Buyer agent to Enhance Consumer Awareness: SAATHI

Sandip Sen, Sabyasachi Saha and Karina Hernandez

Department of Math & CS
University of Tulsa
Tulsa, Oklahoma, USA
{sandip,saby}@utulsa.edu

Abstract

Personal agents have been developed that assist a user with information processing needs by generating, filtering, collecting, or transforming information. On the other hand internet stores are providing services customized by the needs and interests of individual customers. Such services can be viewed as "seller's agents" whose goal is to push merchandise and/or services on to the users. This leads us to believe that there is a growing need for deploying "buyer's agents" whose goal is to best serve the consumer's interests. The Internet contains a huge volume of information which can overwhelm a buyer. The buyers may often make misinformed decisions based on partial, outdated, irrelevant or incorrect information. We have identified several key functionalities of buyer's agents whose goal is to reduce information overload and improve relevancy and accuracy of information for consumers. In particular, such agents can make consumers aware of complex interactions between specified preferences and prevailing market conditions, provide differential analysis for decision support, and use domain ontologies to help the user reformulate queries to improve satisfaction with query results. We present SAATHI, a prototype buyer's agent that demonstrate some of these functionalities in an apartment locator domain.

Key words: Electronic commerce, personal agents, query reformulation.

1 Introduction

The advent and easy accessibility to the Internet has allowed the average citizen to participate in the global on-line market. One can find e-commerce sites burgeoning all over the Internet. While some of these sites are targeted towards business-to-business transactions, many sites are geared for end user to business interactions [1–3]. E-commerce applications that fall into the latter

category are of relevance to our discussion. With phenomenal success of e-commerce sites like *Amazon* and *eBay*, the entire business model appears to have been reinvented. Not only does e-commerce provide new avenues for selling goods and services and reduced capitals required for marketing and advertising, it also provides the key to tap into a rapidly expanding customer base. The netizens and web surfers are empowered by the possibility to browse the goods and services available from the safe haven of their home and are increasingly comfortable in buying goods and services without the assurance of having checked out the offering 'in person'.

The Internet, however, also generates an abundance of information which is often overwhelming. Search engines return too many links in response to queries. This limits their usefulness as it becomes difficult for the user to differentiate between relevant and irrelevant data. Typical e-commerce sites are developed by merchants and retailers whose goal is to profit by selling as many products and services as possible to the consumer visiting their sites. In addition, the average consumer may be inundated by the volume and diversity of information available on the web and may not have the patience or the time to search and shift through all the available information to make a judicious choice. Often word-of-mouth recommendations, which may have been outdated by a rapidly changing market, will be used to make purchasing decisions. Sometimes the consumer may not be aware of the fact that minor changes in queries may produce a much more profitable choice. All of these factors taken together suggest that though the Internet, and in particular the searching capability provided by WWW portals, can lead a consumer to an e-commerce site of interest to the user, the user can still benefit from additional tools and techniques to select products and services.

The Internet has also enabled the development and deployment of software agent based e-commerce applications [4,5] including sites that allow users to buy and sell goods [6], auction houses [7,8] etc. As a result, agent technology has caught the attention of both application developers and system designers [9] over the past few years. Agents are viewed as a useful metaphor both for developing desktop software designed to assist a particular user [10], as well as for internet-based server-side software that enables e-commerce [4]. The first wave of agents to catch our attention were those that enabled us to skip some of the grunt work, e.g., filtering e-mail [10], scheduling meetings [11], collect newsgroup articles [12], etc. Most of these were desktop applications. With the rapid explosion of the internet and the World Wide Web a different class of agents came to the fore: agents that can gather and collate information on behalf of their user [13]. With simple queries, the user could now perform powerful searches that would have previously required considerable time and effort on his/her part. Personalized web-page recommender systems [14,15] provide a partial solution to this problem as they can suggest web pages of interest to a user based on his/her query and usage patterns [16–19]. For users with

multiple and overlapping interests, *opportunistic exploration* [20] provides the chance to view products in which he/she might be interested in based on last few product selection, long term and demographic interests. Collaborative filtering mechanisms have also been developed using which users can obtain or view choices of similar users [21,22].

The remarkable growth in agent-oriented internet-based applications is encouraging. However, most of these applications appear to open up new possibilities or choices for the user without providing much guidance or help about how best to use this additional information. Though there does exist considerable research in comparison shopping agents [23,6], these agents are not designed to educate the customers about the changing marketplace or the relationships between user preferences. Also the emphasis in most of these systems is on finding the cheapest price for a product. We are more interested in providing guidance and information to support the product selection process.

Our goal is to further enhance the scope of software agent applications by developing agents whose purpose is to educate the user to become a more informed consumer. These agents will serve the interest of the user by understanding the user's goals and will recommend products/services or suggest modification to user queries or requirements that is more likely to produce results that improves user satisfaction. In particular, these agents will have to educate the user both about possible interactions between his/her preferences for different features and the dynamics of a rapidly changing marketplace.

We now use a few example scenario to illustrate our proposed functionality of a *buyer's agent*:

Product feature Selection: User A was looking for a lawn mower with 22" swatch, at least 4.5HP Briggs & Strattan gas engine, mulching option, and at least two years manufacturer warranty in a price range of less than $300. A shopping agent can perhaps find one or more such product. But consider the scenario that all but one manufacturer X, has just entered this market and hence has no product history to demonstrate quality or reliability. provide warranties of only up to a year. In this case the user requirement of a 2-year warranty will eliminate all other options and restrict the results to only the products offered by manufacturer X. This may also result in an escalated price (even though it is still within user specification, it does not necessarily mean the user will be willing to pay 25% more for example for the extended warranty period), or elimination of other features offered by other manufacturers. The "buyer's agent" should inform the user of the implied constraint "$Warranty > 1\text{ year} \Rightarrow Manufacturer = X$". Note that we are not arguing that the agent should ask the user to change specified preferences. Our position is that by providing such additional information, the agent can inform the user about possible consequences of his/her choices.

