sorted by frequency and translated by humans if reasonable. In addition, the translation system looks into the morphology dictionary to capture different forms of the same words. Slovene has a rich morphology so this is essential for good performance. Finally, a spell-checker module corrects spelling errors.

The translation is currently not yet at the level performed by systems translating between larger European languages; however, it is sufficiently good to enable basic understanding of translated text, since the employment syntax is quite limited. The Internet site of the EMA system is presented in Fig. 4 (http://www.ess.gov.si).



EMA

employment agent

Ema is an employment agent (a program) that e-mails you info about available jobs in Slovenia; employees looking for jobs etc. For example, you might be interested in getting e-mails about available jobs for directors; about young researchers with just finished Ph.D. in chemistry etc. Go to "Inside Ema" ("First time" when entering for the first time).

Intelligent agent **Inside Ema**:

1. First time (unregistred) user use:
2. Registred users type username: and click:

Fig. 4. Home site of the intelligent employment agent.

Ontologies.

EMA performs some of its advanced functions based on ontologies. In this way, EMA understands the basic employment tasks. One of the most common employment tasks (besides basic queries) is to find a relation between different job descriptions and jobs definitions. This is in fact one of the basic concept of all employment tasks. E.g., from a basic search query consisting of applicant's properties and desires, one should extract relevant job offers. Conversely, from a job definition one should look for available workers. With machine and statistical text-learning methods (Freitag, 1998; Mladenic, 1999) we (Bezek & Gams, 2001) have designed ontologies, i.e. meta-knowledge about job definitions from interesting words.

Semi-uniformity.

During the first years of the EMA system, several competing Internet-based employment sites were designed. Since EMA was a national employment site sponsored by Employment Services of Slovenia (ESS), an idea emerged to represent also the information from other employment sites. Instead of handwriting wrappers around each particular employment site, we have designed an algorithm that semi-automatically (Ashish & Knoblock, 1997; Kusmerick, Weld & Doorenbas, 1997) attaches to other employment sites and then automatically gathers information from those particular sites. Although the algorithm published in (Gams, 2001) was not fully implemented, we have designed two slightly modified modules of this kind and implemented them in EMA. Another version was independently designed by a student group (JobProvider) on the basis of public lectures and presentations. All versions were functional for at least some time, and on average for several years, thus showing that the idea is sound. On the basis of these experiences we have decided to generalize the algorithm, and design another system, the ShinA system, for e-commerce tasks (Pivk & Gams, 2000).

EMA is a big software system especially for our R&D group of intelligent systems at the Jozef Stefan Institute. It consists of several modules and submodules, written in different programming languages and runs on several platforms and operating systems. The system is a 30 thousand lines program written mainly in C, and also in other languages and with Internet programming tools. Together with text and data it occupies 30 MB of space.

The EMA agent was, and still is, among the most successful applications of intelligent agents in Slovenia and in Central Europe. In the first year of its implementation, our country was the third in Europe to offer national employment information through the Internet. At that time, we were the first country in the world to provide over 90% of all nationally available jobs on the Internet. On the other hand, in absolute terms there are employment systems in big countries or employment systems connected to major Internet information providers such as Yahoo or AltaVista that provide orders of magnitude bigger amounts of employment information.

EMA was among the most often-visited non-entertainment sites in Slovenia. At its peak, there were over 200.000 visits monthly, which represents one tenth of country's population. Most visitors were driven by unemployment, or by the desire to find a better job.

6. E-trading information gathering

Another implementation of our system is ShinA (SHoppINg Assistant)–a mediator system that automatically collects product description from a number of online stores on user's behalf. ShinA performs a different task than EMA – that of a shopping assistant. ShinA was reprogrammed from scratch, but still based on experiences and the common algorithms of EMA. ShinA's crucial ability to semi-automatically gather information from various sites replaces hand-coded wrappers of EMA. (Yang, Lee & Choi, 2000).

