A Case-Based Approach to Knowledge Navigation

Kristian J. Hammond
Robin Burke
Kathryn Schmitt
Artificial Intelligence Laboratory
The University of Chicago
1100 East 58th Street Chicago, IL 60637

Case-based reasoning and software agents

The AI Lab at Chicago has begun development of a new set of software agents designed to manage the flood of data colloquially called the "information superhighway". Our approach takes its lead from case-based technology [1,2] in that we are building systems that emphasize the use of examples over explicit queries or questions for communicating with the user.

We propose to develop three sorts of systems: browsing systems (called FIND-ME systems), preference-based task organizers (called BUTLERS), and Internet news group agents (called CORRESPONDENTS). All three types of agents are designed to help a user navigate through an information space and either find or construct responses to fit the user's needs.

Two features distinguish these systems. The first is that they are derived from the case-based ideas of using *retrieve* and *adapt* as the core problem solving model. The second is that they use existing archives and data-bases as resources to be "mined" on demand rather than as fodder for batch processors that learn new concepts or construct new knowledge bases independently of a user. This too draws from the case-based philosophy of waiting until a problem arises to solve it.

Our proposed efforts will investigate all three sorts of software agents. The focus of this proposal, however, will be on the FIND-ME systems and CORRESPONDENTS, the most developed of our projects.

Find-me Agents

The problem

The FIND-ME systems are designed to allow a user to navigate through a set of possible solutions or products that fit their needs. The class of problems addressed by the FIND-ME systems is best explained through an example:

> You want to rent a video. In particular, you'd like something like *Back to the Future* which you've seen and liked. How do you go about finding something?
>
> Do you want to see *Back to the Future II*? Do you want to see another Michael J. Fox movie? Do you want to see *Crocodile Dundee*, another movie about a person dropped

into an unfamiliar setting? *Time After Time*, another time travel film? *Who Framed Roger Rabbit?*, another movie by the same director?

The goal of the FIND-ME project is to develop systems that deal with this sort of search problem. These problems relate to domains with the following features:

- There are many choices. (Approximately 20,000 in the video domain.)
- New elements cannot be generated by the system or the user. (The system cannot recommend a movie that doesn't yet exist.)
- The space is defined by a vocabulary of features that may not be accessible to the user. (How many people would know to describe *Back to the Future* as an instance of a "stranger in a strange land" theme?)
- The user discovers new examples in the course of his or her search. (New movies that the user hasn't seen are presented.)
- The user discovers the features that define the domain in the course of his or her search. (Figuring out that "buddy" films exist.)
- While they may not be able to articulate their own absolute constraints, users are able to comment on examples. (You may not know exactly how funny you want a movie to be, but you know that *To Kill a Mockingbird* doesn't match your preferences.)
- Finally, person-to-person interaction within the domain takes the form of trading examples. (e.g., "If you liked *Back to the Future*, you're sure to like *Crocodile Dundee*.")

Many complex selection problems have these characteristics, for example, personnel selection. It is difficult to specify completely what kind of person is right for a job, but it is easier to look at a person's resume and come up with a response such as "Give me some one like this, but with more leadership experience."

In order to cope with these task features, we take the approach in FIND-ME of always presenting examples and allowing the user to suggest changes, which then are used to retrieve further examples. The user never has to articulate exactly what he or she wants, but only has to comment on what is right (or wrong) with an example that they are looking at. Users move through the space by looking at examples and communicating their likes and dislikes to the system. The system responds by adjusting its search strategy. By working from examples we avoid the need for the user to have extensive prior knowledge about the domain in order to find what he or she wants in a database.

The central ideas

The FIND-ME projects are driven by three main concerns: **interface metaphors**, **reasoning through examples**, and the support of **non-hierarchical search**.

One of the core ideas of the FIND-ME systems is to develop user interfaces that are metaphorically linked to known artifacts. Just as the Macintosh environment makes use of the desk-top metaphor, we have constructed interfaces that are analogs to existing artifacts that aid search within a domain. This allows users to have strong predictions about the effects of their own actions, reducing the explanatory burden on the systems. The overall goal is to provide users with a sense of where they are in the domain space and how they got there.

The aim is to protect the user from the fact that he or she is searching through an multi-dimensional feature space. Instead we present the user with a familiar artifact, such as a magazine or video store aisle, that has the added feature of being dynamic. This means that users can browse through an environment that stays stable with respect to what they have already seen, but is dynamic in that the next items are always those suggestions that the system believes the user wants.

