The Wasabi Personal Shopper: A Case-Based Recommender System

Robin Burke

Recommender.com, Inc.
and
University of California, Irvine
Dept. of Information and Computer Science
University of California, Irvine
Irvine, California 92697
burke@ics.uci.edu

Abstract

The Wasabi Personal Shopper (WPS) is a domain-independent database browsing tool designed for on-line information access, particularly for electronic product catalogs. Typically, web-based catalogs rely either on text search or query formulation. WPS introduces an alternative form of access via preference-based navigation. WPS is based on a line of academic research called FindMe systems. These systems were built in a variety of different languages and used custom-built ad-hoc databases. WPS is written in C++, and designed to be a commercial-grade software product, compatible with any SQL-accessible catalog. The paper describes the WPS and discusses some of the development issues involved in re-engineering our AI research system as a general-purpose commercial application.

Introduction

Although electronic commerce is clearly burgeoning, electronic product catalogs leave much to be desired. Typically implemented as a form-based front-end to an SQL database, these catalogs are frustrating to the user who does not know exactly what she is looking for. Other catalogs rely on text search. In one software catalog, entering "Windows NT Workstation" brought back over a thousand responses, since the names of compatible operating systems are often mentioned in product descriptions.

The Wasabi Personal Shopper (WPS) provides a conversational interface to a database, based on the principles of case-based reasoning. The user examines a suggestion from the system – an item from the catalog – and responds to it with a critique or *tweak*. The system uses the item and the associated tweak to formulate a new query returning a new item for consideration. The result is a natural traversal of the catalog, honing in on the product that best meets the user's needs.

Example

The Wasabi Personal Shopper (WPS) has been applied to a database of wines known as the VintageExchange <URL: http://www.vintageexchange.com>. Still in development, VintageExchange aims to be a web-based market-maker for individuals trading wines from their personal cellars. When we became involved, the site had already implemented a database of wines using descriptions and ratings licensed from the *Wine Spectator*, a well-known wine industry periodical. Consider the following hypothetical exchange between a user and WPS as implemented with the VintageExchange database:

> Vintage Vinnie selects a bottle from his on-line cellar database and marks it as deleted, since he drank it last night. It was a 1994 Rochioli Russian River Pinot Noir (Three Corner Vineyard Reserve). The *Wine Spectator* describes it as follows: "Delicious Pinot Noir from the first sip. Serves up lots of complex flavors, with layers of ripe cherry, plum and raspberry and finishes with notes of tea, anise and spice. Tannins are smooth and polished and the finish goes on and on, revealing more nuances." (Rating: 94)
>
> That was the last one in the case, so Vinnie checks to see if any similar wines are available. He selects "Find similar." The system returns with ten more Pinot Noirs, the top-most being a 1991 Williams Selyem Russian River (Rochioli Vineyard). It gets a higher rating (95) from the *Spectator*: "Rich, ripe and plummy, this is generous from start to finish, dripping with vanilla-scented fruit and spice flavors while remaining smooth and polished. The flavors linger enticingly. Delicious now."
>
> Sounds great, but Vinnie notes the price ($45) and clicks on the "Less $$" button. The answer is a 1988 Pinot Noir Mount Harlan Jensen from California's central coast. It is still rated highly (92) and is $10 cheaper. The description says "Packs in a load of

> fresh, ultraripe, rich black cherry, currant, herb and spicy earth overtones. Deeply flavored and very concentrated, with smooth, supple tannins and a long, full, fruity finish. A distinctive wine. Drink now." Vinnie clicks "Buy."

This example shows how WPS allows the user to express his preferences without having to make those preferences explicit in a query. The wine that he had and liked serves as Vinnie's entry point into the system, and his preference to spend less is conveyed with a simple button click. There are about 2,400 Pinot Noirs in the system, many at the same price and rating points. All the wines found are described as having large quantities of fruit, being "smooth" and "spicy", and having long finishes. While these are qualities of many good wines, other comparable Pinot Noirs are described quite differently. For example, an otherwise similar 1989 version of the same Mount Harlan Jensen wine gets the following description:

> Broad, rich and spicy, with nice toast, coffee and brown sugar nuances to the basic plum and currant aromas and flavors that linger. The finish echoes fruit and flavor as it spreads. Approachable now, but best after 1995.

