Agents to Assist in Finding Help

by

Adriana Santarosa Vivacqua

Bachelor of Science in Computer Science

Pontificia Universidade Catolica, RJ (1993)

Submitted to the Program in Media Arts and Sciences, School of Architecture and Planning, in partial fulfillment of the requirements for the degree of Master of Science in Media Arts and Sciences at the Massachusetts Institute of Technology

September, 1999

Signature of Author ______________________________ [illegible]
Adriana S. Vivacqua
Program in Media Arts and Sciences
August 6, 1999

Certified by ______________________________ [illegible]
Patricia Maes
Associate Professor of Media Technology
Media Arts and Sciences
Massachusetts Institute of Technology

Accepted by ______________________________ [illegible]
Stephen A. Benton
Professor of Media Arts and Sciences
Chair, Departmental Committee on Graduate Students
Program in Media Arts and Sciences

Agents to Assist in Finding Help

Adriana S. Vivacqua

Submitted to the Program in Media Arts and Sciences, School of Architecture and Planning, on August 6, 1999, in partial fulfillment of the requirements for the degree of Master of Science in Media Arts and Sciences.

Abstract

The problem of finding someone who might be able to help with a particular task or knowledge area exists everywhere, be it in groups of students or corporate settings. Time and effort are spent looking for relevant information when another person in the community could easily provide assistance. Our approach to addressing this problem is to use software agents to assist the search for expertise. Previous research on this topic has been mostly in the areas of information marketplaces, referral systems or information repositories. We built a system, called Expert Finder, which provides a testbed for ideas and techniques developed in the context of this thesis. Expert Finder analyzes previous work of both the novice and the expert to automatically categorize expertise and match it with the user's problem while providing a community-based incentive mechanism. We present alternative profiling and incentive mechanisms to those that had been presented in previous work. We chose the Java Programming domain for our initial implementation and testing of the system. Expert Finder uses each user's Java source files to determine their expertise and uses a Java domain model to match questions and experts. It keeps track of users' willingness to help out as a measure of the social capital in the community. We ran some tests using the prototype system to assess how well these ideas worked, and results are also reported in this thesis.

Thesis Supervisor: Patricia Maes
Title: Associate Professor of Media Technology

Agents to Assist in Finding Help

Adriana S. Vivacqua

Thesis Readers:

Research Advisor [illegible]

Henry Lieberman
Research Scientist
MIT Media Laboratory

Academic Advisor [illegible]

Patricia Maes
Associate Professor of Media Technology
Media Arts and Sciences
Massachusetts Institute of Technology

Thesis Reader ______

Mark Ackerman
Associate Professor - Computing, Organizations, Policy, and Society (CORPS)
Information and Computer Science
University of California, Irvine

Thesis Reader ______

Walter Bender
Senior Research Scientist
Media Arts and Sciences
Massachusetts Institute of Technology

Acknowledgements

I am grateful to my advisors, Henry Lieberman and Pattie Maes, for the opportunity to work on this project. Their guidance, insight, comments and revisions were invaluable for this thesis. They patiently listened to and thought about my ideas and kept me on track when I might have strayed from the topic. Henry's scenarios and suggestions were very helpful in defining the system. Pattie's comments and questions were invaluable as well and kept me focussed but thinking about other possibilities.

I also wish to thank my readers, Mark Ackerman and Walter Bender, for their commends and suggestions. Their interesting questions made me think harder about my work and try to address issues I otherwise wouldn't have. They were very responsive and able to turn the thesis around in very little time, which I appreciate.

I am also indebted to my test subjects, who not only volunteered their code but also responded to my questionnaires in record time. Without you, this thesis would not have been complete. I should also thank Barry Crabtree and BT, whose interest and support made this work possible.

This section would be incomplete if I did not thank my friends here and abroad. Conversations with other Agent's students helped define this thesis. Nelson Minar, Bradley Rhodes, Neil Van Dyke, Giorgos Zacharia and Michael Ernst put up with my darkest moods and helped me through the final stages of the process. Early discussions with Brian Smith (back when he had a window office) inspired and helped shape some of this work.

Bananas@media, Fernanda and Claudio, were oft-needed anchors to home. Jen, Tanzeem, Sunil and Doug were great friends who made me feel welcome here. The Lindyfolk, especially the MIT lindyers: Ben, Rif, Amanda, Jen and Brad (it's all his fault), who kept me dancing and reminded me there was life outside the lab. My friends back home, whose frequent email not only kept me aware of the news but also brought a little bit of home to my everyday life (even if often in the form of nonsensical banter which only we could understand and appreciate.) My parents and brother for their support and love. This one is for grandma.

Table of Contents

Table of Figures

Table of Tables

Chapter I: Introduction

The problem of finding assistance is one that presents itself very often in daily life. It is quite common for a person to find him or herself in need of help even when performing routine tasks. The more specific the needs, the harder it is to find someone who can help.

This problem exists in several different domains and forms. For instance, in a classroom setting, a student might have difficulty with a given problem set. Someone who has taken the same course before would be a potential source for help when the professor or teaching assistant is unavailable. Fellow students are also a good source of help. Alternatively, consider someone who needs to obtain knowledge in an area other than his or her own, such as a librarian using Excel for the first time: it would be helpful to have someone who knows about Excel assist the librarian whenever needed (and she doesn't necessarily need tech support, she might need some simple advice.) In many cases, a person might end up doing a lot of research in books and help files, when it would have been more productive to talk to someone with the right expertise who could help solve the problem.

Talking to someone else is, in fact, a common solution for this kind of problem. A person asks his or her friends or co-workers, who are believed to have the necessary knowledge to help solve the problem. If these don't know, they usually provide a referral to someone else they believe might be helpful. Navigating this social network, one may get to a local expert.

However, often there is no expert close at hand, which makes the task harder. The use of computer technology could greatly help this process: recent advances in communication and networking technologies have made it possible to link people across great distances, thus bringing non-local experts, who would otherwise not be able to help, closer. If we extend the search to encompass a larger, networked community, more experts will become available. However, it is necessary to have good communication mechanisms in place to support this information exchange. Information can be hard to find. In fact, Kautz notes that "Part of the value of a piece of information resides in the degree to which it is not easily accessible" [Kautz, Selman & Milewski, 96]. Making information more accessible is one of our goals.

Motivated by the frequency with which this problem occurs, we decided to tackle it. The Expert Finder was created to try to address these issues. It is an interesting one which branches off into several areas of knowledge, involving issues like user profiling and determining what is expertise; matching needs and experts and incentives for information exchange. We looked into several areas of knowledge for inspiration and previous work that might help guide our own.

The Expert Finder is an agent that compiles a user's expertise profile from his or her personal files, and makes it available for other agents to see. When users have problems, they can query the Expert Finder and it will look for an appropriate expert. By matching questions asked with experts able to answer them, we seek to link people with questions to people with answers. Expert Finder also keeps track of users' responsiveness to the needs of the community in an effort to encourage collaboration. We have implemented the Expert Finder for the Java programming domain and used real user's files to test its performance.

There are many systems that are somehow related to our system, especially in the CSCW (Computer Supported Collaborative Work) domain. To our knowledge, few have used Software Agent Technology and even less concern themselves with building communities and fostering collaboration. We try to find ways to create groups that exchange knowledge over a network, emphasizing the community aspect.

Similar projects in this area range from agent-based systems to information repositories to task-based expert systems. Current state of the art involves little work on incentive mechanisms, though. In general, these systems rely on existing links (social networks or hierarchical structures) or money (creating a marketplace) to motivate people into collaboration. In our system, we try to provide an explicit non-monetary incentive mechanism for motivating people. We also explore the use of personal files as a source of information about the users, something that hasn't been explored so far.

In this document we describe the Expert Finder project, from its inception to its current status. We start by describing the problem we're addressing in Chapter II and provide a

Scenario in Chapter III. We then describe our approach, architecture, lessons learned and future work. We finish with an analysis of related work that has been done in the area.

Chapter II: The Problem

The problem addressed in this thesis is that of finding an expert who can assist with a problem. People often need assistance and the easiest way to get that is by asking someone they know. When they don't know anybody, they turn to an "official expert", someone who works with this subject on a day-to-day basis (as in hiring a lawyer or going to a doctor.) Sometimes, however, going to the formal channels is not necessary, as the advice needed may be simple and available from a local friend. Then again, sometimes going to the local (informal) expert is not always the best option, since he or she may not know the answer and choose to give an incorrect one (or guess) instead of admitting ignorance. Issues of incentive go hand in hand with the choice of expert: one would expect to pay for a doctor but not for a friend's advice. This problem is obviously a complex one and involves several issues, some of which we address in this thesis. Imagine the following scenarios:

1. Kerrin is a new Microsoft Word user. She has to write a lengthy report,

which includes some complicated calculations done on Excel. She doesn't know how to include the calculations in the document. She tries the on-line help system, but still can't quite figure out how to do it. She then turns to her co-workers, asking someone who has been using the Office suite longer than she has, who helps her solve her problem.

2. Diana is learning to speak Spanish. She still has difficulty understanding some of the terms and grammar of the language, but needs to read a letter in Spanish and write a response. She looks for but cannot find any native Spanish speakers. She comments on it to Claudio, a friend who happens to be a native Portuguese speaker. He asks to take a look at the letter, and can indeed understand most of it even though he has no formal education in Spanish. He helps construct a simple response (a harder task, given that he doesn't know the Spanish language) and calls upon another friend (who knows the language), who then corrects the letter to Diana's satisfaction so she can send it.

3. Jen has to write a client server Java program. She has little experience with Java, although she knows what she has to do. She needs to connect a back-end SQL database to a server side program. She knows how to build queries on the database, but not exactly how to link them to the database. She asks a friend's daughter, who refers her to David. Jen talks to David and he helps her find suitable Java objects for her needs.

4. Marcos decides to try a new recipe from his grandmother's cookbook. He

has the recipe, but not much experience making ethnic food. He wants to make sure he has suitable substitutes, given that he couldn't find some of the ingredients. He also wants to check the timing for each step (and figure out how to make sure that the food is ready.) He doesn't want to call his grandmother or mother, given that they are coming for dinner and he wants to surprise them. Instead, he calls a friend who has some experience with ethnic food (but not that same dish) and he helps him with substitutions and hints for the cooking.

The previous examples illustrate common situations in our daily lives. When in need of help, people often turn to others they know (usually, people found in their social networks) for assistance. However, sometimes people need help and don't know how to get it, for no one in their social circle has the specific knowledge needed. Besides, social networks may become hard to navigate: the larger the network, the harder it is to find an appropriate person. Many sources of information seem to be available, but each has its own set of associated problems:

1. *Manuals and books*: manuals have become huge tomes of information (more often than not, several books), often times poorly indexed, making it hard for the user to find exactly what he or she wants. Many times the information is spread among several heavy volumes, which people don't want to carry around. Besides, there is often a lack of examples (or the existing ones are incredibly simple and not really applicable) and the information is static, not being flexible enough for any but the most simple needs. Most people don't

even bother reading manuals or books anymore. Printed documentation has become almost obsolete.

2. *Electronic Help Systems*: help systems are usually nothing but hyperlinked versions of the printed manuals, so they exhibit all the same content-related problems their printed versions have. They are, however, slightly better, for they have search mechanisms to assist in the location of information. However, very often these are inefficient: searches yield several hits, which the user has to go through looking for the needed information (which might even not be there.) To get better results, it often necessary to know certain keywords (which will yield the appropriate hits) or some form of search syntax, to enable more complex combinations of the search terms. However, more often than not, the lay person doesn't know the magic keywords or appropriate syntax. Navigating the hyperlinks can also be confusing, as it is not always obvious which link to follow to get more information on a topic, and users get lost jumping from one link to another.

3. *The Web*: searching the web for information is an even harder task than searching help systems. Electronic help systems are a finite, closed repository of data. Since the web is an open repository of information, the number of hits for a given search is much bigger than for help systems (even though not all the web is covered.) One might end up with a large number of useless webpages and have to spend time sifting through them (not always with good results.) The web also suffers from the "magic keyword and

complex syntax" problem, with the added difficulty that there are now several different search engines online, and each has its own syntax. On the plus side, being an open repository also means that more information will be available, possibly more examples and different explanations for the same problem, which means that a user is bound to find something suitable. However, these are not always from sanctioned sources (as was the information given in the previous two cases), which means that the information provided by a certain website may not be reliable, or that the user must have some way of assessing how trustworthy a site is.

4. *Usenet and Mailing Lists*: the user can also post questions on Usenet or on mailing lists, but a novice might get no response, as groups formed are very specific and often assume some level of expertise on the subject - even finding the correct group can be hard. One other factor in mailing lists is that a person might assume that the question has already been answered in private (especially if he or she considers it a trivial question) and refrain from answering, so that the questioner doesn't get many answers to the same question. Another problem is getting wrong or misleading answers: when a question is posted to a generic forum, sometimes people who respond may not actually be giving an adequate response (oftentimes unknowingly.) Once again, it is hard to judge how trustworthy the information being provided is, since there is no history or rating attached to the person. It may also be difficult to break into an established community and get them to help.

5. *FAQs*: FAQs (Frequently Asked Questions) are compilations of questions frequently asked on mailing lists or Usenet groups and their respective answers. They are put together by the group to avoid endless repetition of common questions. These are certainly a powerful resource, as they represent common problems and questions people have, and the best answers to them given by a group of people (which makes it more likely that the answer will be correct.) However, each FAQ refers to a specific subject, so if the problem a user has doesn't fit one of the FAQs, no answer will be found. Obviously, if the question is an unusual one (being more specific or complex than the norm), it is not likely to be found in a FAQ. Other than that, there is the problem of finding the correct FAQ to look at, since there are many FAQs circulating on the internet (which also creates a problem of version control: how does one make sure to consult the most recently updated FAQ?) The problems of searching and finding information in a FAQ file was addressed by Burke, with a project called FAQ Finder [Burke, 97]. It was built to leverage information from FAQs: given a natural language question, it searches a database of FAQs for a similar question and retrieves the corresponding answer. However, as noted before, one might want something that is not in a FAQ file, or have specific questions that aren't asked often enough to be included in a FAQ.

6. *Social Networks*: most of the time, people end up asking a friend or colleague who might know the answer and often this friend refers them to someone else. These are what we call social networks. Through navigation

> of these social networks, a person can find valuable resources. There are a few problems, though: social networks work best for the closest links, that is, people one or two (maybe even three sometimes) hops away. The friend of a friend will be inclined to help you because there is a direct connection between the two. The further one gets, the more indirect and impersonal the relationship, until one gets to a point where it is the same as asking a stranger. Furthermore, for a very unusual or specific question, there may not be anyone in the close vicinity. When there is someone, the two parties might have problems explaining their problems and solutions, because novices and experts have different mental models for the same domain, as noted by Feltovich [Feltovich, Ford & Hoffman, 97] and Ericsson [Ericsson & Charness, 97]. One other problem is that referral chains can be hard to navigate, and a chain might come to an end before an expert is reached [Kautz, Selman. & Shah, 97b].

Many of these problems have also been noted by Kautz [Kautz, Selman & Milewski, 96] and others when building their systems. Time is spent looking for information or help, and little is gained. Often, people don't even know where to look. Given that the usual solution is to ask the local expert, we come across another problem, which makes the task harder: often there is no expert close at hand. Extending the search to a larger, networked community, more experts become available, even if there's no physical proximity. However, the introduction of large numbers of people with no direct connection raises the importance of the incentive issue, since people are less inclined to assist someone they don't know than to help a friend or colleague.