The second major insight that we bring to knowledge navigation is that we can use the differences between a presented example and a user's target to formulate a description of the target itself which is then used to access a knowledge base of further examples, an iterative process of exploring the space of examples. Rather than using the absolute features that define the space, we are far more concerned with giving users the ability to say, in essence, "Yes, but..." and change a feature with respect to an example.

Finally, we have designed the overall FIND-ME notion with an eye to avoiding hierarchical search of a space. We don't see the selection process as one of narrowing. Instead we see it as moving through a space of possibilities where any feature can be changed at any time. The danger in a truly open-ended search in a large, multi-dimensional space is that the user can easily become lost. To assist the user's search, FIND-ME systems have active *clerks* that constantly view a user's current preferences and suggest alternative tracks through the space that reflect the clerk's unique goal organization and retrieval strategies. For example, in our CAR NAVIGATOR, each class of cars has its own clerk, that attempts to change the user's focus when a match within its class exceeds a fitness threshold. This allows the user to break away from any tunnel vision that they might have because of either ignoring a feature in the space or simply not knowing how that feature has been constraining the search.

The Car Navigator

The most mature of our systems is the CAR NAVIGATOR, a FIND-ME system that allows users to explore the domain of new cars. The interface to the CAR NAVIGATOR is designed to resemble a car magazine. The initial page (See Figure 1) is table of contents, showing pictures of seven cars, each labeled by class (e.g., small, compact, midsize, large). There is an icon representing each of these classes next to the picture, and the same icons are used along the edges of the virtual magazine as tabs into the various sections of the magazine. The user can examine cars in any car class by clicking the tab with the icon of that class.

To start, the user selects a car class by clicking the picture of the car representing the desired car class. The page turns and the 5 most typical examples of that car class are displayed on the left page (See Figure 2). On the right page the picture of the top-most car from the left side is repeated and underneath it is a collection of icons and controls. Each icon represents an attribute of a car, such as horsepower, price, or gas mileage.

A list of desired attributes representing the system's current understanding of the user's preferences is displayed to the right of the control. The user gets a feel for how well the displayed car matches his current preferences according to the color background of each feature. This color pattern is echoed in the color tabs on the cars remaining on the left page. The user can adjust his