This one lacks the fresh fruit flavors found in the previous wine, and mentions burnt overtones (toast, coffee) absent in the first suggestion. Attention to such subtleties lets the WPS-enabled catalog help users find items that meet their preferences without having to make those preferences explicit.

FindMe Systems

The Wasabi Personal Shopper has its roots in a line of research known as FindMe systems.[1] See <URL: http://infolab.ils.nwu.edu/entree/> for an example of a publicly-accessible FindMe system: the Entree restaurant guide, which has been on-line since the summer of 1996.

The FindMe technique is one of knowledge-based similarity retrieval. There are two fundamental retrieval modes: similarity and tweak application. In the similarity case, the user has selected a given item from the catalog (called the *source*) and requested other items similar to it. First, a large set of candidate entities is retrieved from the database. This set is sorted based on similarity to the source and the top few candidates returned to the user.

Tweak application is essentially the same except that the candidate set is filtered prior to sorting to leave only those candidates that satisfy the tweak. For example, if a user responds to an item with the tweak "Nicer," the system determines the "niceness" value of that source item and rejects all candidates except those whose value is greater.

Our initial FindMe experiments demonstrated something that case-based reasoning researchers have always known, namely that similarity is not a simple or uniform concept. In part, what counts as similar depends on what one's goals are: a shoe is similar to a hammer if one is looking around for something to bang with, but not if one wants to extract nails. Our similarity measures therefore have to be goal-based. We also must consider multiple goals and their tradeoffs, which are always involved in shopping. Typically, there are only a handful of standard goals in any given product domain. For each goal, we define a *similarity metric*, which measures how closely two products come to meeting the same goal. Two products with the same price would get the maximum similarity rating on the metric of price, but may differ greatly on another metric, such as quality.

We looked at the interactions between goals, and experimented with complex combinations of metrics to achieve intuitive rankings of products. We found there were well-defined priorities attached to the most important goals and that they could be treated independently. For example, in the restaurant domain, cuisine is of paramount importance. Part of the reason is that cuisine is a category that more or less defines the meaning of other features – a high-quality French restaurant is not really comparable to a high-quality burger joint, partly because of what it means to serve French cuisine.

We can think of the primary category as the most important goal that a recommendation must satisfy, but there are many other goals also. FindMe systems order these goals. For example, in the Entree restaurant recommender system, the ranking of goals was cuisine, price, quality, atmosphere, which seemed to capture our intuition about what was important about restaurants. As part of the development of WPS, we realized that different users might have different goal orderings or different goals altogether. This gave rise to the concept of the *retrieval strategy*. A retrieval strategy selects the goals to be used in comparing entities, and orders them. In VintageExchange, the standard goal ordering is wine type, price, quality, flavor, body, finish, and drinkability, but the system also has a "money no object" strategy in which price is not considered.

Given the goal ordering in the strategy, the process of finding the most similar candidate becomes an alphabetic sort. We sort the candidates into a list of buckets based on their similarity to the source along the most important goal, then sort each of these buckets based on the next most important goal, creating a new list of finer-grained buckets. (The details of the sort algorithm are discussed in [Burke, Hammond & Young, 1997].)

One benefit of this solution is that we can iteratively reduce the set of candidates under consideration as the sort proceeds. Since we are only interested in returning a small number of recommendations to the user, we need to preserve only enough buckets to ensure that we have enough candidates. Buckets farther down in the sort order can be deleted since their contents would never be presented.

[1] See (Burke, Hammond & Young, 1997) for a full description of the systems involved in this effort.