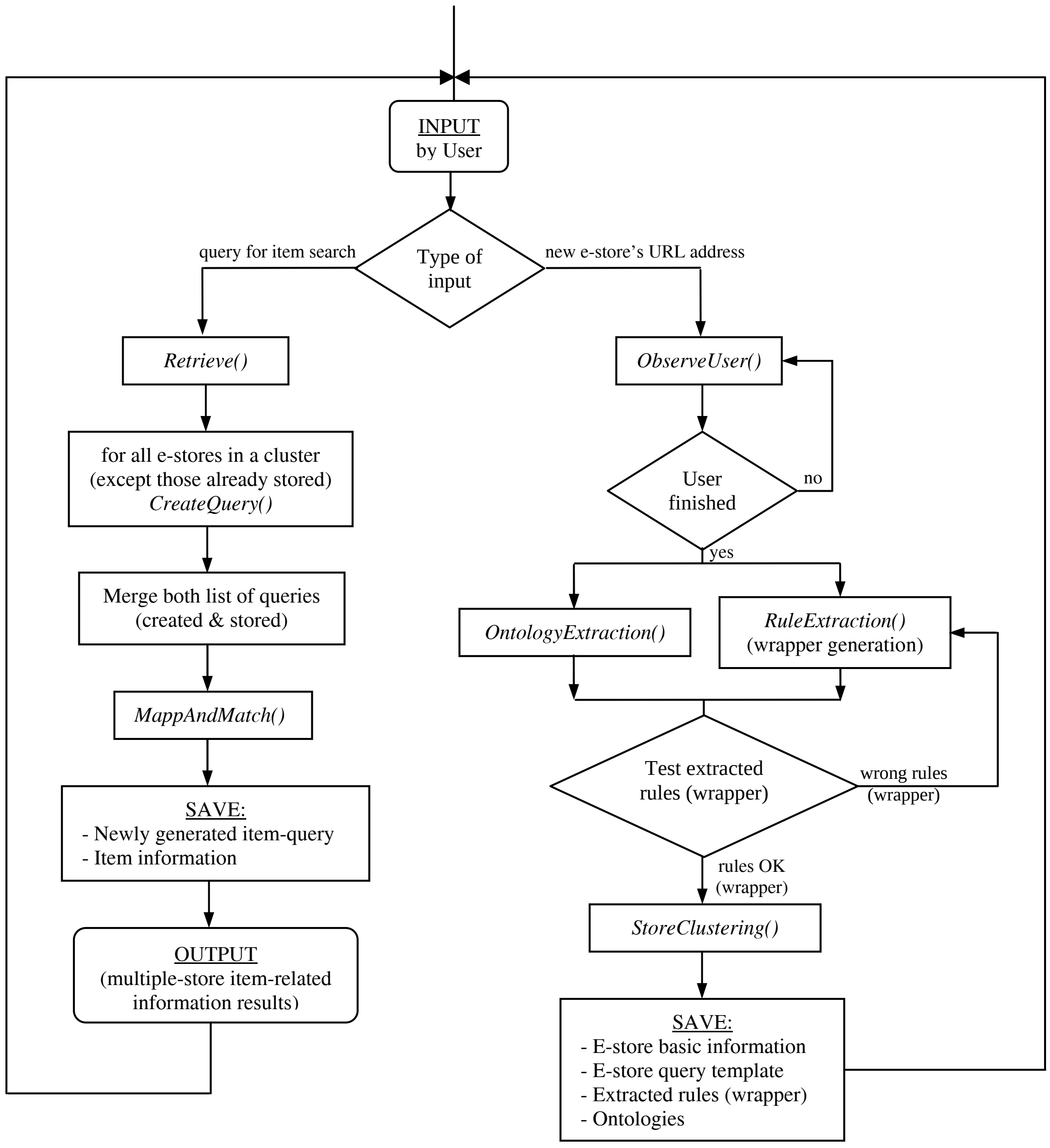


Fig. 5. Workflow diagram of the ShinA system.

6.1. Workflow of the agent

The mediator system enables a user to enter two types of data. The first type is a new e-store URL address and the second type is a request for multiple-store product information (Fig. 5).

In the first case, where a user enters a new e-store URL address, the system observes the user's communication with the e-commerce site. Since e-commerce sites have various input mechanisms, the system must locate the position of a query form that accepts user's search-for-item requests, and must be able to intelligently parse at least the most relevant parts of it. The user's communication with the e-commerce site finishes either by entering a search query into an input field of the form or by selecting an ontology (i.e., product catalogue link). This suffices to generate a query template. In this way, ShinA generates an appropriate query template for each online store.