The major issues addressed in this thesis are:

1. *Building Expertise Profiles*: how does one build expertise profiles (as opposed to interest profiles)? Which sources of information are most valuable? Are the same ones being used for profile building (generally email files and webpages or browsing patterns) well suited for expertise also? What must these sources of information reflect? What should they relate to?

2. *Finding an Appropriate Expert*: who's the best expert to help? Is it always the topmost expert or are there situations when someone at a lower level of expertise could also help? What if nobody with the given expertise is found? Is it possible to recommend someone even then? Based on what? If there's no specialized expert, who else would be able to answer the question? How is that determined? Where does this information come from?

3. *Motivating the Experts*: how do we get people to collaborate? What mechanisms need to be in place to facilitate information exchange? Are monetary incentives necessary? Is there another way of forming new communities and taking advantage of existing ones? How do we reinforce the existing community and take advantage of it? How do we create new communities?

In the next chapters, we delve into each of these in more depth. We have constrained our domain to that of Java programming, but are trying to keep an open mind regarding possible domains. This thesis also opens up opportunities for future research in how to mediate novice-expert interactions, how to build context-sensitive agents (agents that could determine when the user needs help by keeping track of their actions) and collaborative ontology building tools.

Chapter III: A Scenario

In this section we present a scenario of usage for the Expert Finder system. We first present the two main characters, Jen and David, and set up a context. Then, we present the two ways in which the same problem could be dealt with, with and without Expert Finder.

III.I Meet Jen and David

Meet Jen: Jen is an accomplished COBOL programmer. She has been in the computer business for a while, doing systems analysis, contract programming and consulting for various companies. She has wide experience in database programming and large centralized systems. She has just gotten a new job, working for an important consulting company. This company wants to move into new technologies and decides to build their new applications in Java, even though it has little corporate experience in this language. Jen's new project will be in Java.

Meet David: David is a hacker. He started programming at the age of 15, and enjoys learning new technologies: he has been playing with Java for a while now. He has worked with user interfaces, computer graphics and client-server systems at one time or another. He now works as a systems programmer for a large software company, which does most of their work in Java.

Jen's new project is a client-server system for a bank: clients of the bank will download software that will communicate with the bank over secure connections, and perform transactions through their computers. The system uses encryption, database manipulation and a graphical user interface.

Given that Jen is a novice Java programmer and a former COBOL programmer, she has a hard time understanding the concepts of an object-oriented language and learning all the existing packages and classes. She breezes through the database part, though, building all the server-side SQL routines without much trouble. Her problems start with the database connection to the program…

III.II The hard way

Jen knows she needs to connect her server side routines and database with the front end. However, she doesn't know what objects are available and which of those to choose. She asks around the office, but nobody is familiar enough with the Java language to help her make smart decisions and navigate JDBC objects and connections. She manages to get some help with setting up drivers on her machine so she can at least access the database. She then defines the functionality that should be included in the front end and goes about researching how it should be done.

She turns to the JDK documentation but is unable to find much information on this new library. She tries to build some of the structures, but finds that it's a tedious and slow process, as she must test the objects she might be interested in using. She pokes around on the Internet and, lurking in some of the user groups, finds out that there are some books on JDBC which might help her. She quickly chooses one and buys it. It gives her some very basic notions, but not nearly enough to help her build her application. She needs more details on how to call the server-side stored procedures she has created.

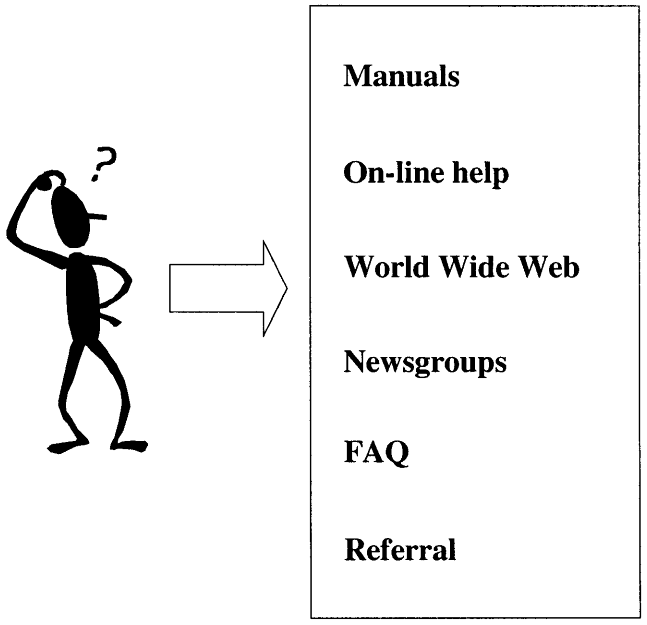


Figure 1: Without the Expert Finder, the user must go through several different channels and sources looking for the information.

She wades around the many different newsgroups until she finds some that show promise of at least being on topic. She reads their FAQs before posting a question, to make sure she wasn't posting something that had already been addressed. Disappointingly, she gets no answers. Following the conversations for a few days, she ends up deciding that most of them are useless to her, with tight communities of a certain skill level, where people tend to get off topic or carried away by one particular issue. Persistent, she subscribes to a few mailing lists. Most of them are high traffic and she ends up with too many messages in her mailbox. Even the digest versions of these lists are long and poorly organized, so she can't find anything in it. Besides, people seem to be more interested in discussing their own problems than addressing the problems of a new user like her.

She finally decides to get in touch with a friend's daughter, Sarah, who studies Computer Science at the local university. Sarah has never programmed in Java, but knows several more advanced students who have. In fact, Sarah's boyfriend, David, is something of a whiz, who has done much Java programming and works for a high-tech company. Jen reluctantly sends him an email, to which David replies with a brief explanation and pointers to some websites about JDBC. He also tells her to look up the *prepareCall* method and the *CallableStatement* class, which may be what she's looking for.

III.III Enter the Expert Finder

Jen knows she needs to connect her program to the database. However, she doesn't know what objects are available and which to choose to call the stored procedures she and others have created. Instead of asking around the office, though, Jen goes to her Expert Finder agent and enter a few keywords, which describe what she wants to know.

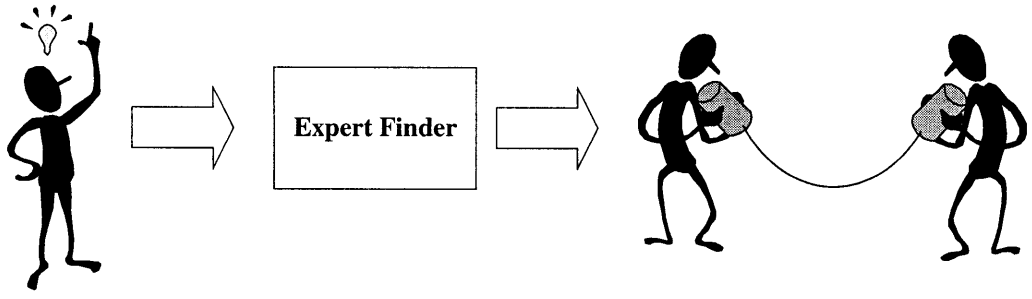


Figure 2: Information flow with the Expert Finder: the user goes to the EF and it recommends somebody who may be able to help.

Jen has a close relation with her agent: it periodically reads through her Java source files, so it knows how much she knows about certain Java concepts and classes. In fact, it reads through all of her "hello, world" programs and other programs she wrote while studying with the "Learn Java in 21 Days" [Lemay & Perkins, 97] book. Expert Finder verifies what constructs she has used, how often and how extensively, and compares those values to the overall usage (usage levels for the rest of the participating community) to establish her areas and levels of expertise. It "knows" she has written very little Java code and has used each of the constructs only a few times. Jen checks her profile (she can see and edit her profile on the profile-editing window) and decides to publish all of it, allowing others to see all she knows (and doesn't know.) Table 1 shows Jen's usage for each construct and calculated profile.

Jen types in the keywords "sql", "stored" and "procedure". From the domain model, the agent knows that sql is related to database manipulation – java.sql is a library of objects for database manipulation. From the model, the agent knows which classes are included in this library.

Area	Usage	Expertise Level
java.io	10	Novice
java.util	15	Novice
java.sql	1	Novice
java.awt	8	Novice
System	20	Novice
Integer	10	Novice
Vector	10	Novice
elementAt	5	Novice
println	20	Novice

Table 1: Jen's areas and levels of expertise

The agent then takes this query and communicates with other agents, getting their users' expertise profiles. It calculates other users' "suitability" by looking at their profiles, verifying which libraries and classes they know how to use. It picks out David (whose profile can be seen in Table 2), because he has used the "java.sql" library and its objects and is at a level of knowledge above but not too distant from Jen's. Given his previous work experience and coding samples, the agent knows that David is knowledgeable about database manipulation.

Jen takes a look at David's published profile, checks his "halo factor" (an indicator of how helpful he is to the community), and decides to send him a message asking about accessing stored procedures from her Java program. Her message reads:

```
Dear David,

I'm a novice Java programmer and have some problems regarding
database connections and manipulation.  I have created a series of
stored procedures and now need to access them from my program.  Is
```

```
there a way to do that?  Or do I have to re-code all my SQL
statements in my java programs and call them?  Any help you could
give me would be greatly appreciated.

Thanks,

Jen
```

Area	Usage	Expertise Level
java.io	46	Intermediate
java.util	45	Intermediate
java.sql	8	Intermediate
javax.servlet	33	Advanced
java.awt	7	Novice
Connection	11	Advanced
InputStream	5	Intermediate
CallableStatement	10	Intermediate
Println	140	Advanced
GetConnection	2	Intermediate
PrepareCall	10	Intermediate

Table 2: David's areas and levels of expertise. Note that the levels of expertise are obtained through a comparison with others in the community.

David receives the message, verifies based on Jen's "halo factor" that Jen is a new user and decides to answer her question. He sends her a reply, explaining that she should try *prepareCall* and *CallableStatement* and pointing to some online resources which would be valuable to her. His message reads:

```
Hi Jen,

To call stored procedures you should use a CallableStatement, which
```

can be created with the prepareCall method of the Connection class. You have to use the registerOutParameter and get<data_type> methods to retrieve the return values.

Here's a little snippet from one of my code files, which might help you:

```
CallableStatement cstmt = con.prepareCall("{call MyProc(?, ?)}");

cstmt.registerOutParameter(1, java.sql.Types.TINYINT);

cstmt.registerOutParameter(2, java.sql.Types.DECIMAL, 3);

cstmt.executeQuery();

byte x = cstmt.getByte(1);

java.math.BigDecimal n = cstmt.getBigDecimal(2, 3);
```

Take a look at:
http://java.sun.com/products/jdk/1.2/docs/guide/jdbc/getstart/callablestatement.doc.html

Good luck, hope this helps.

David

In this fashion, Jen found an expert and obtained his help much faster than she would without the use of Expert Finder.

Chapter IV: Assisting the Help-Seeking Process

When a person doesn't know how to solve a problem, he or she usually looks for someone in their social network that has the necessary knowledge (an expert) and asks them. However, it can be hard to find an expert in a particular field who is at the right level of expertise and willing to help. Thus, we set out to study systems that could get people in touch with each other and assist information exchange. Note that we are not interested in creating expert agents that could answer users' questions. Rather, we want to create agents that will bring people together, since interaction and reflection are important parts of the learning experience. Also, as Ackerman points out, dealing with people (i.e., asking the expert directly) has the advantage that the expert can be more responsive and assist in formulating the question, and that, when provided by authoritative experts, answers are more reliable that those provided by a random colleague down the hall.

IV.I Approach

Agents, as defined by Maes [Maes, 93], are programs that act as personal assistants who collaborate with the user when performing hard tasks. Agents have been built that can filter email, schedule meetings, match people, recommend webpages, etc. Some of these have learning capabilities, some watch over the user's shoulder and some take direct input. In general, agents become personalized to one particular user, having learned about his/her preferences.

Given the aforementioned problems and scenarios, we have created an agent-based environment, where each person has an agent that is able to find other agents, whose users are able to help the first person. We want this system to encourage collaboration over the network, thus allowing a community to leverage knowledge existing inside people's heads.

We decided to create agents to assist the help seeking process and mediate expert-novice interactions. Agents can automate the profile-building process and decentralize the system, making sure that each user has his/her own personal profile. They can also match a user's needs to other people's expertise and mediate peer-to-peer communication when necessary. Some other functions the agents could perform in this context are: scheduling appointments for knowledge exchange between two users; proactively detecting when help is needed; providing extra information during interactions; negotiating knowledge exchanges (when using an economic incentive model); or adjusting its own profiling mechanisms according to user feedback.

Each agent's goal is *to determine the best expert in the existing community to help its* *user*. Each user has an agent that has knowledge about his or her levels of expertise, extracted from information generated by the users themselves. Upon request, the agent searches the community for other users who might be able to help, based on their profiles. The agents exchange information about their users and figure out which user is a good source of help. The agent itself does not provide answers to problems, it directs people to other people who can do so.

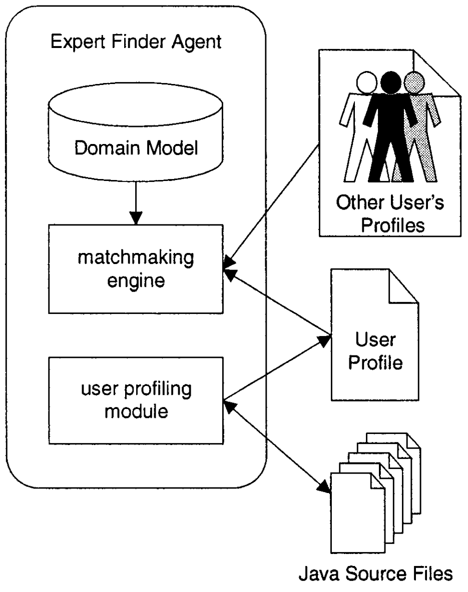


Figure 3: An agent's Internals: Each agent has (1) a profiling module, which builds the user's profile from his/her Java files; (2) a matchmaking engine, which consults and compares other user's profiles and (3) a domain similarity model, used for matchmaking purposes

Figure 3 shows one agent's internal structure. In our system, each agent is able to compile users' profiles, communicate with other agents (analyzing their users' profiles) and decide which other users are more suitable to help its user (given their profiles). It is important to note that all agents are the same, i.e., there are no specialized agents for experts and novices. We believe these roles to be interchangeable according to the situation (a person might be an expert in one area and a novice in another), in the style of Evard's temporary experts [Evard, 97]: she built a system where students in a classroom could ask or answer questions, becoming "temporary experts" according to the situation.

IV.II Domain Similarity Model

Our system uses a domain model for purposes of determining the experts for a given situation. We call this a similarity model because it maps the relations between two nodes, determining how similar they are. The main reason to use such a model is to be able to find somebody who has related knowledge if an expert on that field is not available. In these situations, the system suggests a person who has related knowledge and may be able to help, (since it couldn't find anyone with the specific knowledge needed.)

In this model, each node is a knowledge area (a possible area of expertise) and each relation is weighted, determining the degree of similarity between two areas. These areas may be completely disjoint, such as mathematics and history (a historian won't necessarily be able to help with non-trivial math problems, since there is very little math involved in his or her area of expertise, history) or somewhat related (physics and math: a physicist uses mathematics extensively and might be able to answer some math questions) or strongly related (biology and genetics: geneticists have a generic biology background and are thus able to answer biology inquiries in addition to more specific genetics ones.)

