Do What I Mean: Online Shopping with a Natural Language Search Agent

Barry G. Silverman[1], Mintu Bachann[2], Khaled Al-Akharas[2]
1- Dept. of Systems Engineering, University of Pennsylvania, 2- Equalfooting.com
barryg@seas.upenn.edu
June 2001

Key Words: Natural Language, Search Agent, On-line Shopping, E-Commerce

ABSTRACT

A major obstacle to online shopping is the lack of meaning understanding and precision in e-catalog searching engines. These are problems that Natural Language Query (NLQ) should be able to solve, but historically, has been unable to due to difficulties of scaling to the complexities and performance demands of large-scale online catalogs. Recent research, however, has now enabled NLQ search to hold its own in terms of timing while it improves precision and parsing capability.

1) Introduction

This article describes a bot or agent that uses restricted natural language to help users search product catalogs in large scale market exchanges. Market exchanges are sites that offer a virtual integration across multiple vendors' catalogs so that users can browse, search, bid, finance, buy, and/or ship products. Market exchanges are a major form of e-commerce in both the business to consumer and business to business segments: e.g., see Amazon.com or Ariba.com among many others.

O-ver half of today's 80 million web users shop for or buy products online, and business to business purchasing is expected to rapidly eclipse that level [1]. However, in the rush to provide online presence, many e-market sites have been built quickly, with little infrastructure and capabilities needed to run such an e-business. For example, [1] reports that over 80% of web shoppers have at some point left e-markets without finding what they want and that 23% of all attempted e-shopping transactions end in failure. Four of the top five failure modes are search-related (i.e., page loading times, couldn't find product, system crashed, had to call customer service).

Research also shows that about 15% of online search failures are due to spelling errors and another 40% are due to customers using different terms from those in the website (e.g., patching vs. concrete): e.g., see [1,2]. Because they can't interpret the meaning of the users' query, keyword-based search engines typically are tuned to bring back innumerable hits of everything even remotely relevant, often burying the best choice deep within the list (or omitting it altogether) and providing little help for then searching just the returned set of items. Two approaches exist for improving keyword-based search, conceptual query (CQ) and natural language query (NLQ), and we will shortly describe these (see sidebar), but first let us examine the challenges that catalogs pose for any search method.

SIDEBAR: Conceptual vs. Natural Language Query
All query languages attempt to match the query string to database entries. Interested readers should consult [8, 9, 16-18], but for now we compare but two of the many approaches: **Conceptual Query** – Literally this refers to an expansion of the search terms with the help of a concept tree of closely related terms, however, in practice this also includes any other expansion such as via a thesaurus to generate synonyms. Generally prior to the expansion, the search string is tokenized, stop words are removed, and the tokens is then passed to traditional SQL for matching against the database entries. **Natural Language Query** – In addition to all the steps of conceptual query, this also includes a parser that attempts to identify and label the part of speech of each token in the search string. Such labels can assist in (1) reducing the expansion set to those with the same sense as the original and (2) helping the traditional SQL in narrowing its search.

Part of the challenge that searching the content of shopping catalogs faces is the same challenge that the rest of the web faces – content that was created for human consumption, poses a challenge for machine interpretation or for use by other agents. To address this challenge, a significant military (DARPA) and W3C "semantic web" effort is underway to tag the semantics of the web so that content created by laypersons for use by other people is automatically and transparently marked up for machine and agent interpretation and use (using tools such as the W3C's Resource Description Format or RDF, DARPA agent markup language or DAML, and ontology interface language or OIL, among others) [12, 13]. While we did not use this toolset in the work reported here, having it available and scaled up for industrial use would potentially turn catalogs into part of the fabric of the "semantic web" and would potentially ease the content interpretation of market catalogs such as in the case study mentioned here. One needs to test this idea to be sure. Also, the semantic web efforts just mentioned do not include research on natural language querying, so it would also be interesting and challenging to see to what extent natural language agents such as described in this paper can be extended to work in the broader domain of the semantic web. In general, agents such as described here are intended for narrow domains only, so the challenge would be to populate the web with many such, special purpose agents.