The user can then choose to relax or not relax his/her preferences in the light of this new information. In addition, if the user wants the results to be ranked by some attribute, e.g., price, the agent can do "what if" type queries and suggest the reduction in price the user can get by relaxing one or more of the stated preferences. Such *differential analysis* will allow the user to best restate the query if he/she decided to relax some of his/her constraints. This analysis can significantly improve consumer understanding and awareness of the marketplace.

Suggesting alternate products : User B was interested in buying a portable CD player as a present to a friend. This friend is an avid walker/jogger and B thought that a portable music player would make a great present for the friend. On hearing B's gift idea, a common friend suggested that in place of the portable CD player, B should consider an MP3 player, a lightweight digital music player that can store hours of digital-quality music. This product was recommended because it was easier to carry, has no moving parts and never skips. These features are particularly useful for listeners engaged in physical activities like jogging. Even though this product costs more, B found it to be more appropriate because of its features. The friend was able to suggest the alternate product because of an understanding of B's goal. It may be possible to automate such user goal recognition for specific scenarios. Automating this process in general without significant user guidance is probably infeasible with current technology.

One can list a number of such scenarios where the consumer's initial choice or preference can be modified in the light of new information. The assumption that the average consumer has all the latest information at his/her fingertips is unfounded. On the contrary, rapidly changing market conditions imply that it is next to impossible for the average consumer to keep track of the latest options, deals, package offerings, etc., all of which can influence his/her final choice of what he/she is going to buy and at what price. We proposed a capable buyer agent, SAATHI, that can keep track of changing market conditions and make the user aware about the possibilities and opportunities in the market [1] . This buyer agent empowers the user to make more informed and hence more confident decisions to improve satisfaction.

Our proposed agent, *SAATHI*, provides services orthogonal to those provided by comparison shopping agents in that the emphasis is less on retrieving products matching user descriptions and more on informing the user about the interactions between constraints in that description and the marketplace. For example, in addition to returning products matching the constraints in the user query, *SAATHI* can present related products and/or products that can be obtained by relaxing some of the constraints in the user query. This draws

[1] In this paper, we use the term market synonymously with the information environment of the agent.

the attention of the user to the possibilities which could have been unexplored otherwise and thus broaden scope of more satisfying or favorable selections.

We have developed *SAATHI* as an instance of a buyer's agent for the apartment locator domain. It analyzes constraints in user queries to infer implied constraints based on current market condition [24]. Later in Section 3, we describe its current functionalities and other functionalities that we are currently working to add to the system.

The rest of the paper is organized as follows: Section 2 presents techniques and systems related to that discussed here; Section 3 formalizes the concept of static, dynamic and implied constraints in user queries; Section 4 presents the architecture of *SAATHI*; Section 5 introduces the example domain used to illustrate our system; Section 6 discusses implementation issues of *SAATHI* and presents sample interactions to illustrate its functionalities; and Section 7 summarizes the discussion and identifies possible extensions to the system proposed.

2 Related work

In the last few years the volume of information available to a common user on the Internet has increased exponentially. But it is very difficult for the average user to find all relevant and useful information given the overwhelming amount and diversity of information sources, content, presentation format, etc. There is significant research in electronic commerce to empower the user to take right decisions. Intelligent automated agents are deployed to process information on behalf of the user. The burgeoning interest in agent-based systems and its applicability has produced a number of techniques for intelligent information retrieval.

One application of agent based systems is to increase the quality of information retrieved from the Internet. Typical search engines return too many links in response to user queries, most of which are irrelevant to the user. Web-page recommender systems try to alleviate some of this problem by finding pages of interest to a user based on his/her query and usage patterns [18,19]. Another approach used to facilitate user access to pertinent information is to trace the user profile at a Web store and customize the pages according to that profile [25]. In this approach the description of the store catalog is adapted to reflect the preferences of the users. Hyper-textual pages are dynamically generated by applying personalization rules to user models based on activity profiles.

A recommender system is usually used by the E-commerce sites to lure the

user with other offers that he/she may be interested in [26]. Recommender systems can be either content-based or collaborative filtering based. In the content-based approach, users are categorized by their query and usage patterns and web pages with contents similar to those visited by the user are recommended by the system [18,19]. In the collaborative filtering approach, the system recommends based on similarity among users. The system categorizes the user and suggests similar options that previous users belonging to the same category opted for [27,28]. The Yoda system combines collaborative filtering and content-based querying for providing product recommendations [29]. Fast online recommendation is obtained by offline training using genetic algorithms on data from, among other things, user navigation patterns. The Helpful Online Purchase Environment (HOPE) system uses data mining to generate suggestions predicated both on the user and the content of the web page [30]. HOPE uses tag fields with products to organize them into a hierarchical structure and uses a nearest neighbor algorithm to find related products based on the customer's purchase history. A different approach is suggested by Wei *et. al.* [31,32]. They have proposed a market based recommender system to incorporate multiple heterogeneous recommendation approaches. Here, different agents may have different recommendations and they compete in a market-based protocol to propose their recommendation to the user.