Expanding the biology domain, there are several different specialties: molecular biology, marine biology, genetics and malacology (a branch of zoölogy dealing with mollusks) are just a few of these. In this particular example, molecular biology, marine biology, genetics and malacology are subfields of the more generic biology area. Any specialist in one of those areas will have taken a set of core biology courses, and will therefore have a certain basic knowledge of the field. Experts in any of these fields may be able to answer easier or generic questions on biology, since there are commonalities among them. However, more specific questions may or may not be answerable by experts in other fields. Marine biology and malacology share some similarities. Depending on the level of specialization of the expert and the questions, he or she may know enough about another field to be able to answer questions. The domain similarity model maps these relationships, aiding in connecting different areas and enabling recommendations to be made when someone has related knowledge.

This model would be built using existing conventions and relations between areas and subareas of knowledge. For the medical domain, one could take the minimum curriculum necessary to establish common knowledge to all medical specialties. One might then verify the different curricular demands for each specialty to determine additional knowledge and identify similarities between subfields. Many different fields of knowledge have already been mapped in some form or another, it is only a question of mapping those into a similarity model to be used by the expert finder. Ideally, such a model would be easily extensible and correctable, making the expert finder more flexible and domain-independent. For this thesis, we chose Java programming as our test domain. We delve into the Java domain in the next section.

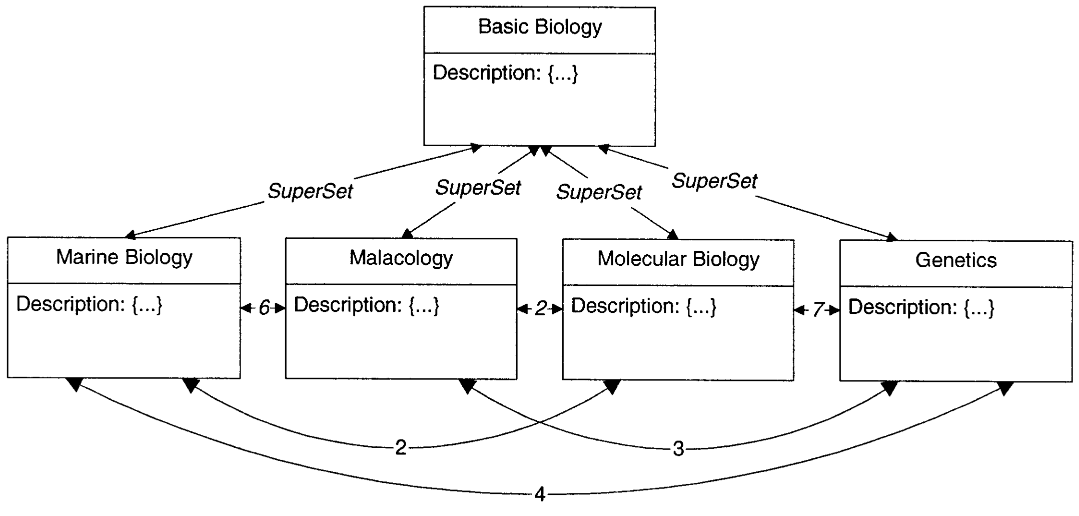


Figure 4: Sample similarity model, for the Biology domain. Marine biology, Malacology, Molecular biology and Genetics are subareas of Biology, and experts in this area will have some knowledge of more generic concepts as well. Similarity between specific areas varies.

IV.II.I The Java Programming Domain

Java is a new programming language. It is constantly changing as new libraries, classes and functionality are being added. It takes time to learn enough about its constructs to be able to take full advantage of them. Because many people are using it, there is a wide range of expertise levels in the community. Some people are coming in with experience in other programming languages, others understand the object paradigm better, some have no experience at all, and some have been programming Java for a while now. It allows for specialization, in that one person might be used to building graphics systems and another database systems. It is a reasonably complex domain and allows us to build a useful tool for actual programmers to use and test Expert Finder.

In the present case, the Java language is highly structured and this structure is described in the documentation, which can be found on the web [Sun, 98]. Being an object-oriented language, the constructs are hierarchically structured into classes and subclasses and organized in different packages (according to their purpose or usage). Furthermore, many classes contain similar methods and the documentation provides an extra hint: the "See also:" entry, which lists related classes, methods or packages for each class.

To build the Similarity Model, we mapped the similarities from the Java language structure by assigning arbitrary values to each of the relationships between classes. The first step in the process was establishing which items would be taken into account for purposes of determining similarity. We used three different items:

1. Sub/Superclass relationships: a subclass is fairly similar to its superclass (since it inherits its methods and properties), however a superclass is less similar to its subclass, since the latter is more specialized and may contain several different resources not available in the original class. For example, the class Container is a subclass of class Component: it inherits several methods (131) and fields (5) from class Component, which means that it bears considerable resemblance to that class (in fact, anyone who knows how to use the inherited methods will also know how to use the ones in the parent class). However, it also defines a whole set of new methods (52), which set it apart from its parent class (that is, people who know how to use the parent class may not know how to use these methods defined in the Container class.) Code: SUB or SUP.

2. Package coincidence: someone grouped the classes according to their functionality

and usage. Therefore, it seems only fair that this grouping also be used to determine similarity. These classes are not necessarily so similar in how they work, but in what they are used for. For instance, package java.awt contains classes used for graphic interface construction, such as buttons, listboxes, drop-down menus, radio boxes, check boxes, etc. Even though these might work differently, a person who knows how to use these classes in general is someone who knows how to build graphical interfaces. Code PAK.

3. "See also" entry: this is a completely arbitrary entry, a hint given by Sun engineers that the items following this list are somehow related to the one being viewed. This might be because of how they work (which opens up another possibility: looking at classes with similar methods that are not inherited from a base class) or what they are used for. Class MenuBar, for instance, is a subclass of class MenuComponent, and is related to classes Frame, Menu and MenuItem through the "See Also" relationship. Code: SEE.

Thus, the documentation pages were parsed into a domain model where one class' similarity to another is determined by {SUB, SUP} + PAK + SEE, where the values for each of the variables may vary according to the type of query (free-form keyword based or selected from list.) These values are highly arbitrary, however, they are parameterized: the model holds the different relations, not the numbers. It would be very easy to change the values assigned to each of the relations, thus changing the similarity values. We realize that packing these into one number is sub-optimal, given that we are dealing with different types of relations: instead, the different types of similarities should be used on the match and displayed to the user so that the he or she can make a more informed choice.

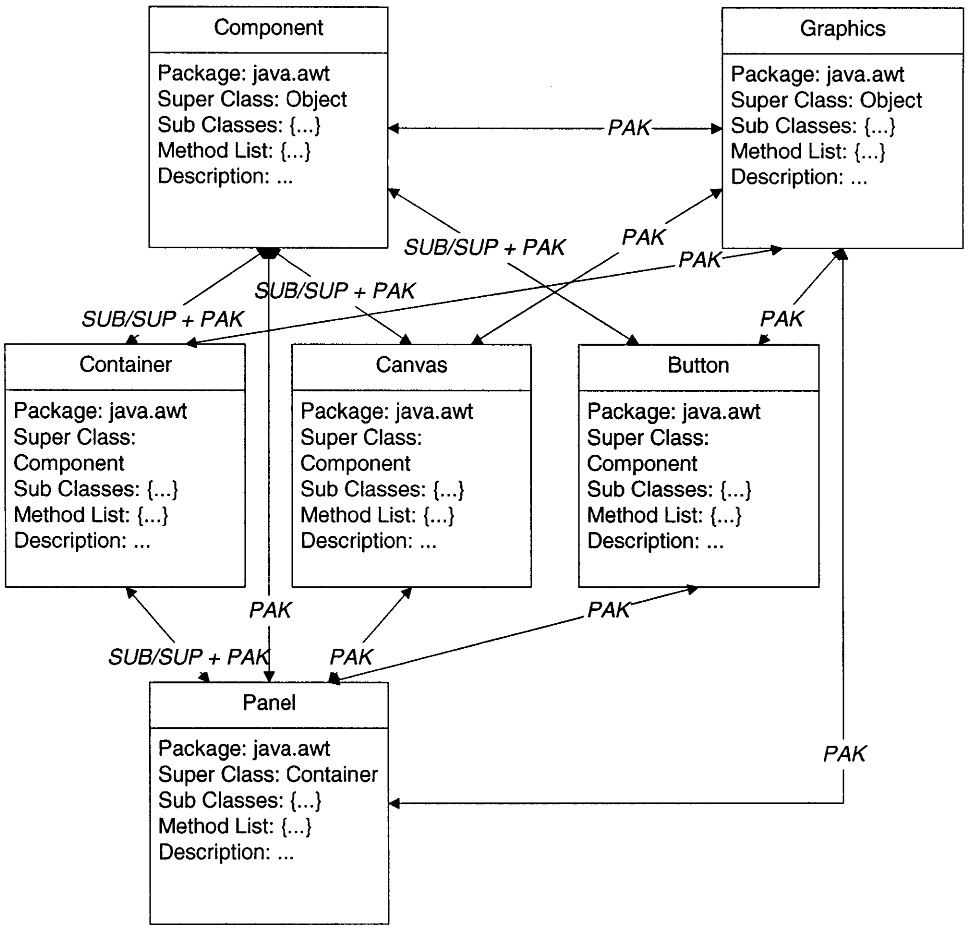
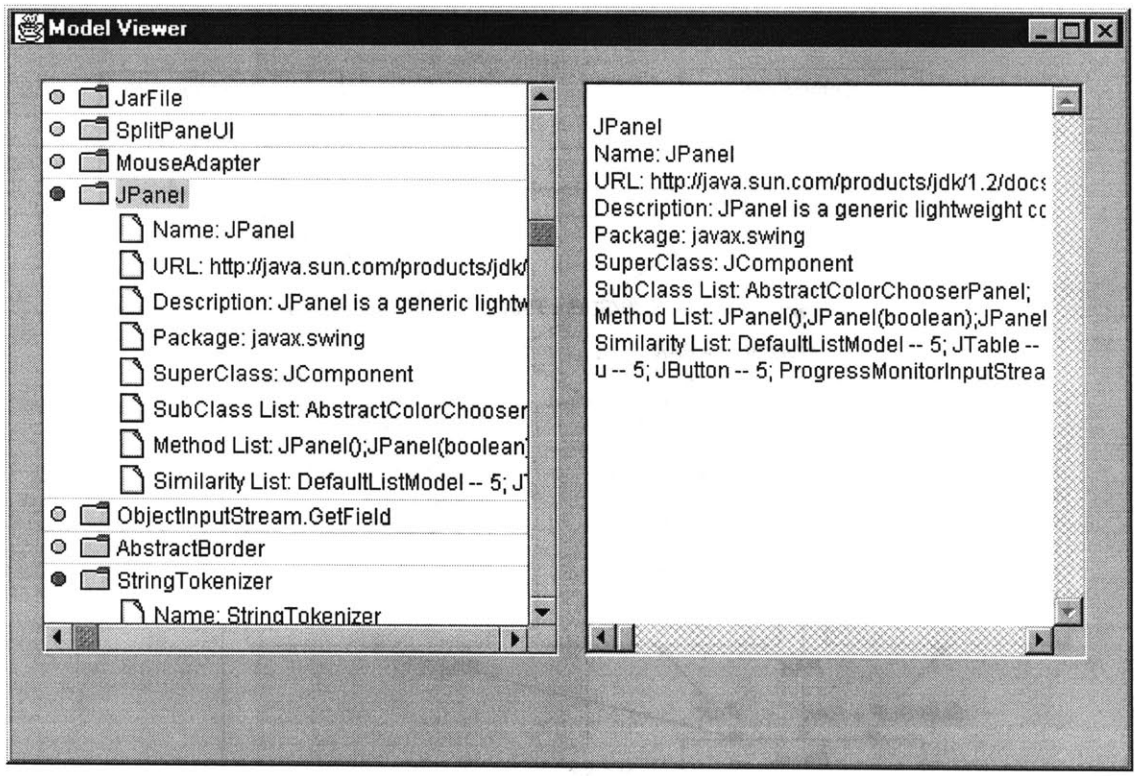


Figure 5: Similarity model for the Java domain (partially shown.)

A byproduct of this work was a domain model viewer, which allows a user to see the domain model, its nodes, classes, description and relations to other nodes. This makes it possible to visualize the relationships between classes and perhaps better understand the rationale behind the match. It could easily be altered to become an editor, allowing the users to make adjustments to the model, fine-tuning the system so that it will come up with better matches in the future. Extending this application to allow editing would also be the first step towards a collaborative ontology editing tool.



```
○ JarFile
○ SplitPaneUI
○ MouseAdapter
● JPanel
      Name: JPanel
      URL: http://java.sun.com/products/jdk/
      Description: JPanel is a generic lightw
      Package: javax.swing
      SuperClass: JComponent
      SubClass List: AbstractColorChooser
      Method List: JPanel();JPanel(boolean)
      Similarity List: DefaultListModel -- 5; J
○ ObjectInputStream.GetField
○ AbstractBorder
● StringTokenizer
      Name: StringTokenizer
```

```
JPanel
Name: JPanel
URL: http://java.sun.com/products/jdk/1.2/docs
Description: JPanel is a generic lightweight cc
Package: javax.swing
SuperClass: JComponent
SubClass List: AbstractColorChooserPanel;
Method List: JPanel();JPanel(boolean);JPanel
Similarity List: DefaultListModel -- 5; JTable --
u -- 5; JButton -- 5; ProgressMonitorInputStrea
```

Figure 6: Model Viewer screenshot: this program allows a user to view the model being used by the system.

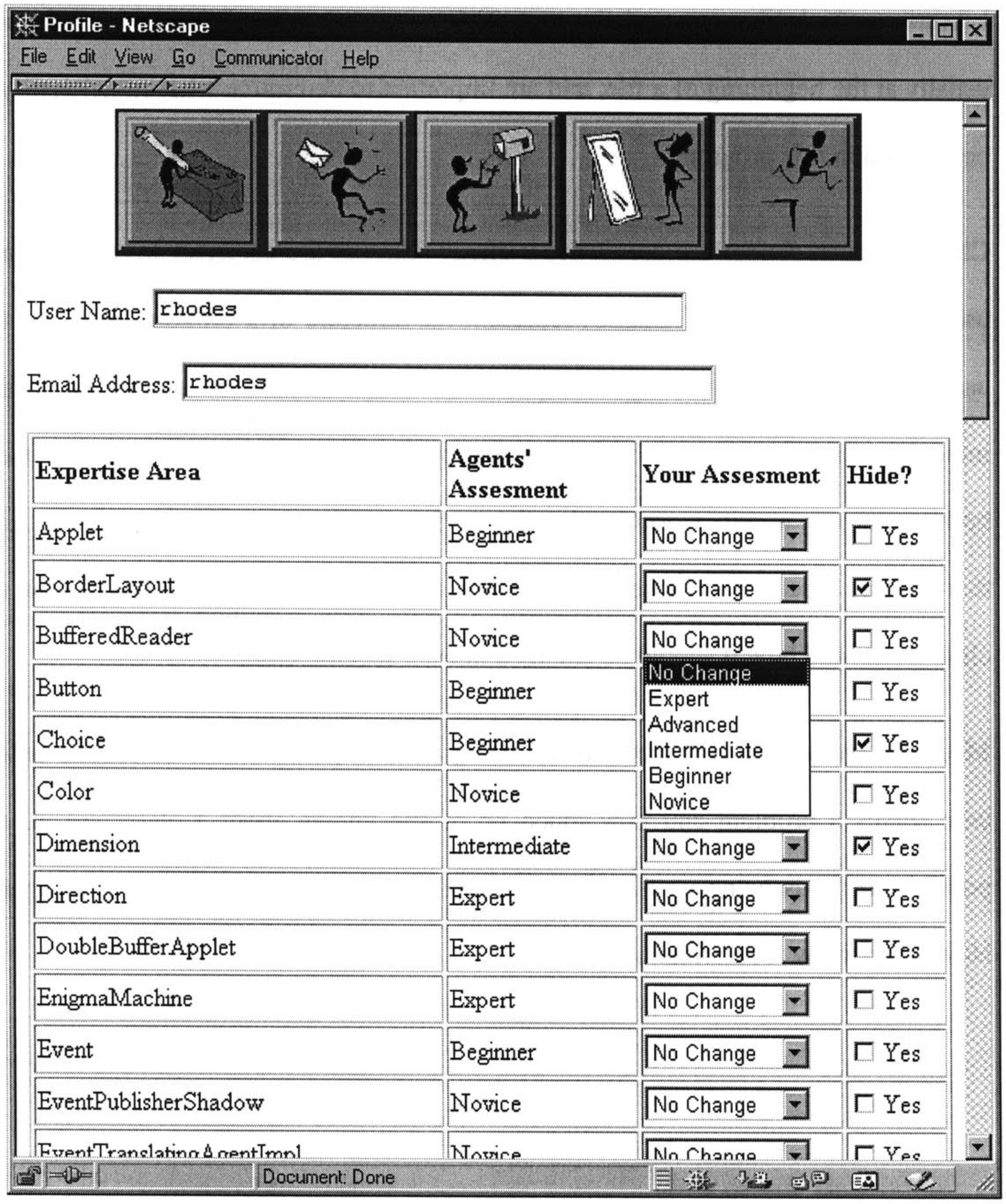
In this case, the hardest part is building the model itself. Once that has been done, the model viewer can be extended to display it with a few alterations to its code. This is certainly not the ideal display for the type of model proposed, since it would be preferable to have a graph display and a "detail window" where users could view each node's internal structure and its links. The present Model Viewer gives the impression that the model is hierarchical, when it is not.