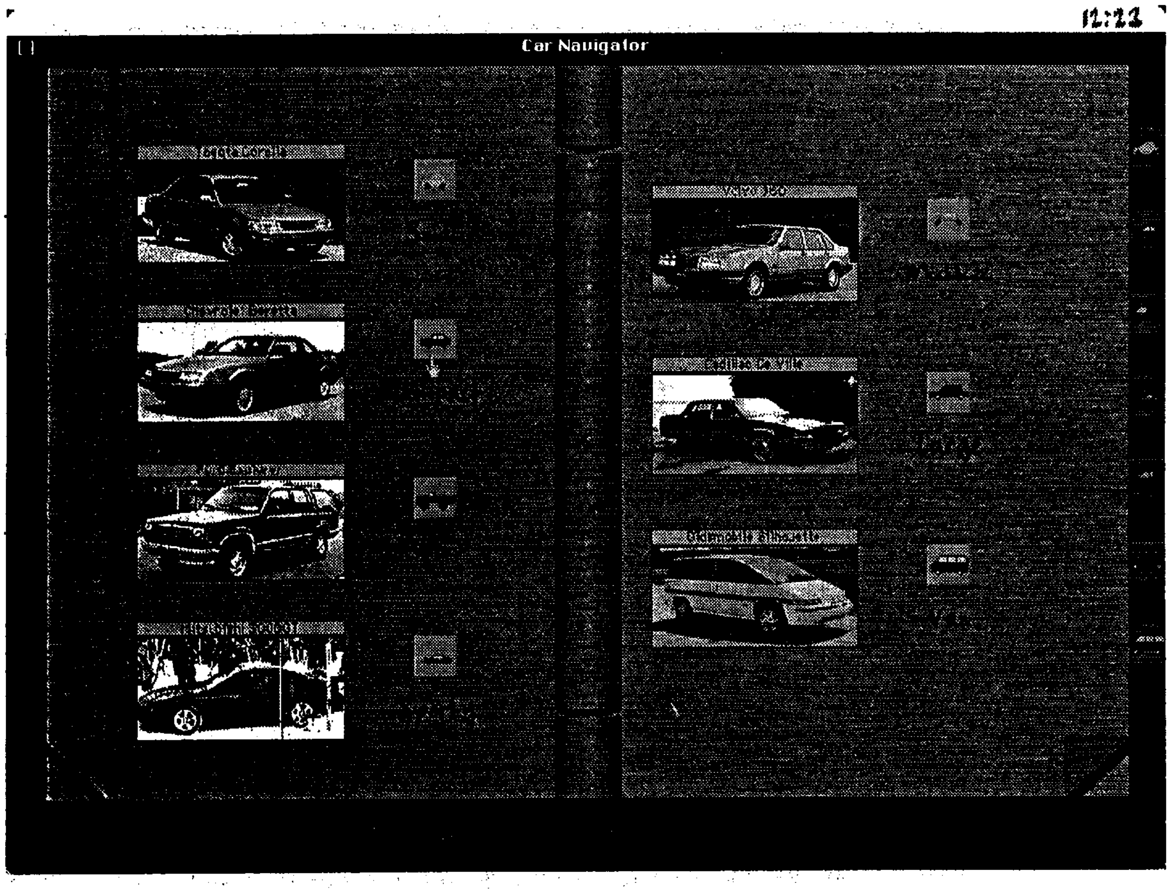


Figure 1: The first page of the "Car Navigator Magazine".

preferences at will and the resulting changes in degree of match are immediately reflected in the color fields.

Often a user requests an attribute value that cannot be attained without altering one or more of the desired values for other attributes. This results in a "yellow light" coming on in the features linked to the one that has been changed, followed by a Quicktime movie containing an explanation. For example, if a user requests good gas mileage and then requests high horsepower the yellow light will come on next to the gas mileage and horsepower features. When this happens, a car salesman explains (via a Quicktime movie) that there is a trade-off between horsepower and gas mileage. The user will have to decrease either the desired level of fuel economy or the amount of horsepower he is requesting.

When users have set the preferences to their liking, they request the retrieval of a new set of cars by clicking the folded corner at the bottom of the right hand page, "turning the page" to reveal a new set of 5 cars, those that best fit the new preferences. Users' preferences may change enough to move them into a different car class. If a better match is found in a car class other than the current one, CAR NAVIGATOR signals this fact with a yellow exclamation mark displayed over the tab with the icon representing the car class with the better match. Users can turn to this section to see the proposed alternative.

The ability to make small changes in the search criteria and see corresponding examples is