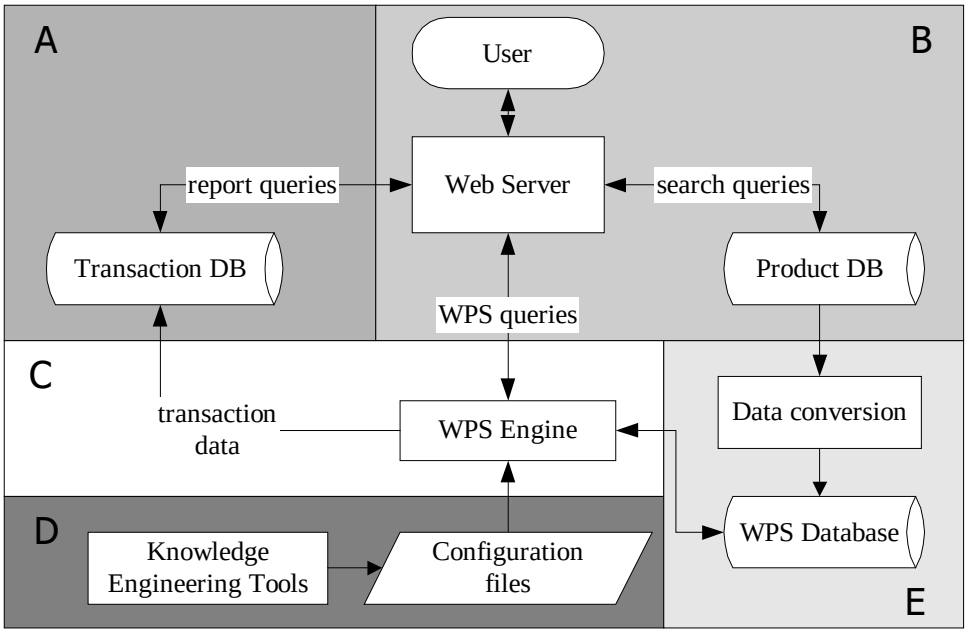

Figure 1. Wasabi Personal Shopper architecture

Architecture

Our research systems showed that the fundamental idea of FindMe worked well. In our informal evaluations, our prototypes returned products that users agreed were similar to their input entries, and the process of navigation by tweaking proved to be a natural, conversational, way to focus in on a desired item. We built enough systems to demonstrate that the idea had cross-domain applicability.

Several obstacles remained to the commercialization of FindMe. Obviously, it would be unreasonable to expect FindMe to be the only interface to a product catalog. It would have to be an add-on on to an existing product database, most likely a relational database using SQL.

Second, all of our research systems had been custom-built. The WPS had to be capable of navigating through catalogs containing any type of product. So, the system needed to be completely general and make as few assumptions about products as possible.

Finally, a truly useful catalog system needs to provide feedback to its owners. Because of the user's unique interaction with WPS, a user profile created while using the system is uniquely valuable, revealing the user's preferences with respect to specific items in the inventory, including "holes" in the product space: places where many customers attempt the same tweak, but come up empty-handed.

The WPS architecture was our response to these new requirements. It generalizes and encapsulates the previous FindMe work, making possible a commercial-grade application. The architecture has five basic parts as shown in Figure 1:

External Information Environment (B in the diagram): We assume WPS will be running in a standard catalog serving environment where the web server will receive requests and forward them to WPS, possibly through an application server.

WPS Engine (C in the diagram): At the core of the system is the WPS Engine. It is responsible for retrieving entities, applying metrics, sorting and returning answers.

Knowledge Engineering (D in the diagram): The engine knows nothing of wines, restaurants, or any other catalogued product. It only knows entities and their features, and the metrics that operate on them. Metrics and strategies are defined as part of the configuration of the system through a knowledge engineering step.

WPS Database (E in the diagram): In the WPS Database, the system essentially has a copy of the product database in its own format. The transformation process that creates this database essentially flattens the catalog into our feature-set representation. Designing this transformation is another important knowledge engineering step involved in building a WPS system.

Profiling and Reporting (A in the diagram): As discussed above, the ability to profile users and identify patterns of catalog browsing is a crucial advantage that WPS brings. The engine creates a transaction database, recording the history of user interactions with the system, to allow for report generation.

The WPS architecture is a work in progress. As of this writing, only the WPS Engine is in its final form. Our tools for data conversion are still domain- and database-dependent. We have high-level specifications and

preliminary designs for the knowledge engineering tools, but no implementations. The configuration files for existing WPS applications are created and maintained manually. We are also still implementing the transaction logging and reporting features.

Database Integration

The most significant implementation issue we faced was one of scale. Our hand-built databases or Lisp data structures could not be expected to scale to very large data sets. So it was imperative that our persistent store of entities and their features be a database. We chose relational databases instead of object-oriented ones since relational databases are the most widely-deployed systems for web applications, although OODBs would have matched our data model somewhat better.

Since we needed to be completely general, we chose a very simple database schema. Each entity has an entity ID, which serves as the primary key for accessing it in the database. The row associated with an entity ID contains two types of columns: single integer values and fixed length bit fields. The single integer columns are used for singleton features like price, where each entity has only one value. The bit field columns are used to hold multi-feature values, such as all of the flavor features of a wine.