IV.III Building Profiles

An important part of the system is having profiles that accurately reflect what the users know and what their levels of expertise are. Each user's areas and levels of expertise are stored in a personal profile created by the agent and made available upon request to other users (provided the user had given permission for the agent to do so). It is important to have some sort of automatic profiling in place, given that, in general, people dislike filling long forms about their skills. Furthermore, an automated method results in a smaller possibility of inaccuracy due to people's opinions of themselves (reflected by either rating themselves too high or too low on the scale.) These profiles are recalculated periodically, so as to keep them up to date. However, we acknowledge the fact that the agent might be wrong in its assessment and allow the user the option of altering his or her profile.

A profile contains a list of the user's areas of expertise, the levels of expertise for each area (ranging from novice - beginner - intermediate - advanced - expert) and a flag noting whether or not this information is to be disclosed. The agent will not disclose any information the user has marked as "non public". A user might change his or her profile at any time, either by altering the level of expertise or by hiding some of the information. Hidden information will not be shown to other users when the profile is displayed but will still be used in calculations of expertise for a given query.

Assessing a user's areas and levels of expertise is done through analysis of his or her personal files. In an academic setting it could be done by analyzing one's papers and citations, as seen in [Kautz, Selman. & Shah, 97b]. For a medical information domain, one could use medical records and records of patients seen by each doctor, to establish experience with given diseases or treatments. The important item is that the files used be part of a person's day to day work or school documents. These reflect subjects a person applies in a daily basis and thus has experience with. Use of plain email (unless focused in a mailing list), for instance, is not of much use: email is generally used much more broadly, not being restricted to work related information. Email is good for determining one's interests, but not necessarily one's areas of expertise. The same holds for web browsing.



User Name: rhodes

Email Address: rhodes

Expertise Area	Agents' Assesment	Your Assesment	Hide?
Applet	Beginner	No Change	☐ Yes
BorderLayout	Novice	No Change	☑ Yes
BufferedReader	Novice	No Change	☐ Yes
Button	Beginner		☐ Yes
Choice	Beginner		☑ Yes
Color	Novice		☐ Yes
Dimension	Intermediate	No Change	☑ Yes
Direction	Expert	No Change	☐ Yes
DoubleBufferApplet	Expert	No Change	☐ Yes
EnigmaMachine	Expert	No Change	☐ Yes
Event	Beginner	No Change	☐ Yes
EventPublisherShadow	Novice	No Change	☐ Yes
EventTranslatingAgentImpl	Novice	No Change	☐ Yes

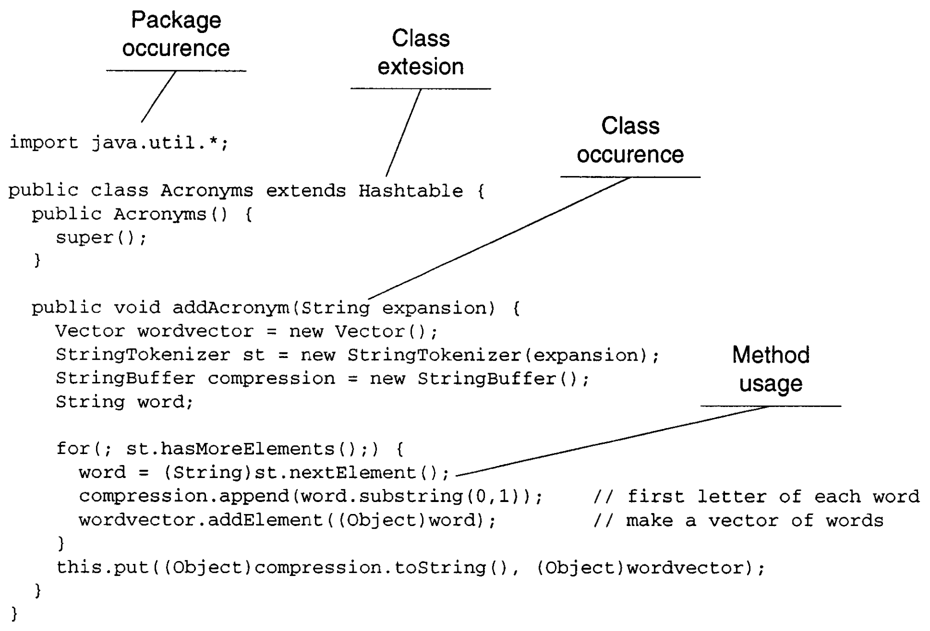
Figure 7: Profile editing window: a user can inspect and edit his or her profile as fit, to compensate for errors in the agent's assessment or hide areas of expertise.

In our case (for the Java programming domain), the agent creates the users' profiles by reading through the user's Java source files and parsing them, analyzing:

1. Libraries: which libraries are being used? How often? Libraries are declared once, usually at the beginning of a file, and are important to determine what functionality is being used in this program.

2. Classes: which classes are used? How often? Classes are declared, instantiated and used throughout the file. Classes represent the structure of a program and how the user has chosen to solve a specific problem. They can also be subclassed, which indicates a deeper knowledge of the object-oriented concept and of the class itself, both its purpose and its working. Implicit in the act of subclassing is the recognition that there is a need for a specialized version of the class and knowledge of how the class works and how it should be changed in each specific case.

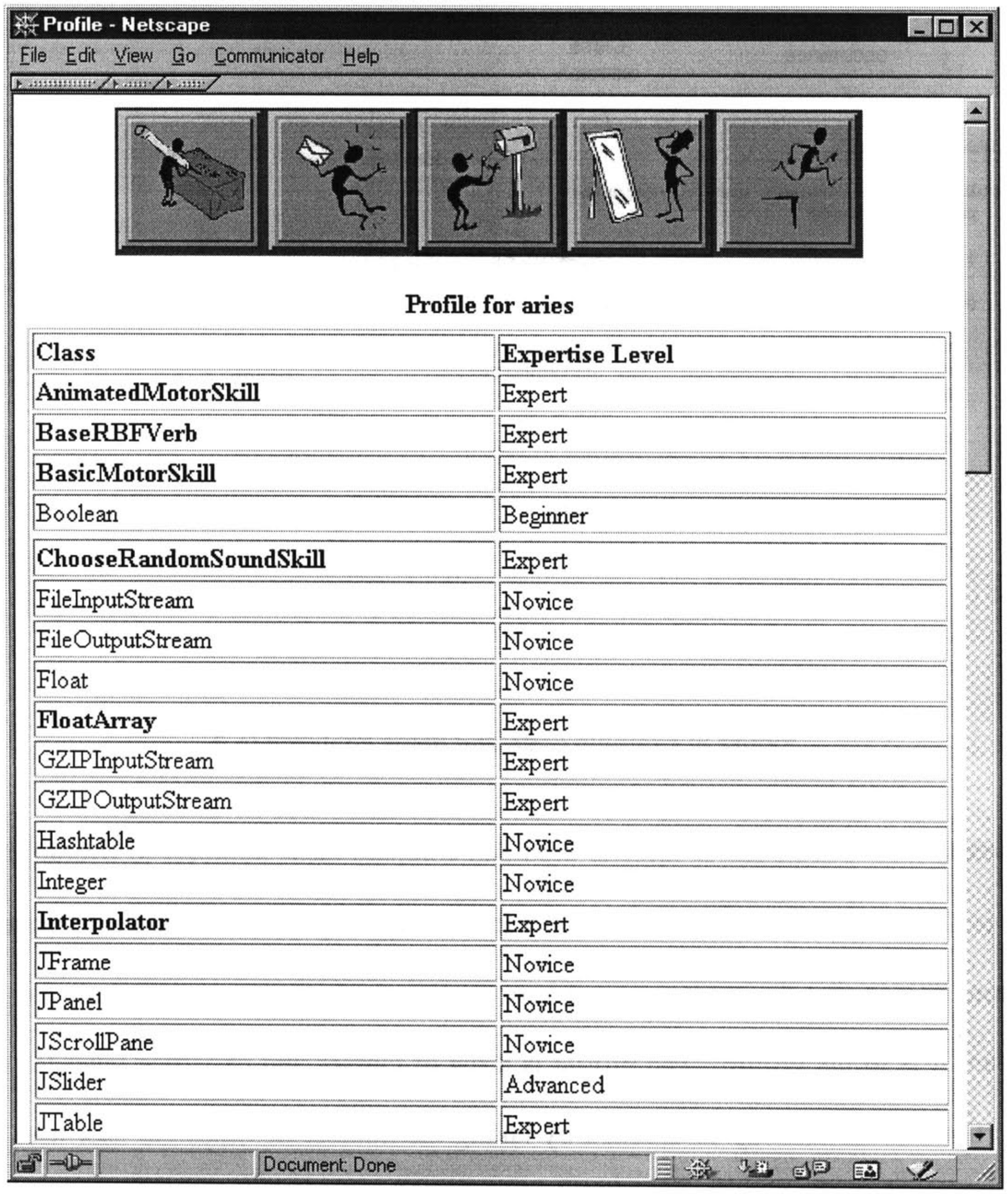
3. Methods: knowing which methods are being used helps us further determine how a user solves a problem and how much he or she knows about a class: are all the possible methods being used? Are only a few methods used over and over again? How extensively is the class used?



```
import java.util.*;

public class Acronyms extends Hashtable {
  public Acronyms() {
    super();
  }

  public void addAcronym(String expansion) {
    Vector wordvector = new Vector();
    StringTokenizer st = new StringTokenizer(expansion);
    StringBuffer compression = new StringBuffer();
    String word;

    for(; st.hasMoreElements();) {
      word = (String)st.nextElement();
      compression.append(word.substring(0,1));     // first letter of each word
      wordvector.addElement((Object)word);        // make a vector of words
    }
    this.put((Object)compression.toString(), (Object)wordvector);
  }
}
```

Figure 8: Example code and items analyzed in it.

We verify how often each of these is used and compare these numbers to overall usage. This is similar to Salton's TFiDF algorithm (term frequency inverse document frequency) [Salton, 88], in that the more a person uses a class that's not generally used, the more relevant it is to his profile. The profile is a list of classes and expertise level for each. Expertise level is initially determined by taking the number of times the user uses each class and dividing by the overall class usage.



Profile for aries

Class	Expertise Level
AnimatedMotorSkill	Expert
BaseRBFVerb	Expert
BasicMotorSkill	Expert
Boolean	Beginner
ChooseRandomSoundSkill	Expert
FileInputStream	Novice
FileOutputStream	Novice
Float	Novice
FloatArray	Expert
GZIPInputStream	Expert
GZIPOutputStream	Expert
Hashtable	Novice
Integer	Novice
Interpolator	Expert
JFrame	Novice
JPanel	Novice
JScrollPane	Novice
JSlider	Advanced
JTable	Expert

Figure 9: Viewing other users' profiles: the items in bold represent classes that have been subclassed. "Hidden" classes are not shown.

The Expert Finder profiling model does not distinguish between shared or copied code. To be able to reuse some code (be it by including libraries or simply by copying code), a person must first, understand it. We assume that, if a user copied code and is modifying it, he or she must know what to do.

Later, we plan on incorporating average class usage (class usage divided by total class usage per person) and extent of usage (number of methods used in each class and operations being performed) for better accuracy. Both are indicators of greater knowledge of the construct and of the language in general.

IV.III.I Privacy

The issue of privacy, though important, isn't central to our research. To use the system, users have to be willing to share their profiles. However, we acknowledge the fact that a person may not want the others to know what he or she knows about, or may not wish to be bothered about one particular issue. The agent allows users to choose which areas of expertise it lets others know about and correct the profile if they think it is incorrect. In this fashion, a person is able to establish the image he or she wants others to have of him or her, since the agent only discloses non-hidden information. We are not particularly worried about people who would pretend they're experts when they're not, since the effect would only be they'd get questions they probably couldn't answer, which would discourage them.

IV.IV Matching Needs and Profiles

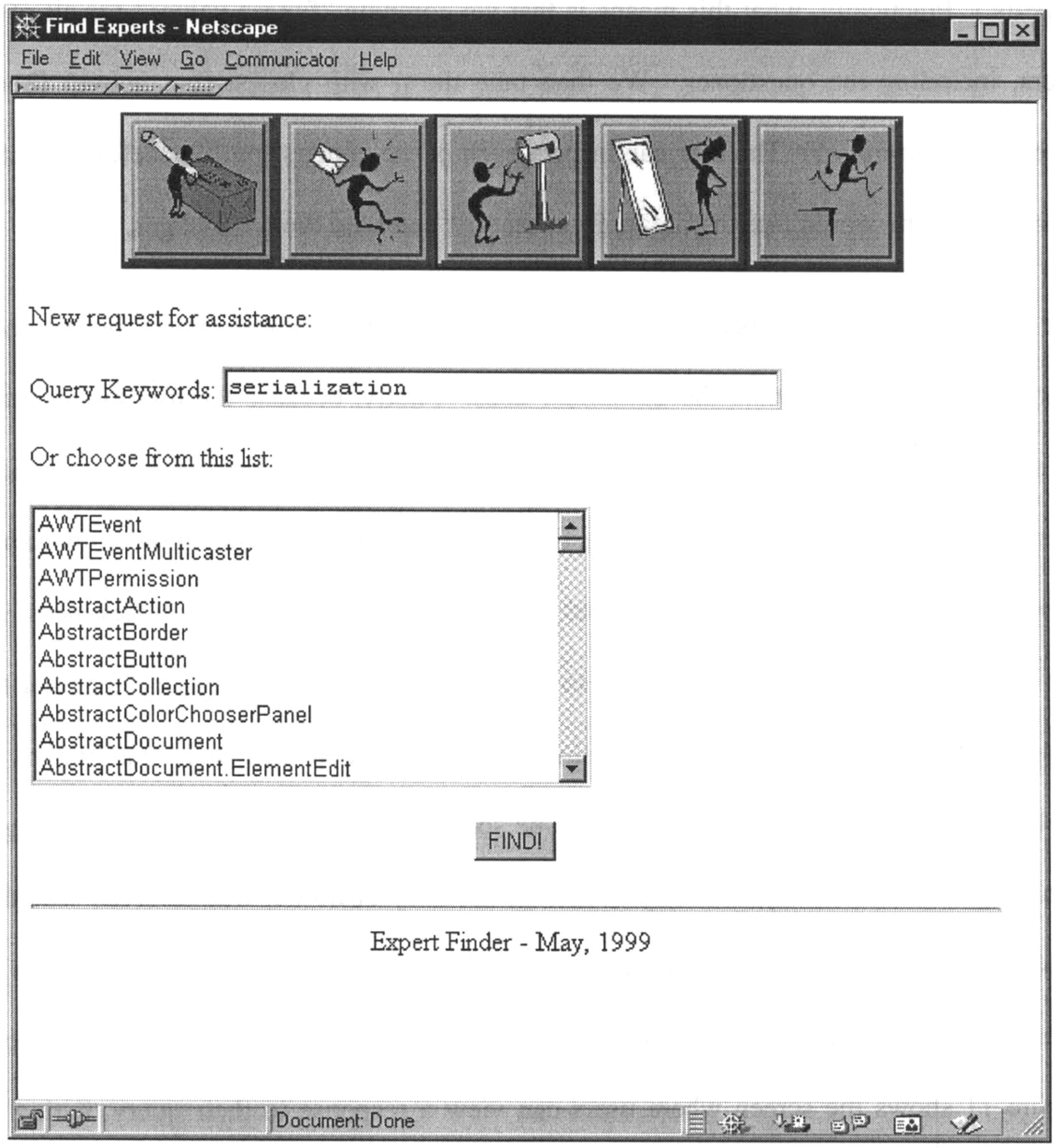
The previously described domain model is used to match queries and experts, so as to be able to identify people with related knowledge, when an expert proper is not available. The system suggests people that may be able to answer questions given the related knowledge they have. Given a query, all related fields of knowledge are taken from the model and added to the query, thus expanding it. This expanded query is then compared to other users' profiles, trying to find an expert. A query can be formulated in one of three ways:

1. Keyword entry: the user enters a set of keywords associated with his or her needs in a text box. The class descriptions are then used to locate appropriate classes from the keywords. These classes are used to find the experts.