Afterwards, the mediator system applies a learning algorithm, described in detail in Section 6.3. In the learning phase, appropriate extraction rules are created that correspond to a most representative pattern (i.e. description of an item). In this phase, ontologies are also recognized and extracted. Evaluation and testing of extracted rules is performed by extracting product information from a randomly retrieved e-store's product page. In case when the extraction of products' description is unsuccessful, new pattern and extraction rules have to be discovered. The system must be able to ignore redundant and unnecessary fragments of a page. Furthermore, it has to delineate a product description and recognize the attributes of a product such as the price, the manufacturer, the availability, the special offer, the size, etc. Next, the e-store is classified into one or more corresponding clusters (used for later search of items), using learned ontologies and basic information about the e-store. Finally, basic information, query template, rules and ontologies are stored into a knowledge database.

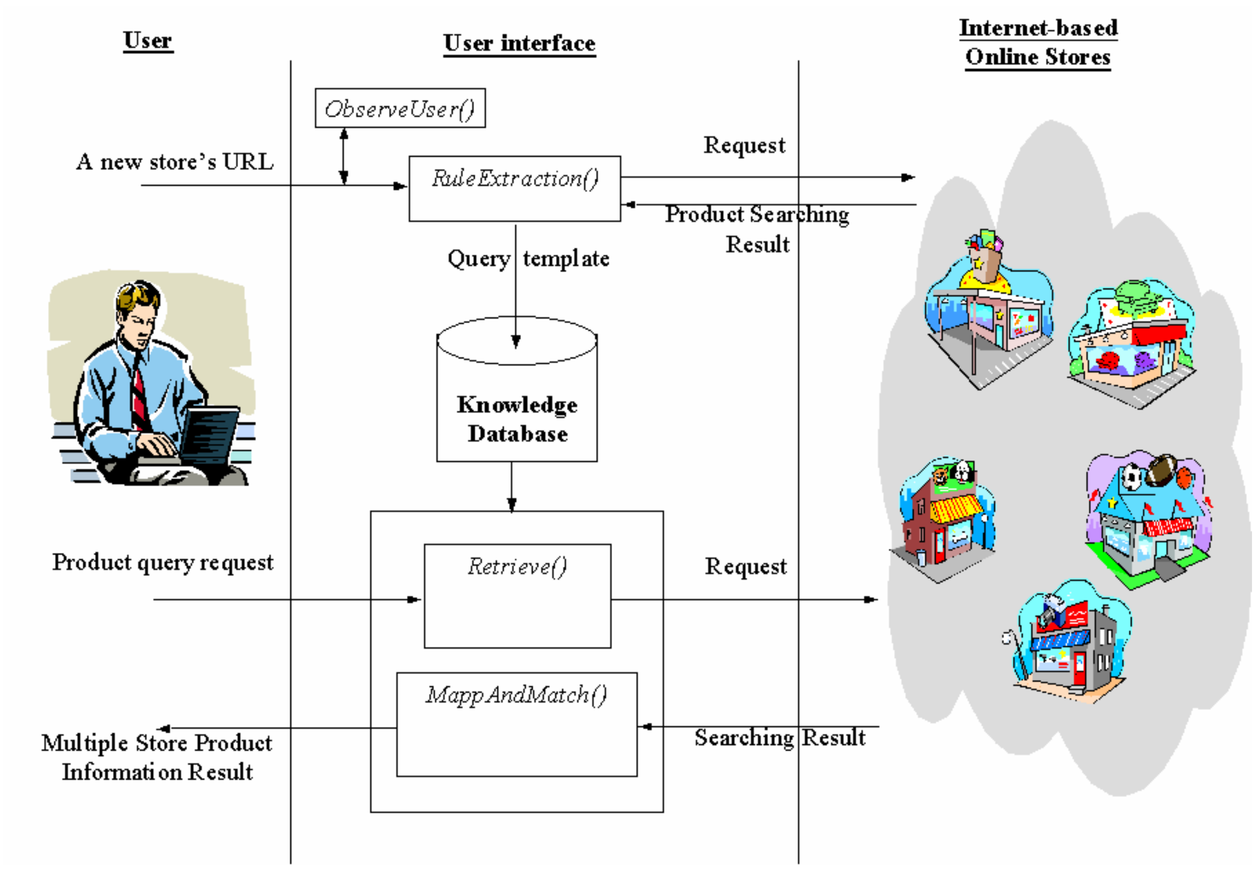


Fig. 6. An overall architecture of ShinA.