This article is about shopping sites, however, so we turn our attention now to challenges that uniquely arise in that domain. One challenge is the question of the incompleteness and inconsistency of the product descriptions across sellers. Some descriptions include partial parameter information (e.g., 2", black) while others focus on how the product might be used, and still others are highly terse and omit most details. A second challenge is the difficulty of matching the buyer's search terms to the wording in the descriptive fields. Where exact term matches don't exist, one must consider issues related to word stemming, spelling errors, abbreviations, synonyms, and related problems. A third challenge is that numeric attributes are poorly and incompletely represented in these descriptions, yet some users will want to search by size, weight, height, etc. A universal feature of catalogs is that there are many products, the product lines are continually changing, and each of the 1,000s of categories of products has a different set of attributes. As a result, attributes are not stored as fields of a table. Rather, the attribute names are stored as data items as are their value settings, a fact that makes search more difficult.

Shopping catalogs often include many dozens of tables (some have 1,000s) in order to describe the products being offered and to support all the services and user needs. In order to improve runtime performance, a denormalized field called a "munge" is often used as the target for the search engine. This munge places into its sub-fields copies of each of the searchable fields of the catalog such as item name, item ID, category name and ID, model or part number, description, price, maker, condition, and all attribute-value pairs. In effect the munge is like a document on each product in the catalog. Since most search algorithms can't infer the sub-fields to search, they will search the entire munge. For example, keyword search will search across all sub-fields of the full munge for a strict match on terms, while CQ will traverse the same ground but with the ability to look for synonyms, alphabetically similar terms, and related conceptualizations. NLQ, in turn, is the only search strategy that infers the labels or field names of each token in the query string and, hence, can then send the (conceptual) search to the precise "subfields" of the catalog in which the query's tokens should exist provided they are in the database.

The reason that CQ does worse than NLQ is that it must deal with term semantics and ambiguities in the absence of a parser. NLQ, on the other hand, can parse the entire query string, label its tokens, and may hold an interactive conversation about the query to confirm its interpretation. NLQ researchers argue that CQ languages can be improved by adding a parser with a grammar restricted to the domain of the database. Such parsers have less-than-general interpretation power, but still can offer much needed trouble management improvements and in helping the interface conform to the user's language. Several in-the-lab systems purport to provide these extensions such as [3, 9, 14, 15], and some of them offer formal slotted grammars (with semantically typed slots) for merging the natural language approach atop conceptual querying.

Proponents of pure CQ in turn argue that the natural language extensions tend to be impractical, and indeed none of the NLQ systems just cited have been evaluated on a large scale. They argue that the "natural language problem" is too difficult and remains unsolved on any reasonable scale. As a result, most large scale relational database management products (eg, Oracle, Sybase, DB II, or Alta Vista Search Engine) that are widely used by market exchanges and other large scale e-commerce sites, include CQ features for those who choose to deploy them, but exclude NLQ. There are some smaller (less formally defined) NLQ systems that appear to work atop specific databases or environments: eg [4,5]. Also, there is a push to add natural language self-help or chatterbots to many e-commerce websites, but these bots handle site navigation and document retrieval issues, and they are entirely incapable of processing catalog or database search requests, a fact that adds fuel to the arguments that only conceptual query is scalable: e.g., see [9]. In sum, there are no examples of NLQ working in large scale e-commerce catalog shopping sites, and one is tempted to believe the proponents of CQ rather than NLQ.