2. Selection of classes from a list of those existing in the model: the user chooses from a list of classes. These are then used to find the experts by doing a vector match on the class list and profiles.

3. A combination of both: the user chooses some items from the list and enters some keywords.

A screenshot of the query screen can be seen in Figure 10. From these different query modes, we can infer the user's needs and weigh the links in the model accordingly. If a user selects items from the list, it is reasonable to assume that he or she knows what is needed and needs help with using these classes specifically. Therefore, sub/superclass relations, denoting structural similarity, are more valuable in finding an expert with the desired knowledge. Entering a few keywords means that the user knows what he or she wants to do, but is uncertain of how to do it. In these cases, functional similarity (as established by packages) is more important in determining an expert than structural similarity. If the user uses a combination of both, it seems fair to assume that he or she wants to do something (as specified by the keywords) and thinks he or she knows how to do it (specified by the selections from the list.) In this case, both relations can be used, although it seems reasonable that functional similarity take precedence over structural: the user almost certainly knows what he or she wants to do, even though he or she may not be doing it correctly (this reflects on picking the wrong items in the list.) Given what type of knowledge is needed, it is possible to weigh the links in the model accordingly, prioritizing functional or structural similarity as appropriate. We have not implemented this in this version of the system, however, it would be an interesting (and not too hard) to make this change and see how well it works. This is one of our plans for the near future.



New request for assistance:

Query Keywords: serialization

Or choose from this list:

AWTEvent
AWTEventMulticaster
AWTPermission
AbstractAction
AbstractBorder
AbstractButton
AbstractCollection
AbstractColorChooserPanel
AbstractDocument
AbstractDocument.ElementEdit

Expert Finder - May, 1999

Figure 10: Query screen – a user may choose an item from the list or enter keywords.

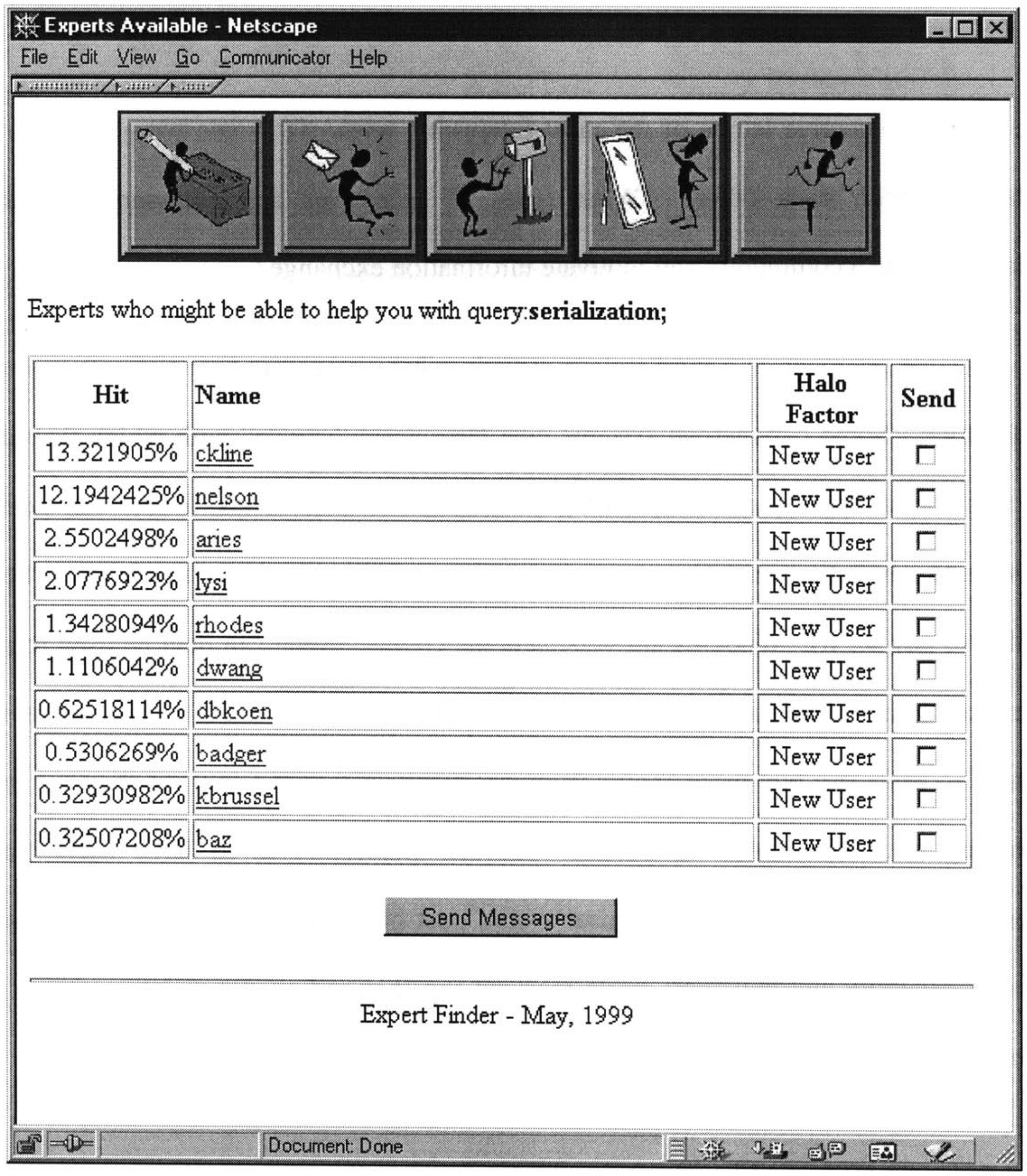
A match is made by first taking the questioner's query (keywords and items from the list) and finding similar topics in the domain model. Taking this set of items, the agent then goes on to contact other agents that are currently active, comparing its user's needs to other users' expertise areas and levels. It computes a vector match, comparing the list of needs to the list of areas of expertise. The agent returns a list of potential helpers, who are just above the level of expertise of the questioner (not necessarily the topmost experts.) Practically, what this means is that we compute "fitness values" for all of the users, including the questioner. We then take the n with closest (but above) fitness, where n is currently 5. The user can inspect each of the experts' profiles before selecting whom he or she would like to contact from that list and send them messages.

We believe that the best person to help is not always the topmost expert, but someone who knows a bit more than the questioner does. Firstly, because the topmost expert is most likely to not be available: he or she probably has several other incoming questions. Besides, he or she may have no interest in simpler questions. But, more importantly, experts and novices have different mental models, as noted by [Ericsson & Charness, 97]. By choosing a person at a level closer to the questioner's, we are bringing together two people who are more likely to have similar mental models and understanding of the problem. Ideally, the expert found would be someone who had recently gone through a similar problem: this way, not only would the memory of how it was resolved would be fresh in his or her mind, but also he or she would be more likely to identify with the questioner (who is now experiencing the same problems) and be more likely to respond. Figure 11 shows the screen where users can view a response to their query, listing the experts available.

IV.V Incentives

The issue of incentive is one of the most interesting ones. How can we get people to collaborate with each other? What types of incentives are necessary to foster collaboration in a community and motivate information exchange?

One possible approach is a marketplace system, where users can buy and sell or trade information. Some commercial systems give their users "credits" with which they can buy answers to their questions [6DOS], [Experts], [Abuzz]. Thus, a user can post a question to a bulletin board-like system and assign a credit value (how much he or she is willing to pay for it). Whoever answers, gets the credits, which can then be used to ask other questions. More recently, some of these have also been exchanging the credits for money, so users can buy more credits from the system if they run out and they get paid according to their number of points. Another possibility is to have users trade information: in such scenario, each user would post his or her areas of knowledge and areas of necessity and two users could then get together and exchange information. In both cases, the system tends to end up being monopolized by a few experts.


Experts Available - Netscape
File Edit View Go Communicator Help

Experts who might be able to help you with query:**serialization;**

Hit	Name	Halo Factor	Send
13.321905%	ckline	New User	☐
12.1942425%	nelson	New User	☐
2.5502498%	aries	New User	☐
2.0776923%	lysi	New User	☐
1.3428094%	rhodes	New User	☐
1.1106042%	dwang	New User	☐
0.62518114%	dbkoen	New User	☐
0.5306269%	badger	New User	☐
0.32930982%	kbrussel	New User	☐
0.32507208%	baz	New User	☐

Expert Finder - May, 1999

Figure 11: Expert list screen – experts are ranked by appropriateness to a given query.

There are, however, other (non-monetary) reasons for helping out. Some studies have shown that people will answer questions for personal reasons, such as "being helpful" or "showing off" [Ackerman & Palen, 96]. According to Grzelak [Grzelak & Derlega, 82], cooperative behavior is behavior that maximizes both the individual's and other's interests. It seems possible, then, to create an environment where people would be more cooperative.

Using referrals would work much in the way that one would be inclined to help a friend of a friend. In this case, given a social network, all that is necessary is to map it and look for experts in this network. Recommending a direct friend (a hop away) or someone from a friend's set of friends (two hops away) is likely to yield someone willing to help, since there is a direct connection between questioner and answerer. But what happens when there is nobody in the "friend of a friend" vicinity?

Taking the information trading example one step further, one could envision an environment where people post their areas of expertise and answer queries, not because they'll get something in exchange from the other person, but because *they'll get some value from the community*. That is, someone else in the community will answer their questions later.

Identification with the questioner is another incentive. People who have experienced the same problem or have been in a similar situation will relate to the questioner and be more inclined to provide assistance. One other factor (which we hope to take advantage from) is knowledge that the other person is a "good citizen", i.e., someone helpful to the community. The idea is that one will be more inclined to help someone who is generally helpful in the community. This is in line with the concept of Social Capital, as understood by Putnam [Putnam, 93] and others: social capital refers to features of social organizations that facilitate coordination and cooperation for mutual benefit. In our system, we try to strengthen the social capital, creating a mechanism to make cooperation easier.

In this spirit, we have built into the system an incentive mechanism to assess the social capital in the community. We keep track of how helpful each person generally is (we call this indicator the halo factor), and display this information whenever someone is asking a question. The halo factor of a person is calculated by taking the percentage of questions answered from those received ([Qa/Qr]*100). It is displayed every time a person sends a question (so that the person being asked can know how helpful this person is) and we hope this will motivate people into answering questions (at least for those who are also helpful.) When a person is new to the system or has never received any questions (Qr = 0), the person is billed as being new to the system. We do not use the number of questions sent as a factor because we don't want to penalize or otherwise inhibit a user from asking questions (and asking how many questions one has asked could be interpreted as how much work one is giving others.)

This does not mean that only a few people will get questions and the rest will only be asking them. As noted before, the best expert is not necessarily the topmost expert, but someone at a level closer to the questioner's. Furthermore, as the system keeps track of questions sent and received, we can more evenly distribute questions when there are multiple experts available. This prevents the overloading of one or a few individuals with all the questions.

IV.VI *Architecture*

The ideal architecture for a system like Expert Finder would be totally decentralized: each user would have one agent, which holds the user's profile, talks to other agents (providing its user's profile and getting profiles from other agents), calculates how well the other users fit when its user needs assistance. In such decentralized system, the agents could use either referral mechanisms [Foner, 99] or some form of address book to get to each other (or a combination of both.) A completely distributed agent architecture which could easily be used for our purposes has been designed by Foner [Foner, 99]. A similar implementation has been used in ReferralWeb [Kautz, Selman. & Shah, 97b].Given that this research is not about architecture, we do not concern ourselves too much with decentralization. We have instead, implemented two different versions of the system, each with a different, simpler architecture.

For the client server, we used a registry server, which holds a list of all active agents. Each agent registered its presence with the server when it first started. This server also holds the most recently updated domain model, which is sent to the agents periodically. It also keeps track of "global values" such as number of users and overall class usage.

The client side held the user's profile and a copy of the domain similarity model. It could communicate with other agents and compare its user's needs to other users' profiles and decide which users are best suited to help. The architecture and information flow for this implementation is illustrated in Figure 12. In this version, each user has an agent running locally, which parses the users' files, keeps their profiles and talks to other agents in other machines.