A different approach is that of "opportunistic exploration" [20]. This concept is based on the premise that users have multiple and overlapping interests, exposure to appropriate items can appeal to a user's latent interests, active interests may be subdued if users are not exposed to their items of interest soon enough, and users prefer simple navigation of online stores to search for their items. The purpose of the system is to influence the short and long term interests of users shopping at an online store by exposing them to a dynamic environment of items that are updated through interaction and to assess his/her interests.

3 User constraints and market conditions

We now present a formal model to represent constraints explicitly specified in a user query and the nature of results returned by querying the current market. The basic assumption in our model is that the environment can be modeled by a relational data model [33]. We believe that the relational model is sufficiently general to effectively model the data requirements of most application domains of interest to us. In the following discussion, we will use the term 'database' for any collection of data, irrespective of whether it is stored locally on the user site or distributed over several sites on the network or the Internet. We will not concern ourselves with the implementation of the relational model (and we

have no control over how the data will be implemented in databases across the network), and hence use a universal relation schema $R(A_1, A_2, \ldots, A_n)$, where R is the name of the domain, e.g., Resorts, and each A_i correspond to an attribute in the domain, e.g., name, category, location. A particular element of the relation, e.g., a given resort, is denoted by an n-tuple $t = \langle v_1, v_2, \ldots, v_n \rangle$, where $\forall i, v_i \in \text{Domain}(A_i)$. For example, a given resort is completely described by a vector of values corresponding to each of the resort attributes.

A typical user query consists of a conjunction of a number of logical tests, where each logical test is a constraint on the range of values a particular attribute can take. Such a query can be formally specified as follows:

$$(A_1\, cond_1\, a_1) \wedge \ldots \wedge (A_n\, cond_n\, a_n),$$

where $\forall i, cond_i$ is a logical condition in the set $\{=, \neq, <, >, \leq, \geq, \mathbf{In}, \#\}$. For the In condition, there are two possibilities:

- If A_i is an ordinal attribute, the In operator tests for values in a contiguous range, and a_i stands for a range $[v_{low}, v_{high}]$, where $v_{low} < v_{high}$ and $v_{low}, v_{high} \in \text{Domain}(A_i)$,
- If A_i is a nominal attribute, the In operator tests for membership of values in a given set and a_i stands for a set of values $\{v_{i1} \ldots v_{ik}\}$ where $\forall j,\ v_{ij} \in \text{Domain}(A_i)$.

For other operators, $a_i \in \text{Domain}(A_i)$. The # condition is a *don't care* operation and will match any value. Other symbols have their standard interpretation. As each logical test except the *don't care* condition limits the possible set of values for an attribute for a match, we view each such test as a constraint specified by the user: $C_i \equiv (A_i\, cond_i\, a_i)$ where each condition is limited to the set $\{=, \neq, <, >, \leq, \geq, In\}$. Since the conjunction between the tests are implied and *don't care* conditions do not affect query results, for brevity we will represent a user query by only the set of constraints: $Q = \{C_i\}$, i.e., we will drop the tests containing the # condition. For example, the query from a user searching for a ski resort in either Colorado or Alberta for 3-5 days at an average room rate of less than \$200 can be represented as $\{(Type =$ 'ski'$)$, $(Location\, In\, \{\text{Colorado, Alberta}\})$, $(Duration\, In\, [3, 5])$, $(RoomRate < 200)\}$.

In response to a user query, all items in the database that satisfies each of the constraints specified in the query should be returned. It is well-recognized that the attributes in a domain are not mutually independent. Hence constraints imposed on a given set of attributes might constrain the values of other attributes in the results of a query. We are particularly interested in such "implied constraints" as an average user may not be aware of these additional constraints implied by the constraints (s)he explicitly specified in a query. We

believe that identification of such implications will improve the user's awareness and understanding of the market and can allow the user to reformulate the original query to better reflect his/her interests and preferences.

3.1 Static and dynamic constraints

In addition to domain, key, and integrity constraints, each relational database also satisfies a set of *functional dependencies* (FDs). The latter is of particular interest to us as it specifies relationships between different attributes that must always hold. For example, in the resort domain a typical FD would be $Location \rightarrow Tax\_Rate$ which implies that any two resorts located in the same location will also have the same tax rate. The FDs in a given domain correspond to *static constraints* that hold for the lifetime of the database. We do not expect that an average user to be aware of most of such static constraints in the domain, but can benefit from such knowledge. Additionally, the user can significantly benefit from the knowledge of other, possibly temporary, relationships currently existing in the database. For example, the relationship represented by the rule "Location=$Bahamas \Rightarrow$ Cancellation-charge = No" may hold at a given point in time, though it may not have been true in the past or may not be true in the future. Such *dynamic constraints* (DCs) differ from FDs in two major ways:

(1) DCs are of the form $(A_i\, cond_i\, a_i) \wedge (A_j\, cond_j\, a_j) \wedge \ldots \wedge (A_x\, cond_x\, a_x) \Rightarrow (A_z\, cond_z\, a_z)$, whereas FDs are of the form $A_iA_j \ldots A_x \rightarrow A_z$. A set of FDs are defined on a given database schema, $\mathcal{S}$. A given FD specifies that if the attributes from $\mathcal{S}$ on the LHS of the FD have the same values for two database tuples (items), the attribute on the RHS of that FD must have the same value in those tuples. DCs are implication rules that constrains attributes on the RHS of the rule in items which match the LHS of the rule. The conditions in the DCs are more general than the implied equality conditions in FDs. Whereas DCs constrain the attribute values in a given item, FDs constrain attribute values in item sets.

(2) Any legal database state or instance, $I_{\mathcal{S}}$, must satisfy all FDs defined on the associated database schema, $\mathcal{S}$. Whether a DC, $\mathcal{D}$, holds or not on a given database, however, depends on the current database instance, changing over time with insertion and deletion of records. Hence, these constraints are dynamic.