In the second case, where a user enters a keyword search for an item, the system first checks whether an equal or similar search has ever been requested. If so, each stored query (linked with user's item search) is modified accordingly and added to the list of potential queries. Otherwise, the system takes the most promising cluster of e-stores and creates queries based on query templates. The created queries are then added to the list of potential queries. Next, each potential query is performed on the corresponding e-store, which responds with a page, hopefully containing information about sought after items with their descriptions. The system tries to extract product information from each retrieved page with appropriate extraction rules. The successful queries and related product information are then stored into Knowledge database.
Finally, product information (from various sites and various formats) is presented to the user in a user-friendly uniform way.

6.2. System architecture

The common data structures, the algorithm and ontologies have already been described in Section 4. Here we describe ShinA in this framework as a particular implementation of the algorithm. Fig. 6 shows the architecture of our ShinA comparison-shopping agent. The task of the observing module *ObserveUser()* is to transform requested pages into a form that enables the system to control and supervise user's communication with e-store sites. The wrapper generator *RuleExtraction()* is the main learning module that constructs a wrapper for each particular store. The wrapper generator learns two things: it recognizes a store's query scheme and ontologies, and it learns how to extract a store's content. The wrapper interpreter *Retrieve()* is a module that executes generated wrappers to get product information. This module forms several actual queries by combining each store's query template with the keywords that the user actually typed in, and sends them to the corresponding shopping sites. The search results from the stores are then collected and fed to the uniform output generator module *MappAndMatch()*. The output generator integrates all the search results and generates a uniform output. Information such as extracted rules, query templates, ontologies, item-related information, e-store-related information, etc., is stored into the Knowledge database.

6.3. Learning Algorithm

The main learning task specific to ShinA is to build a rule from one or several resulting pages. Such a rule is used to learn the format of product description from successful searches at a new shopping site. We use an inductive learning mechanism to accomplish this task. Here, the examples correspond to the pages of a search result, and the concept to be learned corresponds to the extraction rule (Yang, Seo & Choi, 2001).

Each page contains one or more product descriptions that matched the query. A product description is composed of a sequence of product attributes. For example, a music store displays search results in which the attributes are the CD title, the artist name, the price, etc.

The wrapper-learning module has to find the starting and ending position of the list of product attributes within the result page, and recognize the pattern of a product description. To do this, our method is divided into three phases. In the first phase, the HTML source of the page is partitioned into three parts. The first and the last part are redundant and irrelevant fragments of the page (header, advertisement, script) that must be ignored. The useful middle part must be extracted, and consists of a set of attributes that describe the product. If there is more than just one product description on the resulting page, more different patterns of product representations may occur. In the next phase, the algorithm recognizes product attributes by examining HTML tags (delimiters) and categorizes them, accordingly. The product description is thus viewed as a sequence of categories, and the algorithm finds repeating patterns in it. The most frequent pattern becomes the representative product description.

6.4. Implementation

ShinA's Web interface is shown in Fig. 1. Unlike EMA, ShinA is a dedicated system without modules for translation, speech, and large amount of locally stored data. However, the core task of parsing and understanding heterogeneous sites is more difficult in e-commerce than in employment because of higher diversity of e-stores.

The ShinA system has been implemented on a Windows 2000 operating system, running Apache HTTP Server, version 1.3. As developing tools, JDK 1.3.1 and Jakarta Tomcat 3.2 were used. The system is written in Java and is based on servlet/JSP. It consists of approximately 10 thousand lines of code. Tomcat is a servlet container that is used in the official Reference Implementation for the Java Server and JavaServer Pages technologies. Both Apache Server and Tomcat are developed in an open and participatory environment and therefore freely available.

7. Related work

BargainFinder (Krulwich, 1995) and Jango (Jango) are first-stage comparison shoppers. They specify functions that agents must have in order to be applied to Electronic Commerce. Both systems employ the manual rule extraction method.

ShopBot (Doorenbos, Etzioni & Weld, 1997), like our system, suggests an automatic rule extraction technique by analyzing and learning the shopping sites. In order to integrate specific product information, Shopbot first removes irrelevant information such as advertisements by using an inductive learning mechanism and then extracts necessary product information. Shopbot, however, uses several strong biases about the structure of HTML and the display format of product search results to learn. Therefore, Shopbot is unable to learn a shopping site that does not conform to these strong biases. By contrast, the only bias in our method is that the result of a product search should be displayed in a semi-structured way, that is, each product description unit has the same output format, which conforms to almost all shopping sites. We expect our method to function more robustly than Shopbot. The thesis can be verified by empirically testing the success rate of correct wrapper constructions. We plan to measure the performance of our agents in the near future.