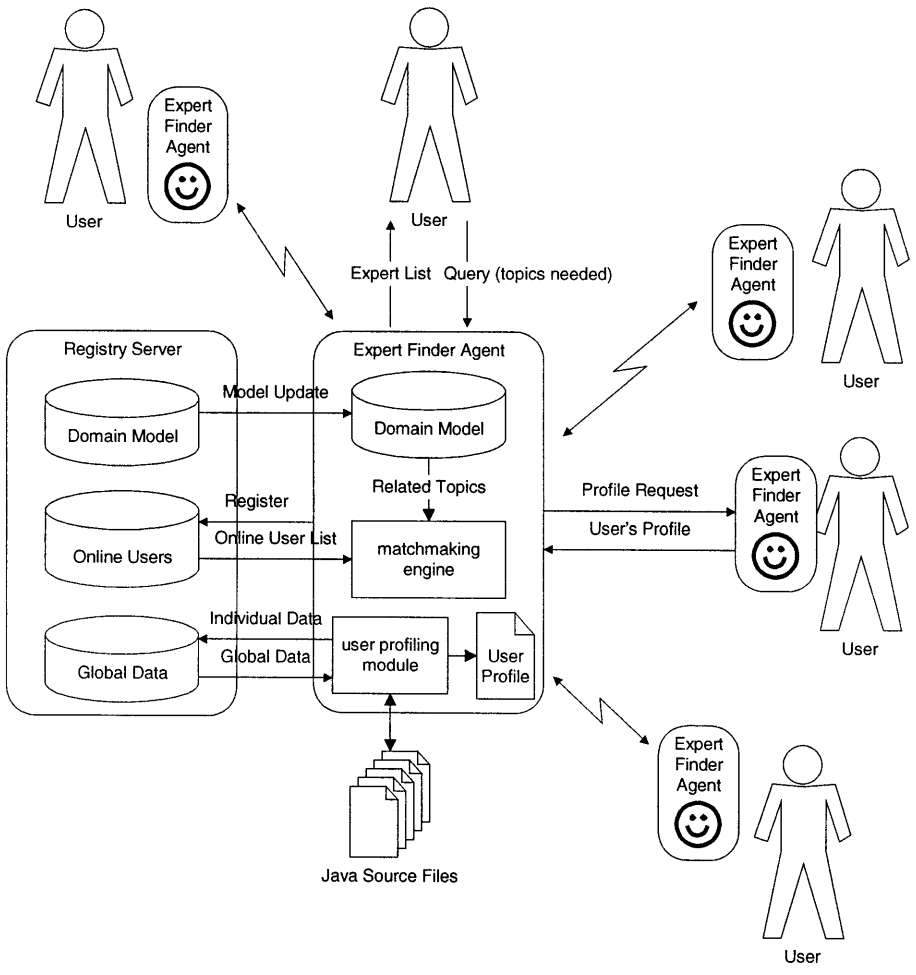


Figure 12: Client-Server Architecture: A user queries his/her agent. The agent consults the registry server and obtains a list of all currently active agents. It then goes on to contact each of these, requesting their users' profiles. It takes the profile, the query, related topics and its own user's profile and determines good matches.

For the web version, we implemented a centralized system, where the server holds the information about the users and the domain model, and all queries are sent there to be answered. The users' code is also kept in the server and centrally maintained, which means it could be volunteered as example code if necessary. Centralizing the system allows us to keep track of messages sent and received, in case we want to use them as resources for future transactions, providing previously asked questions and answers for a user. Hopefully this would help avoid unnecessary or repetitive inquiries or at least make them easier to answer.

It is important to note that a hybrid system, where expert-novice interactions are saved and then reused to avoid similar questions being asked multiple times would also make perfect sense, and is something we would like to add in a later version of the system. In this fashion, the agent could suggest not only experts who might be able to help, but also previous answers to the same question and other resources.. This would potentially cut down on the number of questions asked.

IV.VII Interface

The Expert Finder interface was built for simplicity: it is web based, and, once the user has logged in, he or she can inspect his or her profile and make changes if necessary. A button bar (Figure 13) on the top of each page gives each user the options available: making a new query, viewing responses to his or her previous questions, viewing questions he or she may have received, editing the profile and logging out.

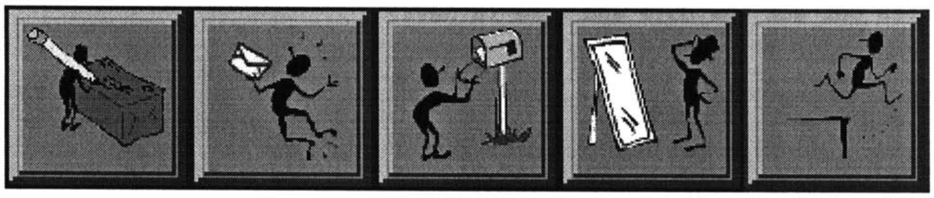
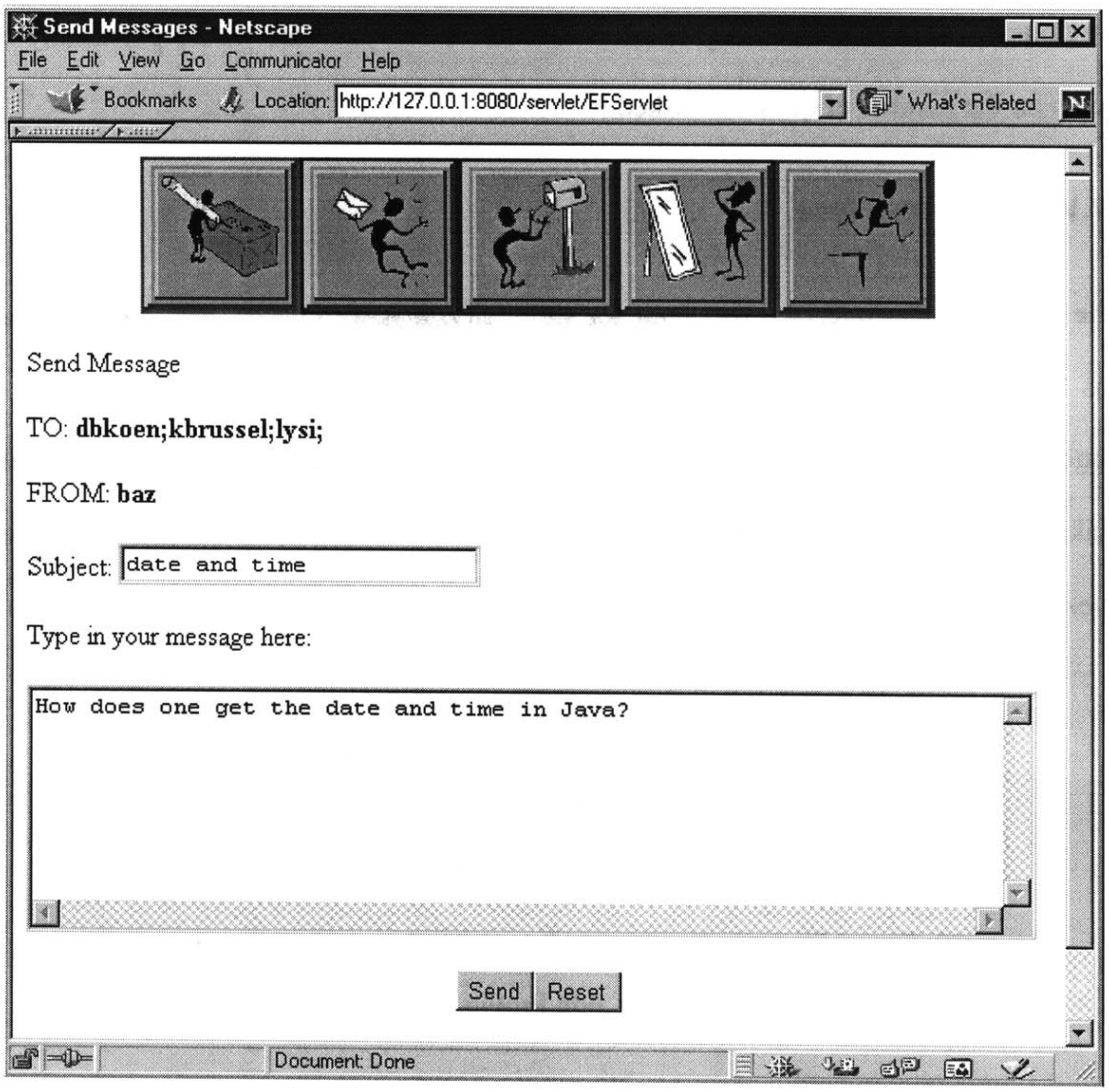


Figure 13: Expert Finder button bar. Left to right: Query, View Responses, View Questions, Edit Profile, Logout.

A user can edit his or her profile on the profile-editing screen, shown previously in Figure

7. The queries are submitted to the system via the query interface, Figure 10. The results of the query are then shown in the result screen (Figure 11). Clicking on one of the expert's names, the user may inspect this person's profile in detail, verifying which classes he or she knows how to use, as seen in Figure 9. Still on the result screen, the user can select which experts he or she wants to send messages to, select the corresponding checkboxes and click "Send Message" to go to the message composition screen - Figure 14.


Send Messages - Netscape
File Edit View Go Communicator Help
Bookmarks
Location: http://127.0.0.1:8080/servlet/EFServlet
What's Related
Send
Reset
Document: Done

Send Message

TO: **dbkoen;kbrussel;lysi;**

FROM: **baz**

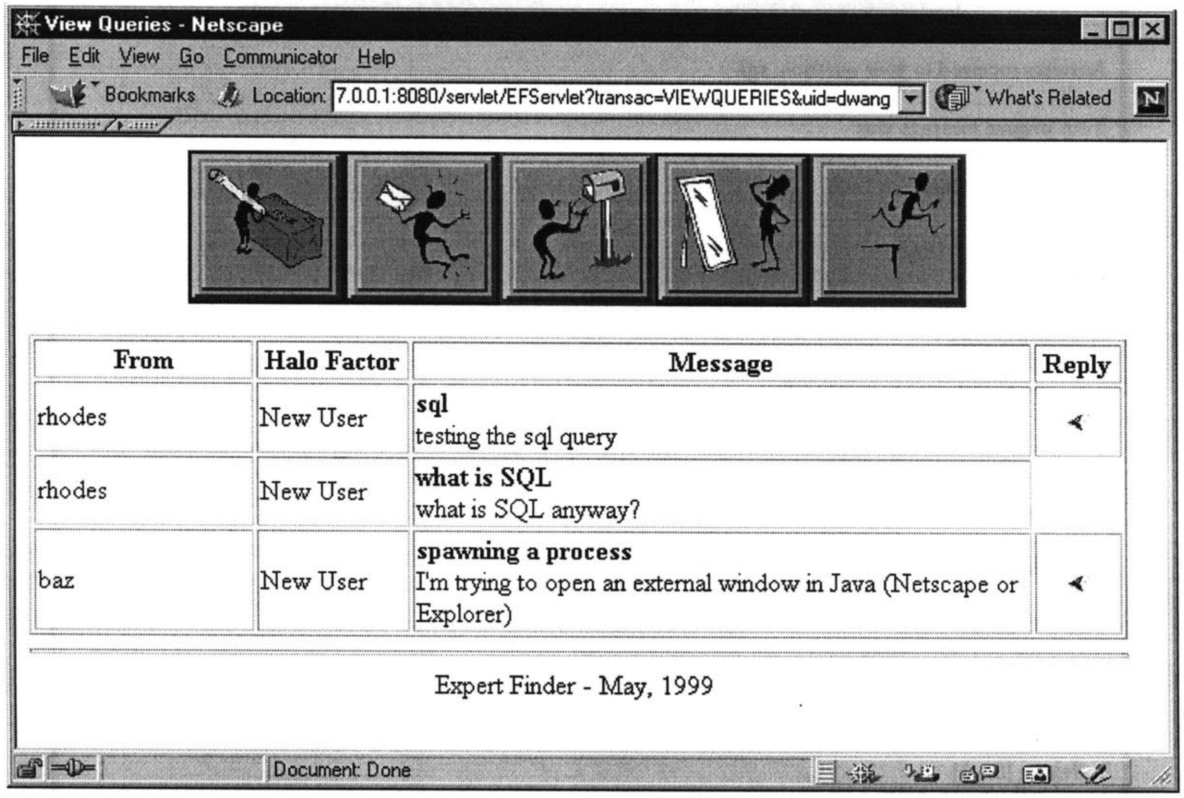
Subject: date and time

Type in your message here:

How does one get the date and time in Java?

Figure 14: Send message screen. The user composes and send a message to the experts.

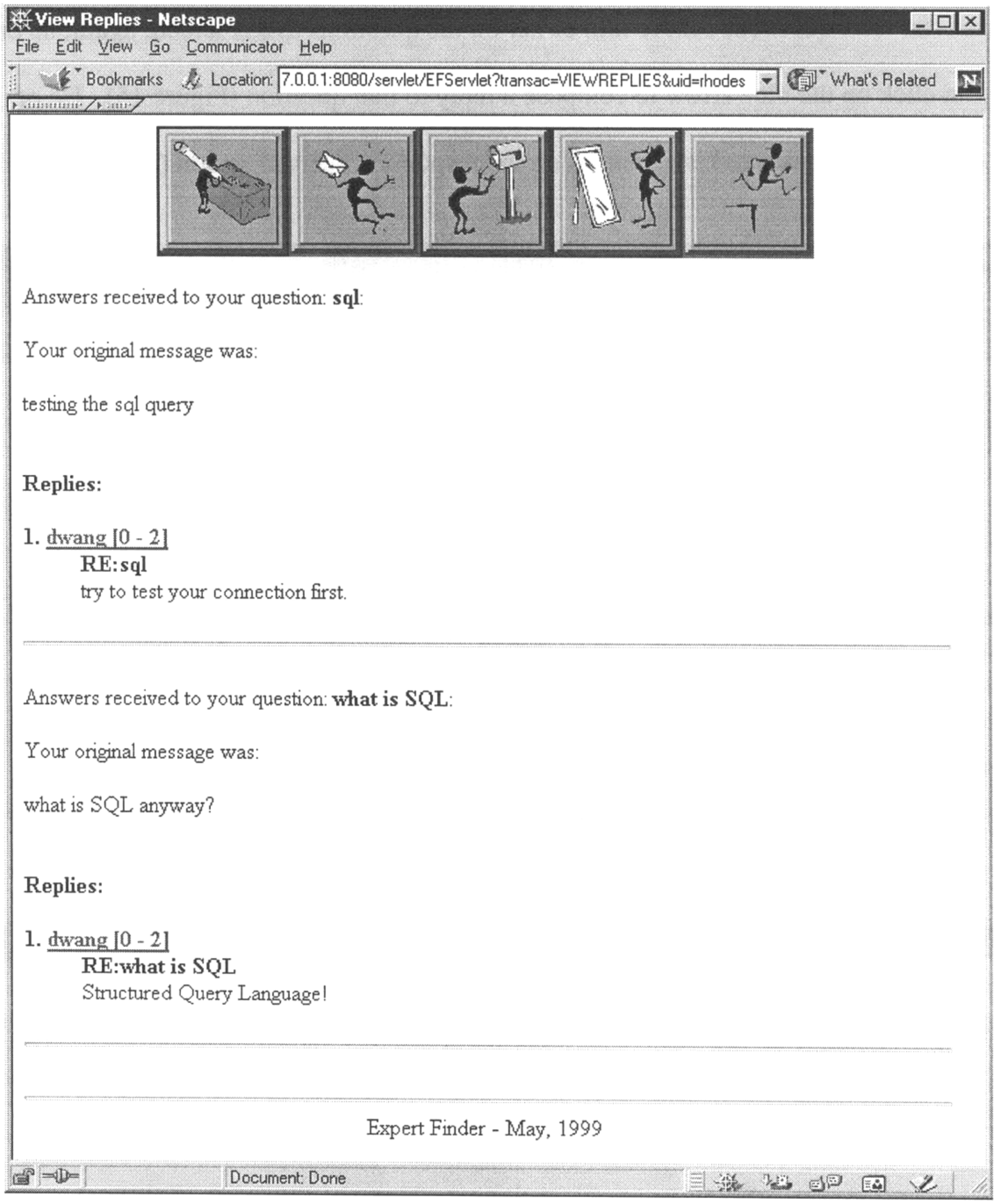
A user can view questions sent to him or her on the questions asked screen: again, he or she can click on the blue arrow to compose a reply to that question (Figure 15). He or she can view responses to his questions on the responses received screen (Figure 16).


View Queries - Netscape
File Edit View Go Communicator Help
Bookmarks
Location: 7.0.0.1:8080/servlet/EFServlet?transac=VIEWQUERIES&uid=dwang
What's Related
Document: Done

From	Halo Factor	Message	Reply
rhodes	New User	**sql** testing the sql query	◀
rhodes	New User	**what is SQL** what is SQL anyway?	
baz	New User	**spawning a process** I'm trying to open an external window in Java (Netscape or Explorer)	◀

Expert Finder - May, 1999

Figure 15: View of the questions received: the expert can click on the blue arrow on the right to start composing a reply.