2) <u>System Architecture and Algorithm</u>

This section provides an overview of how the agent performs NLQ. We make the assumption that a Markov decision process is suitable to analyze the semantics and morphosyntactics of the user's query. Specifically, at any moment in time, the agent is in one of a finite number of states (s=1,S) and must choose one of a finite set of actions (a=1,A) to transition to the next state. More specifically, optimizing a Markov decision process is a dynamic programming problem that maximizes E(U), the expected discounted rewards across future periods as follows:

$$\text{Max } V^* = E\left[\sum_{t=1}^{T} U(s_t, a_t)\right] \qquad (1)$$

where,
V* = optimum value point (in terms of precision and recall)
U( ) = reward function or utility from selecting action a_t at state s_t

Equation (1) may be solved if one loops across iterations (t=1,T) and for each iteration, one then loops across all states (s=1,S) to find the action or set of actions that maximizes both current and future rewards so as to avoid local optima. This expansion is captured by finding the maximal value of the following function after testing all possible actions, a=1,A:

$$Z_t(s,a) = U(s_t, a_t) + \delta \sum_{s_{t+1}=1}^{S} \pi(s_t, a_t, s_{t+1})\, V_{t-1}^{*}(s_{t+1}) \qquad (2)$$

where,

$\pi(s_t, a_t, s_{t+1})$ = the transition probability of being in state s_{t+1} immediately after taking action a_t from state s_t
$V_{t-1}^{*}(s_{t+1}) = Z_{t-1}(s_{t+1}, \text{argmax}_a(Z_{t-1}(s,a)))$ where argmax_a finds the maximal action

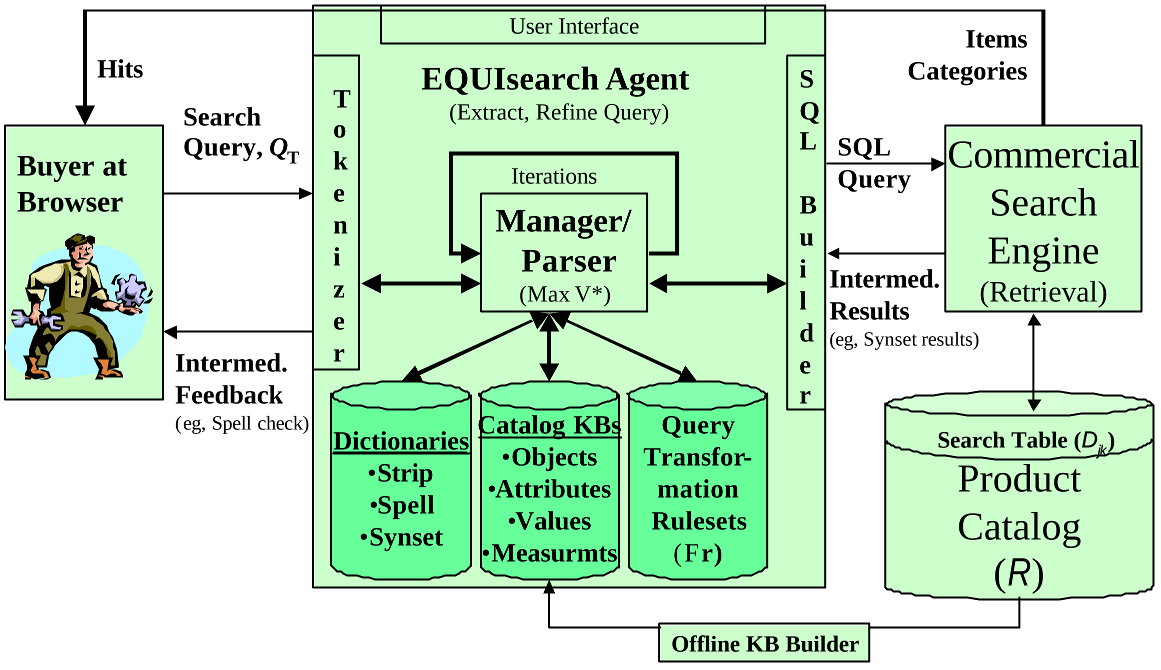
Thus, recursive equation (2) summarizes the standard computable "value iteration" formulation. To use this we must define the permissible states and actions, the reward function, the transition probabilities, and the other terms of the equation within the context of catalog search problems. We refer the interested reader to [11] for these details and provide only an overview of these items in what follows and in Figure 1.