Since the relationships captured by conjunctive rules can aid the user's understanding of market dynamics, we propose that *SAATHI* will infer such rules periodically by analyzing market data. Note that though such inference from data is not justified for inferring FDs because current relationships do not imply permanent relationships, such inference do serve our purpose of keeping

abreast with market conditions.

3.2 Implied constraints

The relevant technical question then is how to infer relationships of the above type by inspecting the current instance of a relation. There exists several data mining approaches for inferring relationships in data. Our goal is to present these relationships to the user in the form of easy to comprehend rules and hence we did not consider mechanisms like neural or Bayesian nets. For ease of use, we use only rules corresponding to Horn clauses, i.e., rules of the form $(A_i\, cond_i\, a_i) \wedge (A_j\, cond_j\, a_j) \wedge \ldots \wedge (A_x\, cond_x\, a_x) \Rightarrow (A_z\, cond_z\, a_z)$ where the LHS can be written as a conjunction of constraints and RHS consists of a single constraint. Each such rule r, can then be represented by a pair (L_r, C_r), where L_r is the set of constraints on the LHS and C_r is a single constraint on the RHS.

Several rule learning mechanisms have been developed in the machine learning literature that can possibly be used to infer such rules [34–36]. We believe that propositional rule learners will be sufficient for most domains, and hence did not use systems that can learn first order rules, e.g., FOIL [36]. In our implementation, we used the rule generation facility associated with C4.5, perhaps the most well-known decision tree learning system [37].

Let $R_{I_\mathcal{S}}$ be the set of rules learned given a database instance $I_\mathcal{S}$. If computational time and costs were negligible, such rule learning procedures could be run in response to each user query. Given finite computational costs, however, rules should be learned periodically with the frequency chosen so as to always have up-to-date learned rules. When a user poses a query, $Q = \{C_i\}$, the constraints in the query can be used to match the antecedents of the latest learned rules. The consequents of the set of the matched rules constitute implied constraints given the user query and the current relationships in the market database as captured by the learned rules. Note that a rule, r is matched if all the constraints on the LHS, L_r is contained in a query. The set of implied constraints for a query Q given the learned results $R_{I_\mathcal{S}}$ can then be written as following [2]:

$$\tilde{C}_{Q,R_{I_\mathcal{S}}} = \{C_r | r \in R_{I_\mathcal{S}} \wedge L_r \subseteq Q\}.$$

[2] We would like the implied constraints to be identical to the set of dynamic constraints D holding on the database instance at the time of the query. In practice though, limitations of inductive rule learning schemes, pre-processing of the database instances for learning schemes, time difference between rule updates and query, etc. mean that the implied constraints are approximations of the actual dynamic constraints.

For example, if we reconsider the lawn mower example used in the "Introduction" section, when the user asks for 2-year warranties, this can generate the implied constraint that the manufacturer of all such products is X. This information may make the user relax some of the constraints in the original query. Our position is that being made aware of these additional implied constraints, which the user might not have been aware of, the user can make more informed decisions about purchasing goods and services of interest.

3.3 Constraint relaxation for optimization

Another aspect of our work involves providing differential analysis of the user query to present the user with an analysis of how each individual constraint in the query influences the set of results returned. We presume that in general the user's goal is to optimize a given criterion, e.g., cost of a vacation package. The goal here is to help the user optimize that decision criterion, e.g., maximize area or minimize rent in the apartment locator domain. If the user chooses any one of these criteria, a differential analysis will identify the constraint or constraints in the original user query to be altered/relaxed to maximally improve the quality of the query results. This facility allows the user to immediately understand different "what-if" scenarios which can allow the user to identify constraints that would be most beneficial to relax. For example, the user may find it preferable to relax the constraint of covered parking to get a 10% reduction in rent.

To perform such differential analysis we use the set of constraints in the user query. We create a set of queries by dropping exactly one of the constraints in the set at a time. As above let $Q = \{C_i\}$ be the user query. Let $Q_{\bar{C}}$ be the query where the constraint C has been dropped from the query Q, i.e., $Q_{\bar{C}} = Q \backslash \{C\}$. Then the set of new queries to perform differential analysis, D_Q is given by: $D_Q = \cup_{C \in Q} Q_{\bar{C}}$. We also ask the user to provide a criterion to optimize. Typically this is the value of one of the attributes of the domain, say A. Without loss of generality, we assume that the user wants to minimize the value of A in the items matching the query.

Given a database instance $I_{\mathcal{S}}$, let $M^q_{I_{\mathcal{S}}}$ be the set of matches returned for the query q, and $t^{q,A}_{I_{\mathcal{S}}} \in M^q_{I_{\mathcal{S}}}$ be the tuple or item in the result with the minimal value for A, i.e., $t^{q,A}_{I_{\mathcal{S}}} = \arg\min_A \{t(A) | t \in M^q_{I_{\mathcal{S}}}\}$, where $t(A)$ represents the value of attribute A in tuple t. We denote by $v^{q,A}_{I_{\mathcal{S}}} = t^{q,A}_{I_{\mathcal{S}}}(A)$. the corresponding minimal value. For the sake of brevity, we will drop the $I_{\mathcal{S}}$ subscript where the database instance is known from context. We note that for any constraint $C \in Q$, as $Q_{\bar{C}} \subset Q$, $M^{Q,A} \subseteq M^{Q_{\bar{C}},A}$, i.e., the same or more matches are returned with a relaxed query, and $v^{Q,A} \geq v^{Q_{\bar{C}},A}$, i.e., an equal or a smaller (better) value for the minimizing criterion is returned with the relaxed query.