View Replies - Netscape
File Edit View Go Communicator Help
Bookmarks
Location: 7.0.0.1:8080/servlet/EFServlet?transac=VIEWREPLIES&uid=rhodes
What's Related
Document: Done

Answers received to your question: **sql**:

Your original message was:

testing the sql query

Replies:

1. dwang [0 - 2]
   RE:sql
   try to test your connection first.

Answers received to your question: **what is SQL**:

Your original message was:

what is SQL anyway?

Replies:

1. dwang [0 - 2]
   RE:what is SQL
   Structured Query Language!

Expert Finder - May, 1999

Figure 16: Viewing answers received to one's questions.

Chapter V: Evaluation

As an evaluation for this work, we built a prototype system (for the Java programming domain), generated profiles for a 10 users, ran 20 queries through the system and determined whether the experts suggested by the system would be able to answer those questions through a questionnaire. Questions used covered different angles of the language, and were taken from the Experts-Exchange forum, thus constituting real problems people have. They ranged from very specific ("How do I add an item to a JList") to the more generic ("What are static entries and what are they good for?"). We asked the experts whether they would be able to answer each of the 20 questions. Possible answers were "Yes", "No" and "Not flat out, but I would know where to look", for cases where an expert might not know an answer right away, but would be able to quickly get to an answer by consulting some of his/her resources. We also showed the users their profiles, so they could verify how well it represented their expertise. We allowed them to edit their profiles and then compared what the agent had said to what the users claimed.

V.I Profiling

To test how well the profiling module worked, we generated profiles for 10 users and had them look at their profiles and edit them (on the profile editing window, shown in Figure 7). We then took the original and edited profiles and checked to see how many items were altered and by how much. Users were also encouraged to send any additional feedback they deemed necessary. Users' profiles are kept in files divided into four parts:

1. Generic information: user name, base directory under which to look for code files, email address, list of files used in compiling the profile.

Name:	John Doe
Base Directory:	H:\Projects\ExpertFinder\Profiles\doe
Email	Doe@pobox.com
File List:	H:\doe\src\Compass.java H:\doe\src\GpsDread.java H:\doe\src\GpsRead.java H:\doe\src\Gpstest.java H:\doe\src\PhysObj.java

Table 3: Generic information in the profile

2. Totals: total number of times the user has used a certain class, library or method, and the classes the user extends in his or her code.

Package Totals:	java.io: 5; java.util: 2; java.lang: 2;java.awt: 7; java.awt.event: 3;java.net: 1
Class Totals:	System: 42; StringTokenizer: 2; PhysObj: 1; Point: 2; Graphics: 18; Frame: 7; Thread: 34; Double: 2; Dimension: 6; MouseEvent: 4; String: 21; Socket: 5
Method Totals:	SetPriority: 1; nextFloat: 5; exec: 2; stringWidth: 1; nextInt: 1; sleep: 18; getY: 3; getX: 4; waitFor: 1; whitespaceChars: 2; getRuntime: 2; exit: 3; start: 15; nextToken: 5
Extended Classes	Frame: 7; Thread: 3

Table 4: Totals in the profile

3. Agent's calculations: this is the expertise level the agent calculated for the user. When displayed, these are translated into novice, beginner, intermediate, advanced, expert, according to the value calculated.

Agent's Calculation:	java.io: 3.356; java.util: 0.651; java.lang: 9.091; java.awt: 4.167; java.awt.event: 6.122; java.net: 1.538; System: 4.444; StringTokenizer: 5.263; PhysObj: 100.0; Point: 4.348; Graphics: 9.524; Thread: 21.519; Double: 6.667; Dimension: 4.379; MouseEvent: 11.765; String: 0.688; Socket: 6.024

Table 5: Agent-calculated expertise

4. User values: User correction to the agent's calculation, values to be hidden from other users, and pointers to incoming and outgoing messages.

User Values:	Java.io: 60; java.util: 40; java.awt: 20; Graphics: 80; Thread: 60
Hidden Items:	StringTokenizer; Point; Double: 2
Incoming Messages:	A5
Outgoing Messages	Q16; Q17

Table 6: User values: user's alterations on the profiles, pointers to messages (stored separately, in text files).

Users were presented with their profiles, and asked to edit them through the profile editing interface. In average, it seems users edited about 50% of their profiles (some users changed less than 50 %, but others changed more than that.) The number of changes ranged from 9% for the least altered profile to 63% for the most altered (this was one case where we expected something like this: we knew we had few code samples from this particular user to properly represent his abilities, which was reflected by the large number of changes to the profile.) In average, changes were made to about 40% of the expertise areas. Both increases and decreases on expertise levels were made, with about one third of all changes being a decrease and the rest an increase in level.

These changes happened, with but a few exceptions, on commonly used classes or classes many users use. In these cases, all users felt they were very knowledgeable (intermediate and above) on the given class or package, even though their profile indicated otherwise. One such example is the Hashtable class: all the users increased their level of expertise with the Hashtable class to Advanced or Expert, from whatever it was before. This discrepancy happens because there were many experts using this class and what we calculate for the profiles is what percentage of the total usage belongs to each of those experts. Thus, whenever many people are using the same class, this percentage gets spread between the various users, not necessarily placing any of them as the top expert (i.e., if someone is responsible for 55% of the total usage for the Hashtable class, he or she will be placed in the intermediate level.)

This may indicate a lack of variety in the sampling, for all users were reasonably proficient with the Java language, and tried to adjust their profiles accordingly. Classes that were used by only one or two people usually had one very knowledgeable person and one at Beginner level.

The decreases for the most part happened when there was only one user who used a given construct, and was therefore deemed the expert. Oftentimes the user decreased the rating to either advanced or intermediate level. This again happens because we take the combined usage into account: if nobody else is using this class, the user is responsible for 100% of its usage in the community, and is, therefore, an Expert. Only one user decreased his expertise level saying that he had copied some code and had no idea how it worked. The others, if they copied something, understood how it worked. A few commented on how part of the code (reflected on the profile) had been written by someone else, but they were aware of those and how they worked, and indeed felt they were responsible for that as well.

As far as how large the changes were, 31% were 1 step changes (for instance, novice to beginner), 33% 2 step changes (novice to intermediate), 26% 3 step changes (novice to advanced) and 10% 4 step changes (novice to expert.) These numbers seem to indicate that the agent's calculations weren't so far off the mark: changes were of one or two steps for the most part and about a third of the 4-step changes were expertise decrease changes, from a setting of one expert to novice (this reflect the situations where the user is the only one using a construct, even if he or she is not an expert in it.)

To sum up, this doesn't seem to be a bad approach to building user expertise profiles, but it could certainly be improved to become more accurate. One user commented that he felt it made more sense to rate his knowledge on the packages rather than on individual classes. He felt that there was a need for more abstract categories or groupings (one example was the checkbox and choice classes, which are very similar and knowing about one means knowing about the other.) This is something we tried to map with the domain similarity model, and reflects a greater need for more abstract level knowledge. The existing profiles are very fine grained and held no extra layer of abstraction.

V.II Matchmaking

Overall, the system performed well enough, always placing at least one expert who had had said he could have answered the questions (either right away or looking it up) in the first three recommendations. We now go into more detail about what happened.

Number of success cases (recommending experts who would be able to provide an answer) was around 85%. Breaking these down, 35% were "immediate success" cases (the first expert recommended said he'd be able to answer it right away) and 50% were "delayed success" (the expert answered that he'd be able to answer by looking it up.) In 15 % of the cases, the expert said he would not be able to answer the question.

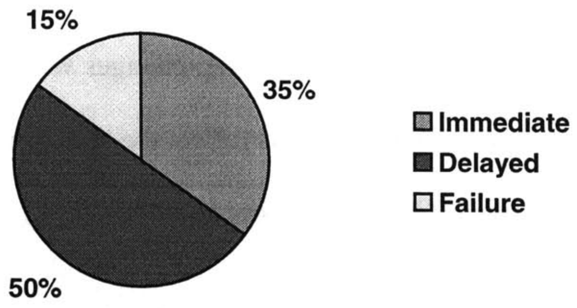


Figure 17: Distribution of Success/Failure cases.

However, in 50% of the cases, the person found to be the topmost expert said he'd be able to answer the question right away, in 35% of the cases he said he'd know where to look and in 15% he said he wouldn't be able to answer the question. The discrepancy between finding experts and giving good recommendations was caused by the fact that the system recommends people at a level of expertise close to that of the questioner: given that the questioner had little or no expertise, the system did not always recommend people well suited to answer.

For specific queries (queries about one specific object and how it works), the system produced good assessments of who the experts were. When the users had the knowledge (i.e., they had used the class or library being sought), the system produced correct assessments of who was the topmost expert (even if this didn't translate to good recommendations.) Taking the top 3 experts found (not recommended) for specific queries, we have 52% said they could answer the question, 19% said they would look it up and 29% said they could not. Analyzing the failure cases (where Expert Finder placed users who said they could not answer the questions or that they would have to look it up higher than the ones who said they'd answer it right away), we found that these were either cases in which the related knowledge model was used to get to an answer or cases where there was no indication that a user had this knowledge in his profile.

In the first situation, no expert said he'd be able to answer the question, although some said they'd know where to look. A quick check of the profiles revealed that none of the experts had these classes in their profiles, either. Therefore, the system had to use the related knowledge model to search for experts. The same happened in the second situation, although this time, despite the fact that a user said he knew how to use a given class, there was no indication in his code to support that statement, and therefore the system couldn't place him very high. After the user adjusts his profile (adding the class), Expert Finder will probably do a better job.

In general, in the cases where related knowledge was needed, Expert Finder produced acceptable results, although not necessarily the optimal choices (once again, ranking experts incorrectly.) This probably means that the model needs to be adjusted to produce better output for the similarity relations, which would result in better matches. It seems that, when the questions are specific enough and the user base large enough, there will be little need for a similarity model (since there would always be experts for the given queries.) However, it might be interesting to have the model for situations when experts were unavailable. We'd like to do more testing on this aspect of the system, changing weights and links to see if it becomes more effective.

More abstract queries yielded worse results. Once again, taking the top 3 experts found, we have that 45% had claimed they'd be able to answer the questions right away, 25% said they'd be unable to answer the questions and 30% said they'd have to look them up. Despite the apparently good results, we consider these not to be as good as the previous ones. In most cases, Expert Finder placed experts incorrectly, ranking users who had said they couldn't answer higher than others who said they would be able to answer them. This probably happened due to the method used to retrieve keywords (searching through the specification descriptions), since most of these queries were made using keyword entry. There seems to be a need for a more elaborate level of abstraction in the domain model to enable better matches to be made. One interesting possibility would be to extract text from focused mailing lists or newsgroup posts, and match them not only with their authors (adding more expertise information to a user's profile), but also with the classes they refer to (adding more information to what's already in the model.)

V.III Other Observations and User Feedback

One interesting piece of feedback we received raised the issue of time ("I would have known how to answer these questions last year, when I was working with this – right now, I don't know how to answer them, I have forgotten how it works".) Clearly, in this case, the issue of time is critical: after a while in disuse, skills are forgotten. However, this knowledge can usually be regained more easily (say, by looking at one's old files or a quick reference book) than learning it the first time around. Having worked with a construct before, these people are more resourceful and usually know where to look to get the desired information: they remember having done it, so they'll pull out their files, books or notes and refresh their memories.

Some users claimed they were able to answer some of the questions even though their profiles did not seem to indicate they would. This was probably because either they did not give the Expert Finder all their code (therefore rendering their profiles incomplete), or they had tried something and then discarded it, which means there was no record of it in their code files. Thus, it seems that keeping a history of what a user has used or studied in the past, things he or she has looked at and then discarded, would be a valuable addition to the users' profiles.

A few users noted that there were items in their profile they didn't know about. This was caused because they submitted code that had been written by their UROPs or other assistants. This was one cause for change in user profiles, although users acknowledged that they were responsible for that code and they had reviewed it and understood how it worked.

Testing the incentive mechanism would demand a lengthy experiment, having users (who don't know each other) use the system for a while and establishing whether exposing each person's helpfulness makes a difference in the interactions. We did not run any experiments specifically for that, however, we do believe that exposing these patterns of interaction is valuable in a community, especially when users don't know each other and therefore have no way of assessing to whom they're talking. A reputation mechanism based on previous interactions could also be employed for similar purposes.

Chapter VI: Future Work

This work opens up possibilities in a number of different areas. More work can be done in profiling, i.e. how to extract user profiles from code or other available sources. Work can be done in ontology systems and how to build collaborative ontologies that can be extended as the users needs dictate. Another important area of work is that of context aware systems, and making the agents more proactive: an agent could, reading through the user's error messages, detect the need for help. It could also mediate interactions between novices and experts. Each of these is described in more detail below.

VI.I Profile Building

Profile-building is an obvious area to work on. Where does the information come from? How to process different types of data? What types of data are available and appropriate?

For different domain areas, there will be different sources and types of expertise information available, from which to build user profiles. In academic settings, one could use papers written, read and referenced to determine areas of expertise, similar to MITRE's Expert Finder [Mattox, 98] and Kautz's ReferralWeb [Kautz, Selman & Shah, 97b]. For the medical information domain, one could use patient's records and records of which patients a given doctor has seen (i.e., what he or she has dealt with before.) In this case, there may be a few different situations: one could have a patient looking for other patients (to form a support group) or looking for a doctor (for advice). Or, one could have a doctor looking for another doctor (for instance, one that has experience doing a certain procedure.) Even in the Java programming domain, we could perform more complex code analysis, which might reveal more about one's programming style, abilities and efficiency.

There are different ways of figuring out what one's areas and levels of expertise are. It is important, however, not to lose sight of some important guidelines: expertise becomes more solid with practice. It has been noted that the cognitive changes characteristic of expertise require many years of practice for their development [Feltovich, Ford & Hoffman, 97]. It makes sense, then to use work-related documents, since those reflect activities performed and knowledge used in a daily basis.

One other consideration on this topic is the issue of time, or what we call "decaying expertise": after a while, people forget how to do things, if they don't keep working on it. This is especially the case in this situation, where someone may have used certain functions a year ago and not since, thus forgetting how they worked. In these situations, it would be useful to keep track of when the last time was that a user worked with a given construct or general area. This also goes for other areas, where a user may very well lose track of something he or she hasn't worked on in a while. Therefore, the last time a person has worked on something should be recorded and taken into account when assessing expertise (which means, one might been an expert at some point, but no longer remember what it was all about.) As Seifert [Seifert, et. al] notes, expertise comes with experience, and memory plays an important part.

VI.II Collaborative Ontologies

Another area of research that is related is that of collaborative ontologies. The current Expert Finder system relies on a domain model, and would greatly benefit from its being more flexible. Ideally, users would be able to add and correct the model incrementally, and that would increase performance of the system.

This generates a few questions: how does one allow users to manipulate the underlying model without messing it up? This is not just a knowledge acquisition question, it involves creating mechanisms to ensure that several people can provide input to the system and that the final model represents the collective ideas and not just one person's. This would probably involve creating voting mechanisms for validation of parts of the model, determining each user's trustworthiness before accepting their input and creating ways to validate what is provided as input and prevent inconsistencies in the model. These are all hard and interesting problems, deserving of greater attention.

VI.III Proactive Systems

The most immediate next step for Expert Finder however, would be making it more proactive. For one, having the agent watch what the user is doing and try to figure out when he or she needs help. That could be done by watching, for instance, error messages the user is getting as he or she writes the program. It could also be done by verifying when the user goes to the help system. These are all indicators that the user needs help, and a context-aware agent built into the development environment could very well detect these signals and give some suggestions. Deciding where help is needed could be done by looking at the error messages and at the most recently edited files (what the user is working on right now.)