As a language parser the agent iteratively labels each stripped and stemmed token of the query and subsequently modifies these labels by applying a sequence of transformation rules which attempt to reduce the remaining ambiguities and residual errors left in place by the previous rules. In doing so it labels the state of each term in the query, where state is defined within a sub-grammar. The sub-grammar we created here is called "OAV Triplet Grammar." OAV triplets, or object-attribute-value triplets, are meta-data describing what's in catalogs much the same way RDFs describe the content of the web [12, 13]. Specifically, in product catalog domains, searches are usually for <u>o</u>bjects with <u>a</u>ttributes of a certain <u>v</u>alue. This is something of a noun phrase that includes adjectives. For example, some triplets in various orderings might be: 'mini sized amps' or 'hammer colored red'. The order of the triplets is a task for the agent to discover. Thus a common order variant occurs where the O is sought initially and then the AVs are subsequently used for comparison and search refinement (bolt cutter followed by size, price, and availability). Even more common are searches for one or more V of a given type of O where the A is suppressed (eg, AA Eveready batteries, ½” no.8 slotted screws, or desk chair).

In addition to the rules of the sub-grammar, one obviously must also derive the vocabulary for any given instantiation or catalog. In shopping domains, the vocabularies can be derived for objects and another for attribute-value pairs by crawling the fields of the relational catalog database and by extracting all unique terms (item names are object name, attributes are parameter names, and parameter settings are attribute values). We show this tool for crawling the catalog and extracting the KB values at the base of Figure 1. Using such a tool, one can readily construct the sub-grammar and its vocabulary for any given electronic catalog. The result is stored in KB lookup tables (base of agent in Figure 1) so it can be utilized by transformation rules of the phrase parser to infer and insert O, A, and V labels onto the various terms of any given search phrase. Also at the bottom right of the agent box are shown other rules (Φ_r) that are used to infer OAV meanings from queries.

It is not sufficient that the vocabulary extracted from the catalog provides 100% coverage of the lexicon of a given shopping site if the users of the site are unaware of that lexicon or prone to misusing it. To help overcome such semantic impasses, the tokens in the query string are first stemmed, subsequently expanded for synonyms, and ultimately checked for spelling (bottom left of the agent in Figure 1). This is supported by adding the dictionaries and KBs across the base of Figure 1.

Figure 1 – Overview of The EQUIsearch Agent as a Meaning Translator In the Interface Between Users and Traditional Search Engines



3) Comparison Testing of EQUIsearch's Capabilities

Earlier we said that EQUIsearch is complementary to commercial search engines for online catalogs. To illustrate that point, this section provides an empirical evaluation of query performance both with and without the EQUIsearch Agent for a major commercial search engine relevant to e-commerce catalogs. In general, search engine performance can be measured in terms of four methods: retrieval effectiveness metrics (e.g., recall, precision), user satisfaction measures, transaction log analysis, and the critical incident technique. Silverman et al [10] examine the last three of these. In the current study, we provide a comparison based on effectiveness metrics.

3.1) Measuring Query Effectiveness (recall, precision)

The information retrieval literature defines effectiveness as a measure of the ability of the system to satisfy the user in terms of the relevance or pertinence of items retrieved [8, 12]. Pertinence is assumed to mean 'aboutness' and 'appropriateness', that is, a document is ultimately determined to be pertinent or not by the user (or us). It is helpful at this point to introduce the ubiquitous 'contingency table' shown in Table 1.

Table 1 – "Contingency Table" For Deriving Effectiveness Metrics

	Pertinent (**P** = RP + NP)	Irrelevant (**I** = RI + NI)
Retrieved (**R** = RP + RI)	**RP** "Hits"	**RI** Type I errors
Not-Retrieved (**N** = NP + NI)	**NP** Type II errors	**NI**

Total Catalog = RP + RI + NP + NI

For our purpose we will use ratios of (hits / retrived) and (hits/ pertinent).