The lowest value for the criterion specified A, given the user query Q and the current database instance, is then $v^{Q,A}$ for the item $t^{Q,A}$. Now, we query the database with each of the relaxed queries in D_Q to obtain the sets $T^{Q,A} = \{t^{q,A} | q \in D_Q\}$, the set of items with the minimum values for A for each of the relaxed queries, and $V^{Q,A} = \{v^{q,A} | q \in D_Q\}$, the set of the corresponding minimum values. In response to query $Q = \{C_i\}$ and the optimization criterion A, we return both the pair $(t^{Q,A}, v^{Q,A})$ and a set of pairs $\{(t^{Q_{\bar{C}},A}, v^{Q_{\bar{C}},A}) | C \in Q\}$. The user can then choose to relax any constraint $C \in Q$ and obtain a reduction of $v^{Q,A} - v^{Q_{\bar{C}},A}$ in the value of attribute A by choosing the item $t^{Q_{\bar{C}},A}$, returned with the relaxed query after dropping constraint C, rather than the item $t^{Q,A}$ returned with the original query.

Let $\underline{v^A} = \min_{q \in D_Q}\{v^{q,A}\}$ be the minimum of the values that can be obtained from tuples matching queries with relaxed constraints. $Q_{\underline{v^A}} = \arg\min_{q \in D_Q}\{v^{q,A}\}$ is the corresponding query and the constraint to be relaxed is $C_m = Q - Q_{\underline{v^A}}$. The user may or may not choose to relax this or any other constraint. This analysis, however, does present the user with a detailed picture of the effect of each constraint in the original query on the optimizing criterion. We believe such analysis will enable the user to take more informed decisions and improve his/her utility.

Note that the above analysis considers only queries obtained by dropping exactly one constraint from the original query at a time. It is of course possible to perform a more extensive, and computationally expensive, differential analysis by dropping all possible subsets of the constraints in turn, i.e, use $D_Q = \cup_{R \subset Q}(Q - R)$. We believe that such exhaustive analysis might be an overkill and can even overwhelm the user with too much information for realistic domains with more than a handful of attributes. Hence, though such an elaborate analysis can capture a more detailed snapshot of the domain dynamics, we have not pursued this option.

4 System architecture of *SAATHI*

We now present brief descriptions of the different modules in the architecture of *SAATHI*, our proposed buyer's agent (see Figure 1):

Interface: The interface allow the user to present a structured, constrained query in the application domain. This query is forwarded to the data retrieval engine and to the analysis module. Results returned from both these modules are displayed back to the user in an easy-to-browse format.

Data retrieval engine: The data retrieval engine's purpose is to query the market and gather data as required by the user or the other modules. The engine can be a database query engine in the situation where all the informa-

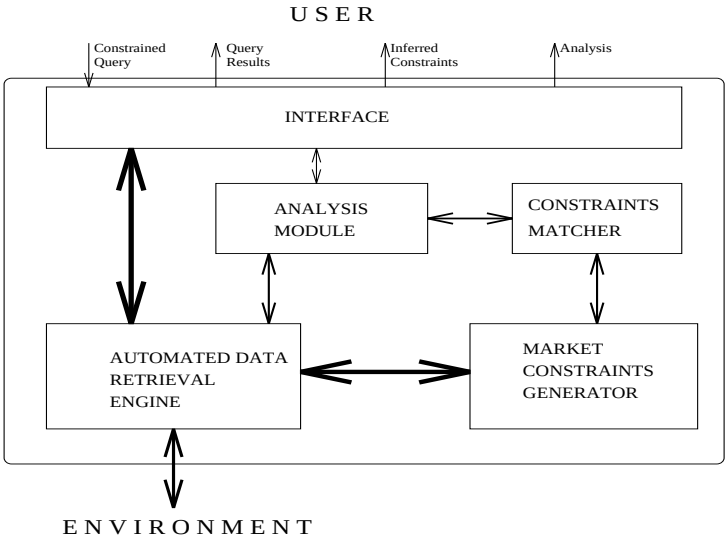

Fig. 1. The architecture of *SAATHI*.

tion resides in a local database. It can also be as complicated as an Internet based distributed information gathering mechanism when data resides at multiple remote sites.

Market constraint generator: The market constraint generator uses the query engine to retrieve a sizable or representative portion of current data in the market. This retrieved data is then mined to unearth any significant existing relationships. Whereas in some domains a simple dump of the relevant relations may be sufficient to gather needed data, in other domains sophisticated statistical sampling may have to be performed to gather representative data from non-local sources.

Constraints matcher: The constraints matcher contains a rule-matching mechanism that matches the constrained user queries with the inferred rules. The result is the generation of all rules that match the constraints of the user query. The consequents of these matched ruleset provides the implied constraints over and above the constraints present in the user query.

Analysis module: The analysis module of *SAATHI* calls the constraints matcher to generate the implied constraints from the user query. It can also perform other kinds of analysis, e.g., differential analysis by relaxing some of the constraints in the user's query. This will enable the module to make specific recommendations to the user about how best to relax constraints in the query to obtain satisfactory results. If provided with a rich domain ontology, the analysis module can also be used to map a user query into a more appropriate query to return closely related product/service information [38].