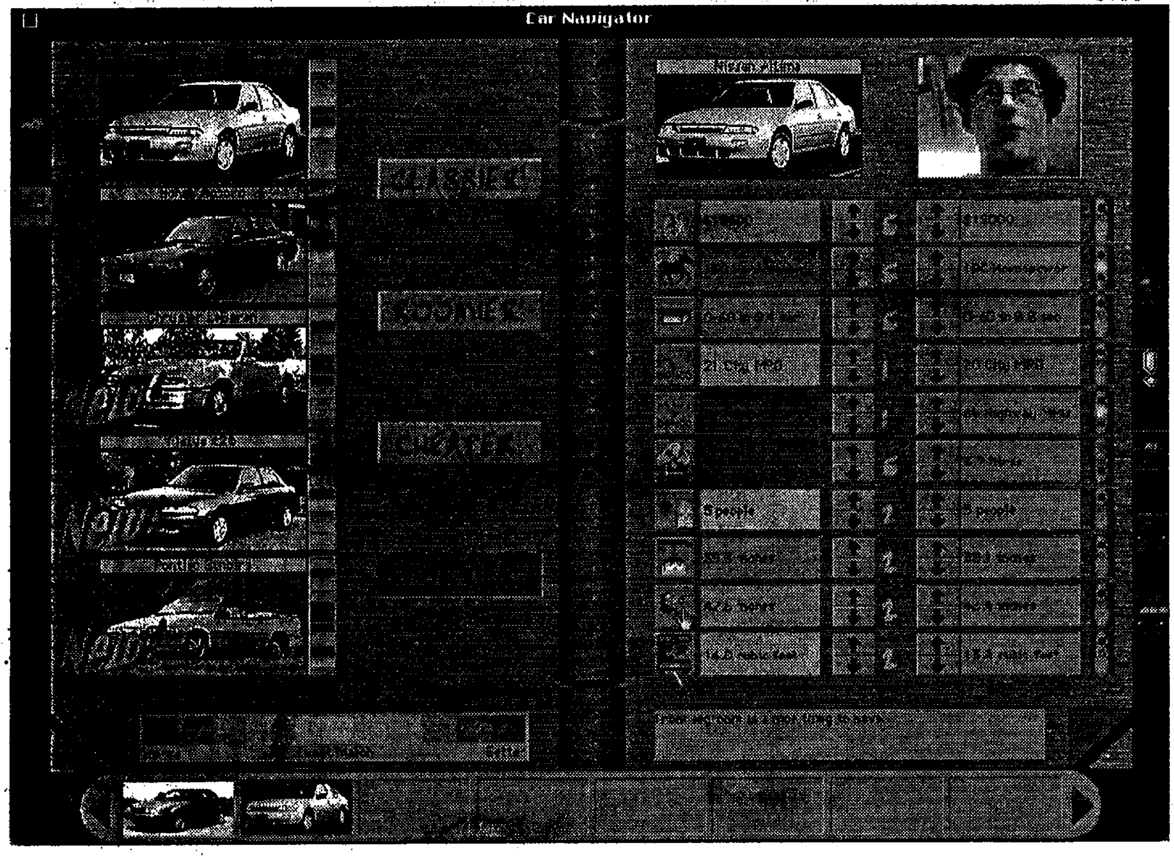


Figure 2: Browsing within the Car Navigator

useful in exploring local areas of the information space, but it does not work as well for making large jumps. If the user wants a car that is "sportier" than the one he is currently examining, this implies a number of changes to the feature set: larger engine, quicker acceleration, and a willingness to pay more, for example. For the most common such search strategies, CAR NAVIGATOR supplies four "big buttons," located on the left-hand page: *sportier*, *roomier*, *cheaper*, and *classier*. Each button modifies the entire set of search criteria in one step, pushing the search in the direction that the user has indicated.

CAR NAVIGATOR uses a weighted-sum similarity metric for comparing cars. The values along each feature are grouped into qualitative classes: very quick acceleration, quick acceleration, etc. which are related in a semantic network. For each feature of each car, the distance between the user's desired feature and the corresponding one for the vehicle is computed, multiplied by the weight given to that feature in the search criteria, and summed. Preference is given to cars in the same class as the current example but the individual clerks light up class markers when a match is found in a different class.

While simple, the CAR NAVIGATOR approach would be useful for searching any space of examples defined by technical criteria, for example, in procurement. The user need only have general knowledge about the set of items and only an informal knowledge of his needs; he can rely on the system to know about the tradeoffs (yellow lights), category boundaries (red lights), and useful search strategies (big buttons).

Video Navigator

We have used our experience in building the CAR NAVIGATOR to begin work on a VIDEO NAVIGATOR that draws from a database on the order of 20,000 movies. In this domain, it is not possible for the system to have an internal representation of the entire database for manipulation. The VIDEO NAVIGATOR, therefore, is an intermediary between the user and a large movie database. Like the CAR NAVIGATOR, the VIDEO NAVIGATOR starts from a core of examples and the user communicates by noting differences. However, the features of movies are not as simple as the largely-numeric values manipulated in the CAR NAVIGATOR.