Recall that the FindMe algorithm as originally designed took a large set of candidates and performed a series of sorting operations. We used promiscuous retrieval deliberately because other steps (such as tweaking steps) filter out many candidates and it was important not to exclude any potentially-useful entity. In our Lisp implementations, we found that applying the algorithm to a large candidate set was reasonably efficient since the candidates were already in memory. This was definitely not true in the database: queries that return large numbers of rows are highly inefficient, and each retrieved entity must be allocated on the heap. Employed against a relational store, our original algorithms yielded unacceptable response times, sometimes greater than 10 minutes.

It was necessary therefore to retrieve more precisely – to get back just those items likely to be highly rated by the sort algorithm. We needed to widen the scope of interaction between the engine and the database. We considered a fully-integrated approach: building the metrics and retrieval strategies right into the database as stored procedures. However, we opted to keep our code separate to preserve its portability.

Our solution was a natural outgrowth of the metric and strategy system that we had developed for sorting, and was inspired by the CADET system, which performs nearest-neighbor retrieval in relational databases (Shimazu, Kitano & Shibata, 1993). Each metric became responsible for generating retrieval constraints based on the source entity. These constraints could then be turned into SQL clauses when retrieval took place. This approach was especially powerful for tweaks. A properly-constrained query for a tweak such as “cheaper” will retrieve only the entities that will actually pass the “cheaper” filter, avoiding the work of reading and instantiating entities that would be immediately discarded.

The retrieval algorithm works as follows. To retrieve candidates for comparison against a source entity, each metric creates a constraint. The constraints are ordered by the priority of the metric within the current retrieval strategy. If the query is to be used for a tweak, a constraint is created that implements the tweak and is given highest priority. This constraint is considered “non-optional.” An SQL query is created conjoining all the constraints and is passed to the database. If no entities (or not enough) are returned, the lowest priority constraint is dropped and the query resubmitted. This process can continue until all of the optional constraints have been dropped.

The interaction between constraint set and candidate set size is dramatic: a four-constraint query that returns nothing will often return thousands of entities when relaxed to three constraints. We are considering a more flexible constraint scheme in which each metric would propose a small set of progressively more inclusive constraints, rather than just one. Since database access time dominates all other processing time in the system, we expect that any additional computation involved would be outweighed by the efficiencies to be had in more accurate retrieval.

Similarity metrics

In general, a similarity metric can be any function that takes two entities and returns a value reflecting their similarity with respect to a given goal. Our FindMe systems implemented the similarity metric idea in many different domain-specific ways. For WPS, we have created a proprietary set of metrics general enough to cover all of the similarity computations used in our other FindMe systems and sufficient to implement the VintageExchange system. Internally, they are implemented as a single metric class with a comparison-computing “strategy” object (Gamma, et al. 1995) attached, so new metric types can be easily added if they are needed.

Natural language processing

As the example at the beginning of this paper shows, textual descriptions can be crucial discriminators in large catalogs of otherwise similar objects. We have found that many of the databases used for electronic commerce contain small amounts of descriptive data associated with each product and large chunks of human-readable text. As of this writing, the natural language component of WPS is not as well developed as the WPS Engine. We have implemented it only for the VintageExchange project, and only now beginning to explore cross-domain applications. However, the language of wine represents something of a worst case for textual description, so we are optimistic that our approach is robust.

Recall that an entity is simply a collection of features. A natural language description must therefore be transformed into atomic features. For wines, we identified four categories of descriptive information: descriptions of the flavor of the wine ("berry", "tobacco"), descriptions of the wine's body and texture ("gritty", "silky"), descriptions of the finish ("lingering", "truncated"), and descriptions of how and when the wine should be drunk ("with meat", "aperitif"), and we identified the most commonly-used terms, usually nouns, in each of these categories. We also identified modifiers both of quantity ("lots", "lacking") and quality ("lovely", "harsh").

Each description was processed as follows. First we broke the description into phrases, then we associated each phrase with a description category, since we had found that each phrase usually concentrated on one aspect of the wine. We eliminated stopwords and performed some simple stemming. At this point, we had four word groups, one for each description category, consisting of descriptive terms and modifiers, and marked with phrase boundaries. Using the phrase boundaries, we associated modifiers with each descriptive term in their scope. For example, in the description "a lovely fragrant explosion of cherry and raspberry", the "lovely fragrant explosion" refers to both of the fruit terms. This gave us a representation based on terms and modifiers.