PRECISION = RP/(RP+RI)
RECALL = RP/(RP+NP)

3.2) Scaleup Test Results

In order to test whether NLQ can scale up, we have deployed it at a market exchange. Specifically, the website is EqualFooting.com (www.equalfooting.com), a B2B online marketplace for the "maintenance, repair, and operations" (MRO) sector: e.g., see [10]. This means basically EqualFooting sells industrial and construction supplies – something like a Home Depot for small contractors only with an order of magnitude more products. The company's official launch date was February 2000 and by June 2000 they were handling one million hits per day (by about 23,000 separate users daily). Also, at this writing their catalog integrates almost 450,000 products offered by over 2,000 sellers.

The catalog is stored internally in an Oracle database. Users at this website now utilize the NLQ agent (its called EQUIsearch), that in turn interfaces with Oracle and its CQ search technology called Oracle *inter*Media. It should be noted that the conceptual search features of a product such as *inter*Media are not immediately usable at a given shopping site. For example, in order to make the production version of the Equalfooting.com database CQ-capable required creating the three dictionaries always needed for conceptual search – spelling, stripping, and synonyms – as these are domain-specific items. Growing the thesaurus involved many false starts and deadends. For example, it is tempting to try and use an existing general purpose thesaurus such as WordNet from Princeton. This includes 95,000 words and all their synonyms, however, this thesaurus brings back too many synonyms, many of which are inappropriate (racial slurs, curses, body parts, religious terms, etc.). Plus most of the specialty terms of a given domain are omitted (e.g., chain saw, Phillips head, and safety gloves are among the 1,000s of items found in a hardware catalog). Instead, based on search log analyses, EqualFooting assembled their own thesaurus for almost 8,000 synonyms critical to this domain. The second dictionary task was to assure the database would have a spell checker (Oracle doesn't ship with this functionality), so one was purchased and installed separately. Although it came with 100,000 words it was necessary to embellish the spell checker's dictionary by adding: (1) the top 1,000 mis-spelled words from the user search logs and their corrections, (2) proper names of all manufacturers and suppliers (the spell-checker assumes initially that all proper names are errors) and how they might be mis-spelled, and (3) many 100s of acronyms with proper spelling (e.g., CD, DVD, HVAC, etc.). The third and final dictionary task was to massage the stop word list for the current domain as some generic stop words are relevant here and others are unique.

For a site that already has prepared all three dictionaries required by CQ (i.e., strip, synset, spell), the extra effort to add EQUIsearch is rather straightforward. The steps are to:

1) run a catalog crawler that extracts all the lexicon
2) construct the lexical knowledge bases (objects and attribute-value pairs)
3) author the relevant rule sets and encode them in the Φ_r
4) complete any interface code needed in the SQL builder of the agent to adapt it to the requirements of the CQ technology (*inter*Media in this case)

We performed these steps manually for the test site during the Fall 2000, and the agent was deployed as of November 17, 2000. It has been in 24x7 continuous operation since.

3.3) Comparison Test Results

For our comparison testing we ran three types of query strings as listed in Table 2. Nouns or objects are the item names or synonyms of those names. Noun-adjective pairs (object-attribute pairs) cover the case where the user seeks to narrow the choices returned. Lastly, although most users are unaccustomed to typing in sentence format, we felt it an important category of search since future users will hopefully be unaware of such search engine constraints. Also, statistics on current search habits indicate that users type about 2.3 words per query with nouns being the most common search, noun-adjective pairs next most common, and multi-word phrases being least common [1,2].

Our test bank shown in Table 2 was designed to approximate this distribution of 2.3 words per query on average (i.e., 146 words divided by 63 queries = 2.3 words/query).