PersonaLogic is a comparison-shopping system that compares specific products themselves rather than shopping sites.

Kasbah, AuctionBot, and Tete-a-Tete (see References) are negotiation mediators with which users can buy and sell products based on negotiation strategies between agents in the virtual marketplace. They do not enter e-commerce sites in an automatic or semi-automatic way like our agents do.

To our knowledge, today's commercial comparison shoppers, including MySimon, PriceWatch, and BottomDollar all employ manual rule extraction methods, and consequently suffer from reduced scalability, and the ability to incorporate new e-stores and adapt to changes in incorporated stores.

8. Conclusion

We have proposed a general domain-dependent information-gathering agent. The system has been implemented in several versions of the two basic applications: EMA for employment, and ShinA for e-commerce. The agent successfully constructs correct wrappers for "reasonable" domain-dependent databases without assuming many structural constraints.

The agent contains a quick, simple and robust inductive learning algorithm that automatically generates wrappers. The extension of the learning method is memory-based learning of success rates of particular queries. This makes it possible to adapt to user habits and to changes at visited sites.

There are some limitations in our current system. Firstly, we have assumed that users give a proper URL and path for the test query. Secondly, each major concept must contain a uniform structure. We think that this is not a severe restriction since most domain-related databases stores that produce semi-structured entity information, contain the same attributes. Thirdly, we only extract the major information from an entity description that may also contain several other attributes. Lastly, the system relies heavily on HTML. If an Internet site provides information exclusively by embedding it in graphics or using Java, the system will be unable to handle the site.

Overall, the idea of semi-automatic information-gathering from heterogeneous Internet-based databases is becoming technologically mature. The problems we have been facing during implementations were not of principal matter. Technically, similar systems seem to be close to full-scale implementation. One can imagine a uniform e-shopping agent visiting and gathering information from all e-shopping databases on the Internet.

We believe that this technique can also be applied to other information integration systems for heterogeneous information sources.

Acknowledgement:

Financial support was provided by an international project INCO-Copernicus 960154, Cooperative Research in Information Infrastructure, CRII, by the Ministry of education, science and sports in Slovenia, and by ESS. We would like to thank the CEO of the Employment Service of Slovenia, Mr. J. Glazer.

References

Ashish, N., & Knoblock, C. (1997). Semi-Automatic Wrapper Generation for Internet Information Sources. In *Proceedings of Second IFCIS Conference on Cooperative Information Systems (CoopIS).*

Atzeni, P., Mecca, G., & Merialdo, P. (1997). Semi-structured and Structured Data in the Web: Going Back ad Forth. *ACM SIGMOD Workshop on Management of Semistructured Data*, pp.1-9.

AuctionBot. http://auction.eecs.umich.edu/

Balabanovic, M., & Shoham, Y. (1995). Learning Information Retrieval Agents: Experiments with Automated Web Browsing. In *Proc. of the AAAI Spring Symp. on Information Gathering from Heterogeneous, Distributed Environments*.

Baek, S., Liebowitz, J., Prasad, S.Y., & Granger, M.J. (1999). Intelligent Agents for Knowledge Management - Toward Intelligent Web-Based Collaboration within Virtual Teams. In *Knowledge Management Handbook*, (ed. Jay Liebowitz), CRC Press, (11-1 - 11-23). Washington, DC.

Basu, C., Hirsh, H., & Cohen, W. (1998). Recommendation as classification: Using social and content-based information in recommendation. In *Proceedings of the 1998 Workshop on Recommender Systems*, AIII Press, pp.11-15.

Bezek, A., & Gams, M. (2001). An agent that understands job description. *Informatica (Ljubljana)*, 25(1), pp.99-105.

BottomDollar. http://www.bottomdollar.com/

Bradshaw, J. M. (1997). *Software Agents*, AAAI Press, Menlo Park, California.

Cowie, J., & Lehnert, W. (1996). Information Extraction. *Comm. of the ACM, 39(1)*, pp.80-101.

Dastani, M., Jacobs, N., Jonker, Catholyn, M., & Truer, J. (1999). Modelling User Preferences and Mediating Agents in Electronic Commerce. In *Proc. of the Agent-Mediated Electronic Commerce (AMEC) SIG-Meeting of AGENTLINK.*

Doorenbos, R., Etzioni, O., & Weld, D. (1997). A Scalable Comparison-Shopping Agent for the World Wide Web. In *Proceedings of the First International Conference on Autonomous Agents*, pp.39-48.