The "term plus modifier" representation was further simplified by breaking modifiers into classes: quantity modifiers could either refer to a large quantity ("oodles"), a small quantity ("scant"), or a zero quantity ("lacking"); quality modifiers could be positive ("tasty") or negative ("stale"). Only the modifier count was retained. "A lovely fragrant explosion of cherry and raspberry" therefore became (large-quantity positive-quality*2 cherry) (large-quantity positive-quality*2 raspberry), since "lovely" and "fragrant" are both positive quality modifiers. The resulting representation loses the poetry of the original, but retains enough of its essence to enable matching.

The final step was to turn this representation into an encoding for manipulation by WPS. Recall that features in WPS are represented as integers. Treating an integer as a 32-bit vector, we used the most-significant 20 bits to represent the content term and the least-significant 12 to represent the modifiers. So, the encoding of our phrase above would ultimately come down to the following two hexadecimal integers: `803eb109`, and `80388109`, for cherry and raspberry, respectively. Note that the last three hex digits (12 bits) are the same, reflecting the modifier encoding, while the first 20 bits represent the particular flavor terms.[1]

Preliminary analysis of other e-commerce data sets suggests that product descriptions in general are not complex syntactically and concentrate on straightforward descriptive adjectives and nouns. Few domains have the breadth of vocabulary present in our wine descriptions. Our experience in building the next few WPS applications will reveal more about the generalizability of the natural language approach described here.

[1] The first three digits, `803`, also happen to be the same, since cherry and raspberry are both part of the "fruit" portion of the flavor hierarchy and are near each other in the configuration file. This, however, is an artifact of our numbering scheme and is not used by the metric.

Related Work

The problem of intelligent assistance for browsing, especially web browsing, is a topic of active interest in the AI community. There are a number of lines of research directed at understanding browsing behavior in users (Konstan, et al. 1997; Perkowitz & Etzioni, 1998), extracting information from pages (Craven, et al. 1998; Knoblock et al. 1998, Cohen, 1998), and automatically locating related information (Lieberman, 1995). Because the web presents an unconstrained domain, these systems must use knowledge-poor methods, typically statistical ones.

WPS has a different niche. We expect users to have highly-focused goals, such as finding a particular kind of wine. We deal with database records, all of which describe the same kind of product. As a result, we can build detailed knowledge into our systems that enables them to compare entities in the information space, and respond to user goals.

In information retrieval research, retrieval is seen as the main task in interacting with an information source, not browsing. The ability to tailor retrieval by obtaining user response to retrieved items has been implemented in some information retrieval systems through retrieval clustering (Cutting, et al., 1992) and through relevance feedback (Salton & McGill, 1983).

Schneiderman's "dynamic query" systems present another approach to database navigation (Schneiderman, 1994). These systems use two-dimensional graphical maps of a data space in which examples are typically represented by points. Queries are created by moving sliders that correspond to features, and the items retrieved by the query are shown as appropriately-colored points in the space. This technique has been very effective for two-dimensional data such as maps, when the relevant retrieval variables are scalar values. Like WPS, the dynamic query approach has the benefit of letting users discover tradeoffs in the data because users can watch the pattern of the retrieved data change as values are manipulated.

As discussed earlier, the closest precedent for our use of knowledge-based methods in retrieval comes from case-based reasoning (CBR) (Hammond, 1989; Kolodner, 1993; Riesbeck & Schank, 1989). A case-based reasoning system solves new problems by retrieving old problems likely to have similar solutions. Researchers working on the retrieval of CBR cases have concentrated on developing knowledge-based methods for precise, efficient retrieval of well-represented examples. For some tasks, such as case-based educational systems, where cases serve a variety of

purposes, CBR systems have also explored multiple goal-based retrieval strategies (Burke & Kass, 1995).

Our use of tweaks is obviously related to CBR research in case adaptation. Note however, that our use of the term is different. Tweaking in the context of CBR means to adapt a returned case to make it more closely match the problem situation in which it will be applied. The tweaks that a user invokes in WPS are applied much differently. We cannot invent a new wine, or change an existing one to match the user's desires – the best we can do is attempt a new retrieval, keeping the user's preference in mind.

Future work

A key element of future releases of Wasabi Personal Shopper will be the addition of the knowledge engineering tools. There are four major steps of the WPS knowledge engineering process: (1) identifying the anticipated goals of shoppers and their priorities, (2) mapping from these goals to entries in the appropriate columns in the appropriate database table, (3) isolating those elements of product data that will serve as features, possibly performing natural language processing, and (4) determining how similarity should be judged between the features so extracted.