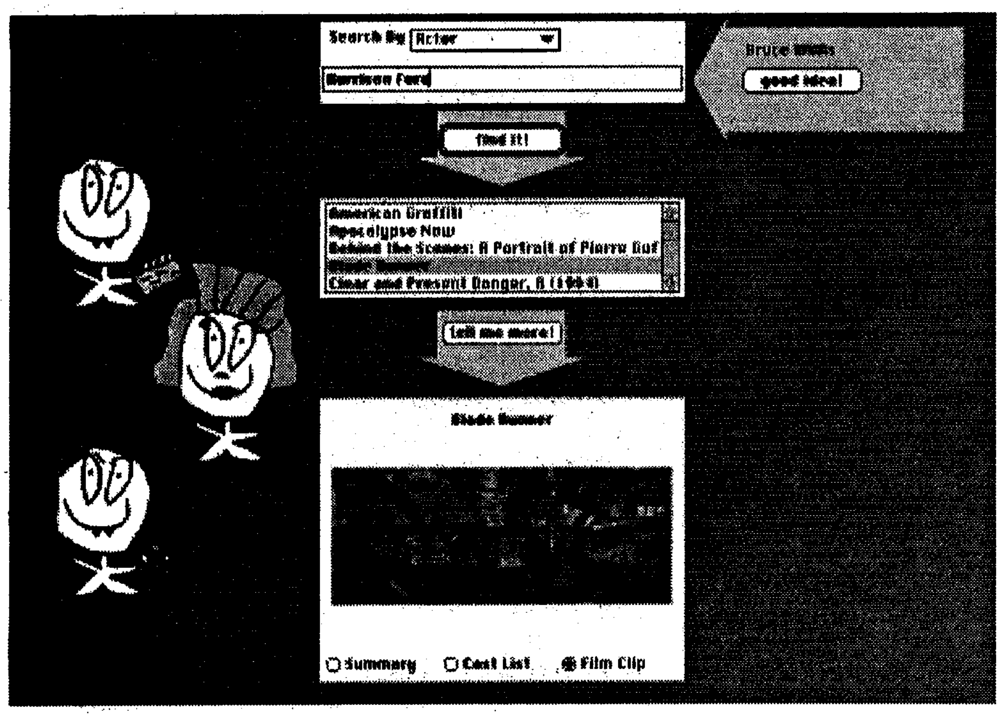


Figure 3: The Video Navigator and its various clerks.

In the VIDEO NAVIGATOR we are continuing to develop the idea of *clerks* (see Figure 3). The clerks in VIDEO NAVIGATOR each have specialized knowledge about different aspects of movies and specialized retrieval strategies for finding appropriate movies. For example, the clerk that specializes in knowledge of actors has a taxonomy of actor types and several strategies for finding similar actors, such as comparing actor types or looking for actors who have been in similar kinds of movies.

The user can browse in the film database by looking for standard search terms: actors, directors, movie titles. However, the clerks are continually monitoring the user's actions and making suggestions. For example, in Figure 3, the actor clerk has observed the user's search for movies containing the actor Arnold Schwarzenegger and suggested another action hero: Bruce Willis. The user can take this suggestion (by clicking on the "Good idea!" button) or can ignore it. The clerk's suggestions help the user avoid a narrow search path; they bring in examples of movies that are relevant to the user's search, but that the user would not necessarily encounter if left unassisted.

The development of FIND-ME technology exemplified by VIDEO NAVIGATOR applies to a class of problems where selection must be made among examples whose specifications are complex and not fully specified. It would be difficult for a computer to look at a person's resume and decide if they have "sufficient leadership experience" for a particular task, any more than the computer can guess about what movie will be sufficiently funny, but the person doing the browsing can easily make such decisions. It is the job of various clerks to use their knowledge of the domain to take the user's feedback and search for suitable examples.

Browsing and Case-based Reasoning

We feel that browsing is an important application of case-based reasoning. People prefer to move about in information spaces in ways that are consonant with their own understanding of the domain [3]. Standard database-style interfaces to large libraries are only acceptable when users are sufficiently knowledgeable to know the literal information recorded in the database: exact names, titles or numbers.

Most people's understanding of cars and movies (and other complex domains) is not so literal. We believe therefore that a browsing system should always make available a "reasonable next place to go," given where the user is now. In the CAR NAVIGATOR, the next place to go is provided by the ability to turn the page of magazine, and get a new collection of cars organized around one's current preferences. In the VIDEO NAVIGATOR, because the feature of movies are complex and difficult to make explicit, we provide a host of clerks, each of which tries to find a reasonable next step for the user, based on its specialized knowledge.

Further, there is an interesting relationship between the "tweaking" in browsing and the notion of adaptation in standard CBR. In CBR systems that do adaptation, the modifications are done by the system in response to a failure or a gap between the goals of the user and the goals satisfied by some plan that th program has retrieved. The result is the creation of a new plan that was not previously in memory. In the sort of browsing systems that we are developing, the same sort of adaptation takes place, but the final product is not a new plan (or new movie, car, etc) but is instead a new prototype that the system can now use to construct indices into memory for an existing plan (or movie, car, etc.). The basic flow of *retrieve* and *adapt* is still in place, but their role is altered by the presence of the user and the fixed nature of the knowledge base.

Butlers

The second type of agent, BUTLERS, are designed to be experts at specific tasks that can be done using on-line information, such as figuring out where to get your muffler repaired, or what hotel to stay in, or where to have a business dinner. The idea is to build specific agents for each task and have them do the work of looking things up in on-line data bases. BUTLERS are information intermediaries that are even more active than FIND-ME systems; they actively accumulate information relevant to their tasks from multiple sources.