Table 2 – Sixty Three Query Strings Used for the CQ vs. NLQ Comparison Test

Nouns (31)		Noun-Adjective Pairs(18)	Sentences (14)
aircompressor	hammer	16 oz hammer	show me all power cords
ballast	ladder	10 inch nail	list me cotton gloves
blower	mallet	bolt cutter	show me gloves made of leather
brad	mitten	copier paper	i want to buy leather gloves
brush	nail	cotton glove	show me steel brushes
cabinet	paper	crimped brush	list me some good cotton gloves
calculator	pipe	cutter wheel	let me see some leather gloves
chipper	processor	leather glove	list me nails for roofing
chisel	saw	nail hammer	list me all wheel cutters
cleaner	screw	pipe clamp	show me bolt cutter
compressor	snapper	power cord	list wire connectors
connector	soap	protective apron	list brush made of steel
cupboard	tube	roofing nail	i want to buy a crimped brush
cutter		safety tape	let me see leather gloves if you have any
drill		steel brush	
fans		tape measure	
glove		white paper	
grinder		wire connector	

Table 3 provides the comparison of effectiveness and timing statistics for the CQ search (via Oracle *inter*Media) vs. the NLQ agent. These results were collected over EqualFooting's intranet and ignore Internet latency. Inspecting the last row of Table 3 shows that, on average, the CQ search was nearly twice as fast (.5 vs. .9 seconds/query), however, it tended to retrieve nearly three times as many hits even though its recall was not as good as NLQ (.8 vs. 1.0) and its precision was only half as good (.5 vs. 1.0). Further, the time advantage of CQ search is relatively inconsequential since most users can't detect a half second difference.

Consider also the first row of the table, noun search, where we see a clear advantage being held by NLQ across the board. Not only is NLQ more precise, but it's faster as well. The only place CQ holds its own is in recall, where both methods are perfect in bringing back all the pertinent items.

Table 3 – Timing and Effectiveness Results for CQ Search vs. the NLQ Agent

Search	**Total Retrieved**		**Precision**		**Recall**		**2 Run Avg Time**	
	CQ Search	**NLQ Agent**	CQ Search	**NLQ Agent**	CQ Search	**NLQ Agent**	CQ Search	**NLQ Agent**
31 Nouns	2965	1091	0.6	1	1	1	0.6	0.4
18 Noun-Adjective Pairs	160	66	0.6	1	1	1	0.5	1.9
14 Sentences	1	71	0	1	0	1	0.4	0.8
Average (for 63 queries)	1580	571	0.5	1.0	0.8	1.0	0.5	0.9

This story changes for noun-adjective pair search (row 2), at least in terms of speed, where NLQ now takes almost 4 times longer than CQ even though it still wins on precision. It should surprise no one that the NLQ is faster than CQ in noun search and slower in noun-adjective search. In noun or object search, CQ must dynamically sort and index the entire munge (many sub-fields de-normalized) while NLQ need only sort through the Item-Name field since it has narrowed the search by labeling the token. Likewise, in object-attribute search, CQ behaves the same as before, but NLQ causes the Oracle *inter*Media to sort an additional set of fields (all those with attributes) and then do a soft join and eliminate items not in all parts of the join. The fact that sentences (row 3) are faster than word pairs (row 2) for NLQ only further confirms the idea that the parsing processes are not very significant determinants of delay. The specific terms of the query are what drive the process. In this case the noun-adjective pairs that the parser extracted from the sentences proved to be less common items in the database, and hence the Oracle CQ engine could process them faster.

4) Results Analysis and Concluding Remarks

This paper has presented an NLQ agent that can complement and improve search engines for online e-commerce shopping catalogs. The results show that the agent improves the precision and recall of the search with no significant overall impact on response time. Several lessons learned are worth discussing further:

- **Scaling Up NLQ** – In order to scale up NLQ, we did away with some of the less practical NLQ proposals in the literature, such as conversational feedback and explanation of the translation of the query to be user prior to executing the query. Instead we converse with the user in the fashion that CQ systems use -- after the query via the display either of hits or of pre-canned, limited explanations of query failures. Also, we implemented it as a Markovian Decision Process. The agent worked and is continuing to work at a relatively large-scale market exchange, which previously operated with CQ search alone.
- **CQ Paves the Way for NLQ** – Relational DBMSs ship with CQ features, however, the three standard dictionaries (strip, synonym, and spell) need to be enabled and domain-filled before CQ can work for a given shopping catalog. The good news is that these are the same three dictionaries that NLQ needs, and if a site has already developed them and deployed CQ, then there is only minimal extra effort to also deploy NLQ.
- **Data Cleansing Obstacles Remain for Any Search Method** – Possibly the most serious obstacle to scale up in any unified shopping catalog is the numerous typoes, missing item name and other data, and poor quality of attribute information. This obstacle is not unique to NLQ search, but equally plagues CQ and keyword searching methods.
- **Speed Differences are a NonIssue For Most Cases** – It seems that NLQ is faster for noun search, but slower than CQ for noun-attribute pair searching. But in either case, the time differences are not statistically significant.
- **NLQ Agent Provides Precision and Recall Improvement** – The results to date reflect about a 50% improvement in precision when NLQ is added to CQ. This means that users experience shortened retrieval sets, and that the items retrieved include far less false positives. In addition there are fewer false negatives or relevant items omitted.
- **NLQ Agent Offers Parsing Services CQ Can't Provide** – Many shopping sites are starting to add chatterbots like Dell's AskDudley [7] that provide navigation help and answer site or content questions in natural-like language. The results to date indicate that users like this type of self-service help, and that when it is present, they build up a higher

expectation that the catalog search will behave in a similarly naturalistic way and they no longer limit their queries to the short keyword format (2.3 words on average) [2]. They pose English-like sentences and questions to the catalog search engine and it seems they are adversely affected by the inability of CQ search engines to parse their questions. As a result, NLQ agents, such as the one demonstrated, here appear to be the answer for shopping sites facing this dilemma.

REFERENCES

1. Anon., "Winning the Online Consumer: Insights Into Consumer Behavior," Cambridge: Boston Consulting Group, March 2000, www.bcg.com.
2. Hagen, PR, Manning, H, Paul, Y, "Must Search Stink?", Cambridge: Forrester Research, June 2000. (www.forrester.com)
3. Owei, V., "Natural Language Querying of Databases: An Information Extraction Approach in the Conceptual Query Language," IJHCS, v. 53, 2000, pp. 439-92.
4. Blum, A., "Add Natural Language Search Capabilities to Your Site with English Query," Microsoft Interactive Developer, April 1998, www.microsoft.com/Mind/0498/equery.htm
5. Anon., "Revolutionizing the Search for Products at E-Commerce Sites," Littleton: Easy Ask Inc., March 2000, www.easyask.com
6. "BotSpot Categories - Chatter Bots," http://www.botspot.com/search/s-chat.htm
7. http://support.dell.com/us/en/askdudley/
8. VanRijsbergen, C.J., Information Retriveral, London: Butterworth, 1979.
9. Dekleva, SM, "Is Natural Language Querying Practical?" Data Base, v. 25, 1994, pp. 24-36.
10. Silverman, BG, Bachann, M, et al, "Buyer Decision Support Systems and Search Agents for eCommerce Websites," IJHCS (to appear),
11. Silverman, BG, Bachann, M, et al, "A Markov Decision Processing Solution to Natural Language Querying of Online e-Commerce Catalogs: , J. of OR in Computing, (submitted).
12. Hendler, J., " Agents and the Semantic Web" IEEE Intelligent Systems, March 01, pp. 30-37.
13. Klein, M., "XML, RDF, and Relatives," IEEE Intelligent Systems, March 01, pp. 26-28.
14. MT Pazienza (ed.), Information Extraction: Toward Scalable, Adaptable Systems, Berlin: Springer, 1998, pp. 95-119.
15. Strzalkowski, T, "Natural Language Information Retrieval, Dodrecht: Kluwer, 1999.
16. Bates, M. J.,"Indexing and access for digital libraries and the Internet: Human, database, and domain factors," Journal of the American Society for Information Science, 49(13), 1998, pp.1185-1205.
17. Sparck-Jones, K., & Willett, P., Readings in information retrieval, Mountain View: Morgan Kaufmann, 1997.
18. Cunningham, H, "Information Extraction, a User Guide," Journal of Computer Science 1997, Vol.2