Freitag, D. (1998). Information Extraction from HTML: Application of a General Machine Learning Approach. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI-98).*

Gams, M. (2001). A Uniform Internet-Communicative Agent, *Electronic Commerce Research, 1,* Kluwer Academic Publishers, pp.69-84.

Greenwald, A.R., & Kephart, J.O. (1999). Shopbots and pricebots. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, pp.506-511.

Gutmann, R. H., Moukas, A. G., & Maes, P. (1998). Agents as Mediators in Electronic Commerce, *Electronic Markets*, 8(1), pp.22-27.

Hammer, J., Garcia-Molina, H., Nestorov, S., Yerneni, R., Breunig, M., & Vassalos, V. (1997). Template-based wrappers in the TSIMMIS system. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp.532-535.

Jakobsson, M., & Yung, M. (1998). On assurance structures for WWW commerce. In *Financial Cryptography '98*.

Jango. http://www.jango.com/

Kasbah. http://kasbah.media.mit.edu/

Kitano, H. (1993). Challenges of Massive Parallelism, In *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, pp.813-834.

Krulwich, B. (1995). Bargain Finder agent prototype. *Technical Report*, Anderson Consulting. http://bf.cstar.ac.com/bf/

Kusmerick, N., Weld, D.S., & Doorenbas, R. (1997). Wrapper Induction for Information Extraction. In *International Joint Conference on Artificial Intelligence*, pp.729-735.

Levy, A.Y., & Weld, D.S. (2000). Intelligent Internet Systems. *Artificial Intelligence, 118*, pp.1-14.

Liebowitz, J., (ed.). (1999). *Knowledge Management Handbook*, CRC Press, Washington, DC.

Liu, L., Pu, C., & Han, W. (2000). XWRAP: An XML-enabled Wrapper Construction System for Web Information Sources. In *Proceedings of the Sixteenth International Conference on Data Engineering*, pp. 611-621.

Lewis, T. (1999). Microsoft Rising : ... and Other Tales of Silicon Valley, ACM.

Maes, P. (1994). Agents that reduce work and information overload. *Comm. of the ACM, 37(7)*, pp.31-40.

Maes, P., Guttman, R., & Moukas, A. (1999). Agents that buy and sell. *Comm. of the ACM, 42(3)*, pp.81-91.

Mladenic, D. (1999). Text-learning and related intelligent agents. *IEEE EXPERT, Special Issue on Applications of Intelligent Information Retrieval*.

Muslea, I., Minton, S., & Knoblock, C. (1998). Wrapper Induction of Semistructured, Web-based Information

Sources. In *Proceedings of the Conference on Automatic Learning and Discovery*.

Muslea, I., Minton, S., & Knoblock, C. (1999). A Hierarchical Approach to Wrapper Induction. In *Proceedings of the Third International Conference on Autonomous Agents*, pp.190-197.

MySimon. http://www.mysimon.com/

Pazienza, M.T. (ed.), (1997). Information Extraction, Springer.

PersonaLogic. http://www.personalogic.com/

Pivk, A. (2001). Item-based Collaborative Filtering Algorithms, In *Proc. of the Fourth International Conference on Information Society*, pp. 65-68.

Pivk, A., & Gams, M. (2000). E-Commerce Intelligent Agents, In *Proceedings of ICTEC'00*, pp.418-429.

PriceWatch. http://www.pricewatch.com/

Schafer, J., Konstan, J., & Riedl, J. (1999). Recommender systems in e-commerce. In *Proceedings of ACM Conference on E-Commerce*, pp.158-166.

Shoham, Y. (1997). An Overview of Agent-oriented Programming, In *Software Agents* (ed. Bradshaw, J. M.), AAAI Press, Menlo Park, California, pp.271-290.

Tete-a-Tete. http://ecommerce.media.mit.edu/tete-a-tete/

Yang, J., Seo, H., & Choi, J. (2001). MORPHEUS: A Customized Comparison Shopping Agent. In *Proceedings of the 5th International Conference on Autonomous Agents (Agents-2001)*, pp. 63-64.

Yang, J., Lee, E., & Choi, J. (2000). A Shopping Agent that Automatically Constructs Wrappers for Semi-Structured Online Vendors. *Lecture Notes in Computer Science, Vol. 1983*, pp.368-373.