5 The apartment locator domain

To evaluate the feasibility and usefulness of the "buyer's agent" concept, we chose two disparate domains for initial exploration: a vacation resort selection domain, and an apartment locator domain. In this paper, we report on our experience with the latter of these two domains. The relational model for the apartment location domain has 10 attributes, e.g., Rent, NumBedrooms, DepositAmount, WD (Washer-Drier), etc. There exists e-commerce sites using which one can search for apartments in a particular city or region [39,40]. We visualize *SAATHI* to be an agent "higher up the food chain" [41] which can query such sources through its data retrieval engine.

6 System in use

In this section we will discuss our proposed system, *SAATHI*, working in the apartment locator domain in the city of Tulsa.

6.1 Implementation issues

For the implementation, we limited our domain to the city of Tulsa. The data about Tulsa apartments is collected by an automated data retrieval engine from several web sites [39,40]. The data collected in this phase is parsed and form an apartment database. The next step was to mine this database to generate the rules. We used the C4.5 system [37], a popular decision tree learning mechanism, to generate the rules that capture the existing relationships in the data. To do this, we ran the rule generator many times, once for each of the attributes in the domain. Each attribute was chosen once as the target attribute, and the rest of the attributes were used to predict the values for this attribute. We had to pre-process the raw data for this stage. In particular, when continuous attributes were used as the target attribute we had to discretize them as the rule generation procedure works with only discrete-valued target attributes. For example, when we used *Rent* as the target attribute, we defined several ranges into which the apartments were classified, e.g., 500–599 was defined as midrange (MR). When *Rent* was used as an attribute to classify another target attribute, however, the continuous values were used, e.g., 450 rather than MR.

The rules learned were not necessarily 100% accurate. We decided to use rules that even though not completely accurate, were accurate in a large percentage of cases they matched. We have used rules with accuracy above a threshold

LEARN-DOMAIN-RULES($Attributes, Examples, Threshold$)
$Rules \leftarrow \phi$
For each $A \in Attributes$
    $A^- \leftarrow Attributes \backslash \{A\}$
    $r_A \leftarrow$ LEARN-RULE-SET($A, A^-, Examples$)
    $r_A \leftarrow$ ELIMINATE-INACCURATE-RULES($r_A, Threshold$)
    $r_A \leftarrow$ DROP-NEGATIVE-CONDITIONS(r_A)
    $Rules \leftarrow Rules \cup r_A$
$Rules \leftarrow$ ELIMINATE-DUPLICATES($Rules$)

Fig. 2. Algorithm for generating all rules in a domain.

of 90%. This means that the implied constraints were not without exceptions, but the exceptions were small enough in number to warrant the presentation of this constraint. We observed that if the rule performance threshold was lowered below 80%, too many rules were being generated. These 'not very accurate' generalizations are more likely to confuse rather than aid the user.

Typical rules learned by the process described above include the following:

$$Rent < \$400 \Rightarrow CoveredParking = N,$$

which means apartments with rent less than $400 do not have covered parking;

$$Bedroom = 2 \wedge Area > 1000sqft \wedge CoveredParking = Y \Rightarrow Rent > \$700,$$

which means apartments with 2 bed rooms, area more than 1000 sq ft and covered parking has rent more than $700.

The algorithm for rule generation is given in Figure 2. The LEARN-RULE-SET function can be any rule learning algorithm; we have used C4.5 in our implementation.

Note that rules are not necessarily causal. The learning process did unearth a lot of patterns that we were not expecting. It is important to recognize that these patterns are necessarily impermanent and hence it may not be useful to search for any fundamental long-lasting correlation between the antecedents and consequents of the learned rules.

A post-processing step that we decided to invoke, DROP-NEGATIVE-CONDITIONS, removed negative conditions from the rule. The reason for removing negative conditions was that for most features it is unlikely that the user will specify a negative constraint; e.g., we do not envisage user's not wanting covered parking space. Alternatively we could have defined the matching process such that a query with no constraints for an attribute will also match a rule which

has a negative constraint. In the following we present a rule before and after post-processing:
Rule before post-processing:

$$(Rent = \$600) \wedge (CoveredParking = N) \Rightarrow FitnessCenter = N,$$

Rule after post-processing:

$$(Rent = \$600) \Rightarrow FitnessCenter = N.$$

The rule with the relaxed constraint is also tested for accuracy and is substituted for the original rule only if its accuracy is above the chosen threshold. This constraint-dropping process generated two types of redundancies in the resultant rule sets: identical rules, and rules that were more general than other rules. So far, we have eliminated only duplicate rules. Since specific rules are at times more accurate than their more general counterparts, we have not eliminated them from consideration. This redundancy will also be useful when the user can dynamically change the threshold for the accuracy of rules to be used in identifying implied constraints.

The resultant rule base that we obtained through the above-mentioned process has a little more than 100 rules. This number depends on the rule accuracy threshold that we have selected.

6.2 Differential analysis

The result screen of *SAATHI* in response to a user query is presented in Figure 3, where the three rightmost frames present results of the query. The query is shown in the first frame while the fourth frame shows the implied constraints. All the items that match the query are shown in the second frame. In the third frame we see the optimal match for dropping each of the constraints. The corresponding option can be viewed by clicking on the constraint that is relaxed. For example consider the case where the user wants to obtain minimum rent and provides constraints of *rent < $500, area > 600 sq. ft., 1 bedroom, deposit < $150 and requires washer-dryer*. In additional to the full query, *SAATHI* also finds the response to additional queries dropping each of the constraints. That is, for each of the constraints in the original query, we now know what is the minimum rent that can be obtained if we drop that constraint. Finally the set of constraints in the original query is sorted in ascending order by these values, which gives a list of constraints the user should consider relaxing in order if he/she wants to get a lower rent than what was returned with the original query. We see that the original query returned a