For example, we propose to develop a restaurant BUTLER that uses knowledge of your personal tastes, where you last ate, who you are eating with, and what sort of dinner it is to search for a place that fits your desires, schedule, and budget. The BUTLER will use information from on-line guides such as the ZAGAT restaurant guide, its own city maps, and your personal schedule to figure out the time and place that suits your needs. In the end, we intend for the BUTLER to hand feedback

back to the restaurant and even broadcast favorable or negative reviews to other BUTLERS, with the idea that this information can then be used by them to make better choices.

Our first step in developing this system will be to create a restaurant BUTLER that is accessed through a personal digital assistant (such as Apple's NEWTON). The hand-held device will communicate with a central server, which in turn searches on-line information sources. Like the FIND-ME systems, much of the selection process will be based on the idea of the differences between current examples and the user's wants, a gentle way of coaxing out a user's preferences rather than forcing the user to compile an explicit query for every search.

Correspondents

In the early days of the Internet, news groups were simply public areas into which users could post messages. They have matured with the addition of moderators, archives, moderated data-bases, and frequently-asked-questions files. Our goal in building CORRESPONDENTS is to continue this development by adding intelligent agents that draw from this accumulated experience recorded by users.

At the core of correspondents are case-based retrieval systems that are like fully-automated FIND-ME systems. CORRESPONDENTs read posted requests or problem descriptions, construct queries with which to search a case-library of possible answers or solutions, and post the answers they find.

FAQFinder

The first CORRESPONDENT we propose to develop is FAQFINDER, an automated question-answering system that uses the files of "Frequently-Asked Questions" (FAQs) associated with many USENET newsgroups. These files are compendiums of the accumulated wisdom of the newsgroup on topics that are of frequent interest. FAQFINDER will take a user's query on any topic, attempts to find the FAQ file most likely to yield an answer, searches within that file for similar questions, and returns the given answers.

The technology we intend to use to develop FAQFINDER is fairly simple (see Figure 4) Given a user's question, the system selects the appropriate FAQ file, using a statistical IR package [5] applied to both the FAQ files and the compiled word listings from the new postings themselves.

Once an appropriate FAQ file has been identified, the matching of the user's question against the file is done using a parser that compares question templates and canonical semantic elements. The goal here is not to parse into an unambiguous meaning, but instead to parse into a rough representation that can be used to support matching. If a match is found, the matching question/answer pair is presented to the user. At every stage of this process, the user is given some control over disambiguation and selection of the final "answer." If no answer is found, the question is handed to a FAQ manager that posts the question to the newsgroup and incorporates the resulting answer(s) into the FAQ file.

While the FAQFINDER program is in some sense a browser, our actual goal is to free users from doing any browsing at all. Rather than forcing users to traverse the knowledge space, we are providing active agents that will do this for them. A similar approach can be applied to the querying of other large textual information sources, such as procedure and documentation manuals.