We feel that step 1 will not be difficult for marketing experts, who already spend their days thinking about what motivates shoppers to buy their companies' products. Step 2, however, will present a user-interface challenge. We do not want to assume that the user has any database expertise, yet this task requires that we specify exactly what parts of a database record correspond to the price, the color, etc. The most difficult task technically is step 3. Here we are building the system's conceptual vocabulary, starting from raw database entries. Finally, our inventory of similarity metrics gives us great leverage over step 4. We have designed strategy-specific wizards that guide the user through the problem of tailoring each metric.

Many e-commerce sites use some form of collaborative filtering (Konstan et al. 1997). These systems make suggestions based on preferences of other users with similar profiles. Our profiling data could make such systems more powerful (since we get useful data from every browsing step, not just the final result). One could also imagine using collaborative filtering data to bias the browsing process, either as part of the retrieval strategy, or as a fallback when the tweaking process hits a dead end.

Acknowledgements

The FindMe research discussed in this paper was performed at the University of Chicago in collaboration with Kristian Hammond, with the assistance of Terrence Asselin, Jay Budzik, Robin Hunicke, Kass Schmitt, Robb Thomas, Benjamin Young, and others. The development of the Wasabi Personal Shopper system benefited from the insights gained during a JAD workshop with Geneer, Inc. and from the efforts of Cate Brady, Drew Scott, Jim Silverstein, Paul Steimle, and Peter Tapling. Thanks also to Mike Lyons and Stephen Schmitt of ArribaTech who made the VintageExchange database available to us.

References

Burke, R., & Kass, A. 1995. Supporting Learning through Active Retrieval of Video Stories. *Journal of Expert Systems with Applications*, 9(5).

Burke, R., Hammond, K., and Young, B. 1997. The FindMe Approach to Assisted Browsing. *IEEE Expert*, 12(4): 32-40.

Cohen, W. W. 1998. A Web-based Information System that Reasons with Structured Collections of Text. In *Proceedings of the Second International Conference on Autonomous Agents*, New York: ACM Press. pp. 400-407.

Craven, M., DiPasquo, D., Freitag, D. , McCallum, A., Mitchell, T., Nigam, K. & Slattery, S. 1998. Learning to Extract Symbolic Knowledge from the World Wide Web. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*. AAAI Press, pp. 509-516.

Cutting, D. R.; Pederson, J. O.; Karger, D.; and Tukey, J. W. 1992. Scatter/Gather: A cluster-based approach to browsing large document collections. In *Proceedings of the 15th Annual International ACM/SIGIR Conference*, pp. 318-329.

Gamma, E., Helm, R., Johnson, R. & Vlissides, J. 1995. *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, MA: Addison-Wesley.

Hammond, K. 1989. *Case-based Planning: Viewing Planning as a Memory Task*. Academic Press. Perspectives in AI Series, Boston, MA.

Knoblock C. A., Minton, S., Ambite, J. L., Ashish, N., Modi, P. J., Muslea, I., Philpot, A. G., and Tejada, S. 1998. Modeling Web Sources for Information Integration. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*. AAAI Press. pp. 211-218.

Kolodner, J. 1993. *Case-based reasoning*. San Mateo, CA: Morgan Kaufmann.

Konstan, J., Miller, B., Maltz, D., Herlocker, J., Gordon, L., and Riedl, J. 1997. GroupLens: Applying Collaborative Filtering to Usenet News. *Communications of the ACM* 40(3): 77-87.

Perkowitz, M. & Etzioni, O. 1998. Adaptive Web Sites: Automatically Synthesizing Web Pages. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*. AAAI Press. pp. 727-731.

Riesbeck, C., & Schank, R. C. 1989. *Inside Case-Based Reasoning*. Hillsdale, NJ: Lawrence Erlbaum.

Salton, G., & McGill, M. 1983. *Introduction to modern information retrieval*. New York: McGraw-Hill.

Schneiderman, B. 1994. Dynamic Queries: for visual information seeking. *IEEE Software* 11(6): 70-77.

Shimazu, H., Kitano, H. & Shibata, A. 1993. Retrieving Cases from Relational Data-Bases: Another Stride Towards Corporate-Wide Case-Base Systems. In *Proceedings of the 1993 International Joint Conference on Artificial Intelligence*, pp. 909-914.