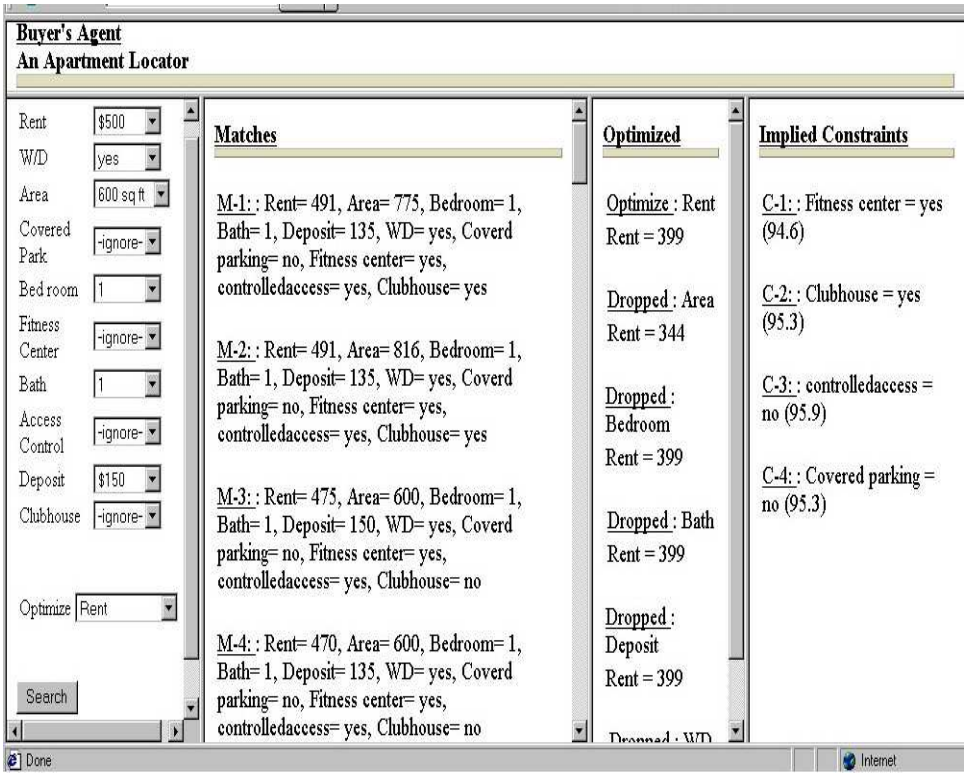

Fig. 3. The result screen of *SAATHI*.

minimum rent of $399, but dropping the area constraint can further lower the rent to $344. The user can then decide whether the extra space is worth the $54/month in rent.

The results returned also include a list of apartments satisfying the original query, and another list containing the set of implied constraints given the prevalent market conditions and the user constraints. One implied constraint for this particular query is that covered parking is not available. The user can click on the constraint to see which of the constraints in the original query implies this constraint, and based on that information can further modify the original query.

7 Conclusions

In this paper, we argued for the usefulness of a "Buyer's agent" which will enable an average user to make more informed choices while choosing products or services from electronic commerce sites. We posit that the average consumer finds it difficult to keep tabs on market conditions, and can look for features in a product he/she is interested in purchasing that can restrict the set of choices, increase price, etc. We proposed a prototype buyer's agent, *SAATHI*,

whose goal is to keep abreast with the prevailing market condition and to inform the user of implied constraints, relative effects of different constraints, alternative products, etc.

We identified the scope of such a buyer's agent, proposed an architecture to deliver the suggested functionality, described the implementation in an apartment location domain, and presented sample interactions with the system. Such a system can be augmented to add persistent queries, in which case, the user can be informed of particular opportunities that he/she had previously queried but have become available at a later time. Another useful modification would be to present "result densities" as part of the differential analysis process, where the distribution of the matches for the optimizing criteria is presented for different query relaxations. Such distributions will provide the user a clearer view of how the different constraints in the query are influencing the set of items matched.

The vision of a buyer's agent is a powerful one: a knowledgeable well-wisher helping the user to select the product or service that is most satisfying for the user. We have presented our initial steps towards that vision.

Acknowledgments: This work has been supported in part by an NSF award IIS-0209208.

References

[1] Amazon.com, uRL: http://www.amazon.com/.

[2] ebay, uRL: http://www.ebay.com/.

[3] Onsale, uRL: http://www.onsale.com/.

[4] H. S. Nwana, J. Rosenschein, T. Sandholm, C. Sierra, P. Maes, R. Guttman, Agent-mediated electronic commerce: Issues, challenges and some viewpoints, in: Proceedings of the Second International Conference on Autonomous Agents, ACM Press, New York: NY, 1998, pp. 189–196.

[5] T. Sandholm, emediator: A next generation electronic commerce server, Computational Intelligence 18 (4) (2002) 656–676.

[6] P. Maes, R. H. Guttman, A. G. Moukas, Agents that buy and sell, Communications of the ACM 42 (3).

[7] emediator, uRL: http://ecommerce.cs.wustl.edu/emediator/.

[8] Kasbah, uRL: http://www.kasbah.media.mit.edu/.

[9] N. Jennings, K. Sycara, M. Wooldridge, A roadmap of agent research and development, International Journal of Autonomous Agents and Multi-Agent Systems 1 (1) (1998) 7–38.

[10] P. Maes, Agents that reduce work and information overload, Communications of the ACM 37 (7) (1994) 31–40.

[11] T. Mitchell, R. Caruana, D. Freitag, J. McDermott, D. Zabowski, Experience with a learning personal assistant, Communications of the ACM 37 (7) (1994) 80–91.