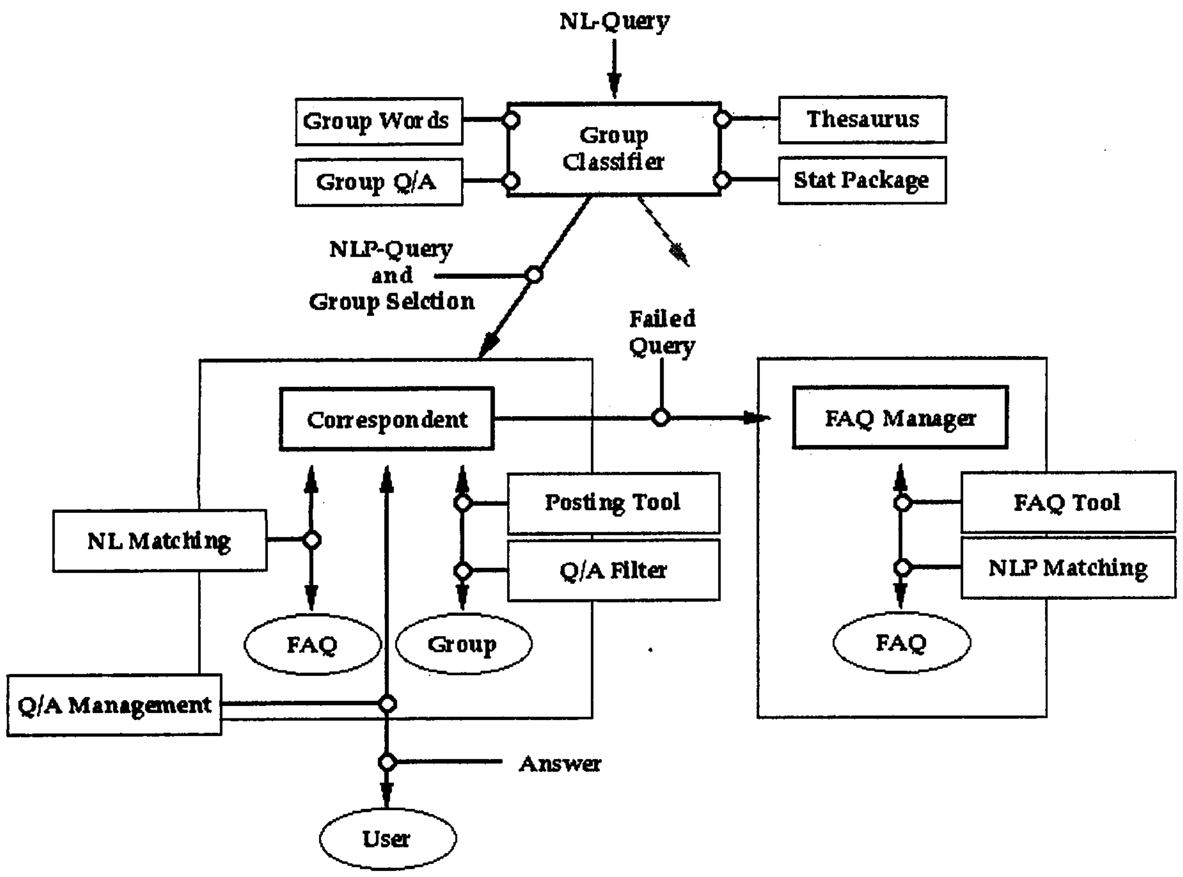


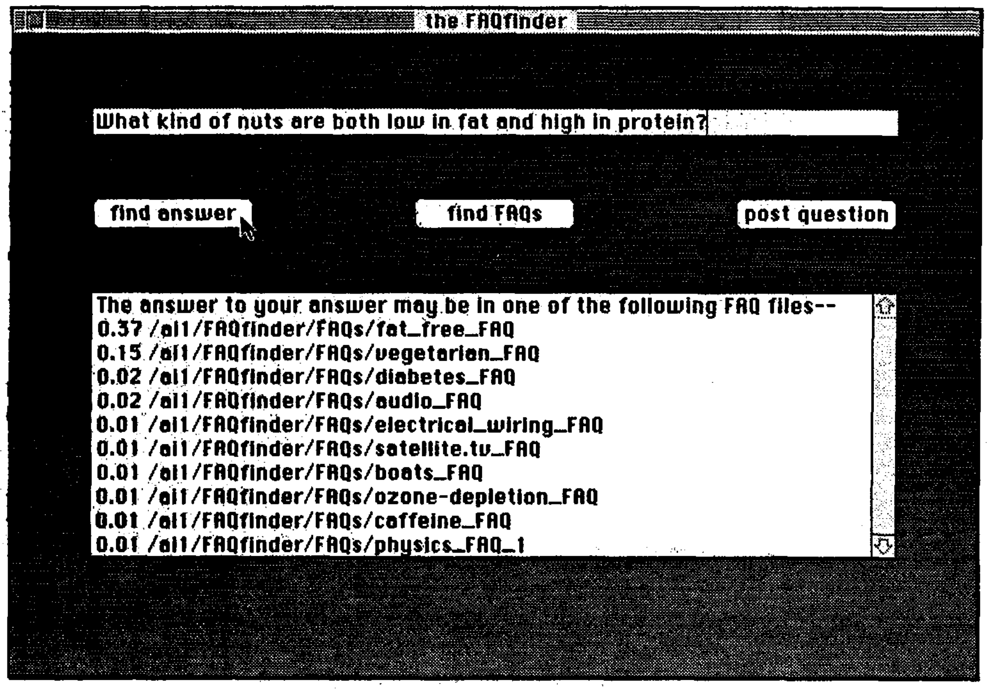
Figure 4: The organization of FAQFINDER

The FAQFINDER project is also interesting in that we intend to use not just the existing archives on the InterNet, but also the existing sociology. One of the more powerful aspects of the new-groups is the desire on the collective part of the users to "get it right." This drive has already resulted in the FAQ file themselves, and we plan to extend this concept to the development of augmented FAQ files that extend beyond the range of the "most" frequently asked questions.

Cyber Chef

We also propose to develop a system called CYBER CHEF that is designed to read requests posted to the `rec.food.recipes` news group and respond with recipes based on the news group's own archive. CYBER CHEF is an extension of Hammond's original work on CHEF [4], a case-based program which retrieved and modified recipes from a small case library to satisfy requests for particular dishes. CYBER CHEF will perform these same functions, but with a greatly expanded case library taken from various on-line recipe archives, and with input requests in natural language taken from the Usenet newsgroup `rec.food.recipes`.

The use of the case library in CYBER CHEF will differ somewhat from the original CHEF. The old CHEF's library was analogous to a real chef's personal memory of recipes that are known by heart; CYBER CHEF's case library is analogous to a shelf filled with cookbooks, which the chef is familiar with and knows how to find things in, but does not have completely memorized. Thus, in CHEF, recipes were stored in program-interpretable memory structures representing the



What kind of nuts are both low in fat and high in protein?

```
The answer to your answer may be in one of the following FAQ files--
0.37 /ai1/FAQfinder/FAQs/fat_free_FAQ
0.15 /ai1/FAQfinder/FAQs/vegetarian_FAQ
0.02 /ai1/FAQfinder/FAQs/diabetes_FAQ
0.02 /ai1/FAQfinder/FAQs/audio_FAQ
0.01 /ai1/FAQfinder/FAQs/electrical_wiring_FAQ
0.01 /ai1/FAQfinder/FAQs/satellite.tv_FAQ
0.01 /ai1/FAQfinder/FAQs/boats_FAQ
0.01 /ai1/FAQfinder/FAQs/ozone-depletion_FAQ
0.01 /ai1/FAQfinder/FAQs/caffeine_FAQ
0.01 /ai1/FAQfinder/FAQs/physics_FAQ_1
```

Figure 5: The FAQFinder interface.

ingredients and steps of the recipe, whereas CYBER CHEF's recipes are stored as raw text, annotated with program-interpretable indices. The text of the recipe will be interpreted and represented bit-by-bit when the recipe needs to be modified (as if the chef, having found the appropriate cookbook, were looking over the recipe to see what needed to be changed).

Ultimately, CYBER CHEF is intended to pull recipe requests from the newsgroup, retrieve and possibly modify recipes from the case library to satisfy the requests, and post the results back to the newsgroup. We intend the program to make use of email sent by newsgroup readers in response to the recipes–both as feedback to replace the original CHEF program's simulator feedback, and as additions to some of indices under which the recipe is stored.

Our current implementation deals with a scaled-down recipe library containing only recipes for sauces. It is capable of retrieving, but not modifying, recipes from this library, in response to natural-language requests. The program does not yet read or post to the newsgroup, although the input requests currently used for testing consist of the text of request postings from `rec.food.recipes`.

Like all of these systems, CYBER CHEF is making use of the core ideas of case-based reasoning in communication, case selection, and plan modification. The insight here is that examples serve to make problems concrete and give the user and the system a common context to refer to.

Conclusion

Our work to date with CAR NAVIGATOR and preliminary implementations of VIDEO NAVIGATOR leads us to believe that this approach is a natural way to help users search in complex and poorly-structured domains. It works best when users can easily evaluate examples, but have difficulty articulating the features they are interested in.

We propose to move from small, self-contained databases (as in CAR NAVIGATOR), to large unstructured collections, such as the submissions to USENET newsgroups. This scaling-up process involves some significant new challenges, such as the matching of natural language questions in FAQFINDER and the adaptation of only-partly understood recipes in CYBER CHEF.

References

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

[2] Hammond, K., and Colleen Seifert. A Cognitive Science Approach To Case-Based Planning. In S. Chipman and A. L. Meyrowitz (Eds.), Foundations of Knowledge Acquisition: Cognitive Models of Complex Learning (pp. 245-267). Norwell, MA: Kluwer Academic Publishers. 1993.

[3] Ferguson, W., Ray Bareiss, Lawrence Birnbaum, and Richard Osgood. ASK Systems: An Approach to the Realization of Story-based Teachers (Technical Report No. 22). Evanston, IL: Institute for the Learning Sciences. 1992.

[4] Hammond, K. CHEF: A Model of Case-based Planning. In *Proceedings of the 1986 National Conference on Artificial Intelligence.* Philadelphia, PA. August 1986.

[5] C. Buckley, Implementation of the SMART Information Retrieval Retrieval [sic] System. Technical Report 85-686, Cornell University. 1985.