[12] K. Lang, Newsweeder: Learning to filter netnews, in: Proceedings of the Twelfth International Conference on Machine Learning, Morgan Kaufmann, San Francisco, CA, 1995, pp. 331–339.

[13] O. Etzioni, D. Weld, Softbot-based interface to the internet, Communications of the ACM 37 (7) (1994) 72–79.

[14] J. Delgado, N. Ishii, Multiagent learning in recommender systems for information filtering o n the internet, International Journal of Cooperative Information Systems 10 (1) (2001) 81–100.

[15] J. Schafer, J.Konstan, J.Riedl, Electronic commerce recommender applications, Journal of Data Mining and Knowledge Discovery 5 (2001) 115–152.

[16] M. Ackerman, D. Billsus, S. Gaffney, S. Hettich, G. Khoo D. Kim, R. Klefstad, C. Lowe, A. Ludeman, J. Muramatsu, K. . Omori, M. Pazzani, D. Semler, B. Starr, P. Yap, Learning probabilistic user profiles: Applications to finding interesting web sites, notifying users of relevant changes to web pages, and locating grant opportunities, AI Magazine 18 (2) (1997) 47–56.

[17] M. Koutri, S. Daskalaki, N. Avouris, Adaptive interaction with web sites: an overview of methods and tech niques, in: Proceeding of the 4th International Workshop on Computer Science a nd Information Technologies, 2002.

[18] M. Pazzani, J. Muramatsu, D. Billsus, Syskill & Webert: Identifying interesting web sites, in: Proc. of the 13th National Conference on Artificial Intelligence, MIT Press/AAAI Press, 1996, pp. 74–79.

[19] J. Rucker, M. J. Polanco, Personalized navigation for the web, Communications of the ACM 40 (3).

[20] D. Bryan, A. Gershman, Opportunistic exploration of large consumer product spaces, in: Proceedings of the First ACM Conference on Electronic Commerce, ACM Press, New York: NY, 1999, pp. 41–47.

[21] J. S. Breeze, D. Heckerman, C. Kadie, Empirical analysis of predictive algorithms for collaborative filtering, in: Proceedings of Fourteenth Conference on Uncertainty in Artificial Intelligence, Morgan Kaufmann Publishers, San Francisco, CA, 1998, pp. 43–52.

[22] H. Kautz, B. Selman, M. Shah, Combining social networkds and collaborative filtering, Communications of the ACM 40 (3).

[23] Logical decisions, uRL: http://www.logicaldecisions.com/.

[24] S. Sen, K. Hernandez, A Buyer's Agent, in: Proceedings of the Fourth International Conference on Autonomous Agents, ACM Press, New York, NY, 2000, pp. 156–162.

[25] L. Ardissono, A. Goy, Tailoring the interaction with users in electronic shops, in: Proceedings of the 7th International Conference on User Modeling, 1999, pp. 35–44.

[26] J. B. Schafer, J. Konstan, J. Riedl, Recommender systems in e-commerce, in: Proceedings of the First ACM Conference on Electronic Commerce, ACM Press, New York: NY, 1999, pp. 158–166.

[27] J. L. Herlocker, J. A. Konstan, L. G. Terveen, J. T. Riedl, Evaluating collaborative filtering recommender systems, ACM Trans. Inf. Syst. 22 (1) (2004) 5–53.

[28] U. Shardanand, P. Maes, Social information filtering: algorithms for automating word of "mouth", in: CHI '95: Proceedings of the SIGCHI conference on Human factors in computing systems, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1995, pp. 210–217.

[29] C. Shahabi, Y.-S. Chen, An adaptive recommendation system without explicit acquisition of user relevance feedback, Distributed and Parallel Databases 14 (2) (2003) 173–192.

[30] T. Sun, A. Trudel, An implemented e-commerce shopping system which makes personal recommendations, in: Internet and Multimedia Systems and Applications (IMSA 2002), IASTED/ACTA Press, Calgary, Canada, 2002, pp. 58–62.

[31] Y. Z. Wei, L. Moreau, N. R. Jennings, Recommender systems: a market-based design, in: AAMAS '03: Proceedings of the second international joint conference on Autonomous agents and multiagent systems, ACM Press, New York, NY, USA, 2003, pp. 600–607.

[32] Y. Z. Wei, L. Moreau, N. R. Jennings, A market-based approach to recommender systems, ACM Transactions on Information Systems 23 (3).

[33] R. Elmasri, S. B. Navathe, Fundamentals of Database Systems, Addison-Wesley, Reading, MA, 2000.

[34] P. Clark, R. Niblett, The CN2 induction algorithm, Machine Learning 3 (1989) 261–284.

[35] R. Michalski, I. Mozetic, J. Hong, H. Lavrac, The multi-purpose incremental learning system AQ15 and its testing application to three medical domain, in: Proceedings of the Fifth National Conference on Artificial Intelligence, Morgan Kaufmann, Philadelphia, PA, 1986, pp. 1041–1045.

[36] R. J. Quinlan, Learning logical definitions from relations, Machine Learning 5 (3) (1990) 239–266.

[37] R. J. Quinlan, C4.5: Programs for Machine Learning, Morgan Kaufmann, San Mateo, California, 1993.

[38] S. Sen, P. S. Dutta, S. Debnath, CHAYANI: A shopper's assistant, Journal of Intelligent Systems 14 (1) (2005) 3–23.

[39] Apartments.com, uRL: http://www.apartments.com/.

[40] Rent, uRL: http://www.rent.net/.

[41] O. Etzioni, Moving up the information food chain: Deploying softbots on the world wide web, AI Magazine 18 (2) (1997) 11–18.