One other issue is mediating novice-expert communication: given that novices and expert have different mental models of the world, it would be useful to have the agents mediate that interaction. The agent could, for instance, help compose the messages by inserting pieces of the questioner's code or the error messages he or she has been getting. It could also help the expert deal with the problem by providing manual pages and other documentation about the classes in question and code samples (preferably of the expert's own code) where the same classes were used to help the expert remember how he or she dealt with this problem before.

Chapter VII: Lessons Learned

We have presented the Expert Finder, an agent that builds users' expertise profiles and helps them find other users who can help them when needed. We have proposed a full system with a profiling module, a matchmaking engine and a non-monetary incentive mechanism. A prototype system was built for the Java programming domain, and user profiles were created using real users' code. We ran a set of queries to test the system and found out that it works but can be improved upon.

We believe that this project is just a start. The problem certainly deserves more work, and we have seen where it works and where it doesn't, and proposed ways of extending the current project to be more generic or better in this domain. There are some ideas that need more experimentation, especially the incentive mechanisms and expert novice mediation agents. We hope to continue working and improving on this theme and others and that the Expert Finder will some day become a run of the mill system available for all to use.

Chapter VIII:Related Work

A series of projects have some relation with expert finder. Some of them are traditional Expert Systems with similar functionality, some are agent-based systems, and some are sociological studies. In this chapter we get into detail on a few of them (the most relevant ones) and briefly mention some of the others. We have also looked at other areas of expertise to better understand what drives people to seek and provide help, and create an environment to foster collaboration. Some of that work is also included in this chapter.

VIII.I Information Marketplaces

VIII.I.I Experts-Exchange

Experts-exchange [Experts, 97a] is perhaps the most successful commercial system in this area. When a user first registers with the system, he or she receives 200 credits, which can then be used towards the purchase of answers, either by posting questions to the system and waiting for the answers or by buying answers to previously asked questions (for a fraction of their original value). The user receives 5 additional credits each day thereafter, which is enough for 2 or 3 questions a month. As Jessup [Jessup, 97] notes, points are a scarce resource for the questioner, which effectively raises the quality of questions, preventing them from becoming chatter. The system uses a predetermined expertise directory, under which questions and answers are posted. This may be expanded through users' suggestions.

To ask a question in Experts-Exchange, the user goes to the topic directories and posts a message with a question under the category that befits it. Self-proclaimed experts will then answer this question. The experts log on to the system and navigate the structure, looking for unanswered questions. They pick a question and post an answer or comment, which is usually a request for clarification. This clarification process goes on until a final answer is given and accepted by the questioner. In this case, the question and answer become "locked" until the questioner has the opportunity to evaluate the answer given. The evaluation is entered into the system and the expert receives the number of credits times the grade assigned to the answer as expert points (which are not exchangeable back into question points.)

In this system, it is hard to assess the quality of the answer. Most of the topics in the ontology have to do with computers and programming, so they are easier to check for response quality. However, for other kinds of topics (such as cooking or art history), it may be hard to assess whether the response was good or not, generating a potential for misleading answers.

All a questioner knows about the answerer is their number of "expertise points", which reflects how good this person is at answering questions. However, these points are all bundled up into one pool: the system does not keep track of where the points came from and, therefore there is no detailed expertise profile of the expert (i.e., one does not know which areas the person is answering questions in, how often or how well.) I believe it would be helpful to the system to add this tracking and create this more detailed view of each user.

A forthcoming version of the system, the Gold Site [Experts, 97b] will be based on actual money, with users being able to purchase question points with their credit cards and experts will be paid for answering questions. The top experts form their free site will be invited to answer questions on the paid site. Also, the topic ontology will be much more limited than the one on their free site. Recognition is the major incentive for people to participate as experts on the free site, on the Gold Site there is a more direct, monetary incentive.

While designing the Expert Finder, we considered creating a marketplace in these molds. The main differences here are that Experts-Exchange doesn't automatically generate a user profile and there aren't any recommendations made to the questioner: he or she simply posts a question in a bulletin board-like system and waits for an answer.

VIII.I.II Beehive

Abuzz's main system, Beehive [Abuzz, 97], is a knowledge management tool based on the marketplace paradigm[1]. Buyers place requests for information in the Beehive

[1] Note: Information on the Beehive system was taken from a white paper which was up on the Abuzz

marketplace and sellers make offerings that reflect their skills and attract specific requests. Each request is composed of a definition and a price range, where prices are set by the buyers. An offering is also a definition and a price range, and anything can be offered: FAQs, documents, Usenet posts, personal assistance... Contracts can be initiated by buyers or sellers. The system keeps track of all the offerings made and filters the marketplace, matching requests and offerings. It uses data mining technologies to extract information about the existing offerings. In essence, it creates a "Yellow Pages" type ontology, creating shortcuts to buyer and seller groups. It is meant to be used in a company setting.

Again, this is a marketplace setting for expertise location. In this system, sellers create explicit offerings that reflect their skills. It probably requires a user-entered and maintained profile, which oftentimes is hard to get, since most users shudder at the thought of filling out forms or lengthy questionnaires.

VIII.II Referral Systems

VIII.II.I ReferralWeb

ReferralWeb, created in 1997 by Henry Kautz and Bart Selman, is a tool to aid the navigation of social networks through referral chaining. Social networks, are groups of people linked by professional, rather than recreational activities [Kautz, Selman & Shah, 97a]. A social network is represented by a graph, where nodes are individuals and links represent direct relationships between them. Optionally, these relationships may be

website last year. It has since been removed and not replaced.

weighted, representing the degree of association between individuals.

ReferralWeb uses social networks to make searches more focused and effective. A social network is valuable because it maps the relationships between people. Thus, an expert will be more inclined to assist someone who is directly connected to him or her, since there are mutual acquaintances.

In ReferralWeb, a person may look for a chain between him/herself and another individual ("What is my relation to Bill Clinton?"); specify a topic and radius to be searched ("What colleagues of colleagues of mine know Japanese?"); or take advantage of a known expert in the field to center the search ("List dessert recipes by people close to Martha Stewart"). Questions are dealt with through referral chaining (navigation of the social network), which experiments have shown to be remarkably effective [Kautz, Selman & Milewski, 96].

The system uses a number of spiders to mine information from the web, building the social network map as it finds relations between people. It uses the co-occurrence of names in close proximity in public documents as evidence of a direct relationship. Documents used to obtain this information were links on home pages; co-authorship on papers; citations of papers; exchanges between people in Usenet and companies' organization charts.

When a user first registers with ReferralWeb, it uses a search engine query to retrieve web documents that include any mention of the user's name. Information Retrieval techniques are used to extract names of other individuals from the same documents. This is recursively done (for the new individuals found) for a couple of levels and the results are then merged into the global model.

Searches for topic areas for expertise are resolved in one of two ways [Kautz & Selman, 98]:

1. A database of expertise is generated by saving all paper titles individuals have written and using the SMART retrieval engine to match those against queries;

2. When a query is submitted to the system, it searches the web for experts, transforming the query submitted into a set of altavista queries (composed of name of person and topic areas.)

An earlier version of the system analyzed email communication to map the social networks. However, that approach had to be discarded because of pressing privacy and security concerns: users were reluctant to give their email logs even after many privacy-related precautions had been taken [Kautz, Selman & Shah, 97b]. This work is closely related to ours. The main difference is that it has no explicit incentive mechanism, relying solely on the social networks for incentive.

VIII.II.II SixDegrees.com

Six Degrees [SixDegrees, 99] is very similar to ReferralWeb, in that it provides a service to assist social network navigation. It's a free website where users can register and enter names of people they know. These people are then contacted by the system and invited to join, entering another batch of people, and so on. In this fashion, SixDegrees builds a large social encompassing all its customers. It then provides tools for users to navigate these relationships: it is possible to send messages not only to your friends, but to your friends friends, etc.

VIII.II.III The Know-who Email Agent

Kanfer [Kanfer, Sweet & Schlosser, 97] has tried to build agents to aid in the search process: the Know-who email agent maps the user's social network by reading trough his/her email messages and tries to determine who is best suited to answer the user's questions. Kanfer points out one of the problems with ReferralWeb agents, namely poor accuracy, and suggests three improvements present in her system.

First, the agent recommends people from within the user's social network who are most likely to be able to answer the user's natural language query, based on the content of email communication with the people in the user's social network. The focus is on email, since it enables a person to easily communicate with many others, greatly increasing their social network. However, since it decreases the number of interactions a person has with others, it provides less opportunity to learn about other people's domains of expertise (usually done through interpersonal conversation or observation of interactions). This makes it harder for a person to know whom to turn to when in need of assistance.

Second, each agent monitors all email messages received by its user and maintains a list of all the others from whom the user has received email. Each person in the network is represented by the cumulative content of email messages sent to the user. When a user enters a natural language query, TFiDF is used to retrieve emails that have been sent to the user that might be related to the query and returning a ranked list of senders who might be able to help. This is the likelihood that a person will be able to satisfy the user's query. After identifying those, the system repeats the process using each of the top-ranked experts as the root of the search, therefore expanding the circle. It repeats the process until a person is found with high similarity to the query. The return response from the system shows the referral chain traversed to get to the recommended expert. It also uses relevance feedback methods to allow users to reformulate queries after the initial results have been provided.

The third improvement over ReferralWeb is using an online dictionary to expand the query. Thus, a query could be expanded to include related terms and improve the accuracy of the results.

The improvements suggested are interesting ones. Email, however, is not so much an indicator of expertise, but rather, one of interest. Additionally, the fact that email has been exchanged does not mean that two people have any sort of relationship or even participate in the same social network: mailing lists, for instance, have hundreds of people who have no link to each other (other than being in the same mailing list.) These users would not necessarily feel compelled to answer questions from a random person, especially if this person doesn't participate actively. We try to avoid using email in our system, in favor of something more specifically work-related.

VIII.II.IV Yenta

Yenta [Foner, 97] is a matchmaking agent, designed to introduce everyone, as opposed to just those who care to speak in public. The system functions in a decentralized manner, with the agents grouping themselves into clusters, which reflect their users' interests. These clusters are then used to perform introductions between people with similar interests.

Foner's point on decentralized systems being more secure and scalable is well taken: we anticipate the possibility of using his proposed architecture [Foner, 99] to decentralize the Expert Finder system. Yenta uses referrals from one agent to another to find relevant peers. Each agent has only local knowledge, but groups of agents self-organize into larger units: the agents form clusters with other like-minded agents and use these clusters to introduce users to each other. It is also possible to send messages to individual users or to groups of users through Yenta.

It derives users' interest profiles from their email and newsgroup messages: it analyzes the different messages and when it notices a collection of similar messages, it stores that as an interest for the user. Two or more users who share an interest form a cluster at the moment their agents discover this similarity. To analyze the messages, it uses an inverse-frequency metric for each word in the document, so that rare words are weighed more heavily than regular ones. It has no mechanism specifically for finding experts or domain model attached to it, so that it may be harder to find someone who has enough knowledge to help. It may also be the case when there is nobody in the network who knows about something, plus there is no incentive mechanism for collaboration, which the Expert Finder has.

VIII.III Information Repositories

VIII.III.I Answer Garden

Answer Garden, [Ackerman & Malone, 90] is a system designed to help in situations where there is a continuing stream of repetitive questions, with some occasional new ones, such as a help desk. Users have access to an ontology of questions, and new ones are forwarded to the appropriate expert.

It is meant to improve organizational memory in a company, where organizational memory is defined as "an organization's ability to benefit from its past experience in responding more effectively in the present" [Ackerman & Malone, 90]. It does so by capturing answers to questions that would otherwise be lost and then providing tools for their later retrieval. There are three key features in the system:

1. A branching network of diagnostic questions: users navigate this structure, answering different questions (which try to determine what general type of problem they have) until they are brought to the end of the path, where a set of previously asked questions and answers resides. One of those might be the user's question.

2. New question routing: whenever a user comes to the end of a path and the question needed isn't there, they can enter a new one, which will be automatically routed to the appropriate expert. The answer is then returned directly to the user who asked it and entered into the database at the point where it originated. When setting up the system, experts define which branches of the tree they are knowledgeable in, and that's how the appropriate expert is determined.

3. Editable branching network: should the experts conclude from users' questions or network usage that the branching network is misleading or

inefficient, they can add to or change the diagnostic questions to improve its usability.

The ontology of diagnostic questions is therefore central to the system and difficult to maintain, and must be managed by a given set of predefined experts, who are hard to find. The questions in the tree are kept simple so that the users have no trouble answering them. The incentive for the expert to use this system is that it eliminates the need to answer many simple questions, since those are taken care of by the system. They are therefore free to concentrate on more interesting problems. For the questioner it is, obviously, to find the answer to their question in some written medium or the person who could answer it. One of the problem with bulletin board systems (which are somewhat similar to this system) is that it is difficult to assess the quality of the answer. Given that this is an environment where groups of experts control the branching network, the answers will be more reliable. Scalability, however, may become a problem: as the system grows, it becomes harder to navigate the question network. Also, the system makes it hard to reflect different points of view or opinions: there may be different opinions among the participants of an expert group and that creates a need for a more flexible structure that would allow all of them to be reflected.

Field studies were conducted [Ackerman, 94], having the X Window system as a domain and users taken from a research group at MIT and a class at the Harvard extension school. These revealed that usage was intermittent, which makes it difficult to schedule the experts' time using the system (assuming that there should always be a human expert present and ready to answer questions.) Some of the participants noted that there was a steep learning curve in finding where the good information was, but those who persevered got good at it. This is probably related to the ontology navigation problem we mentioned above. Apparently there was also some doubt as to what questions were appropriate for the Answer Garden, as users knew these would eventually be entered in the database and didn't want to send questions that might be deemed simple or stupid. They also found that the information database was not large enough for the domain and concluded that either the domain must be made smaller or a greater effort must go into the initial phase of building the network, so that it has better coverage. Another interesting result was that questioners wanted quick answers, and the source didn't matter that much (whether from an expert or from the database.) After a while, users tended to go directly to answer garden instead of looking around for a local (off-line) expert to answer their questions: they trusted the system to provide them with satisfactory answers, through the database or through a remote expert.

At the beginning of the study, many of the experts were concerned about being overwhelmed by too many questions and were therefore given the option to refuse to answer due to their workloads. Some experts used this "way out" almost half of the time, which suggests that there is a need for an "expert redundancy": should one expert not be available, someone else present should be able to provide satisfactory answers. Regarding form, some of the experts provided detailed, formal explanations to the questions, both because they wanted to provide a definite, generic answer that would fit in the database and be useful in several other cases, and because the viewed those answers as a statement of their capabilities in the company or community (experts were not anonymized), which would increase their status with people who did not know them well. Users, however, wanted to find short, easily readable answers and the system's goal was to capture the informal information flows, which this push towards formality of response hindered somewhat. One of the most interesting findings of the study was that questioners, in large part, did not get answers at the right level or length of explanation, which suggests that the assumption that questioners always need their questions answered by experts is false.

A second-generation architecture (Answer Garden 2 [Ackerman & McDonald, 96]) of the system added more flexibility in the determination of experts and an escalation mechanism. In this system, a question is sent not to the answer garden, but to an "escalation agent", which then ships it off initially to a synchronous chat channel corresponding to a work group, hallway or other social grouping and waits for an answer (if there is an expert currently logged in and inclined to answer, the question is answered immediately.) After some time, if the user still doesn't have an answer for his question, the agent routes it to a Netnews bulletin board or other channel with wider distribution. If the question still remains unanswered, the agent keeps going, routing it to a help desk, or to an agent that finds suitable human experts or to agents that search the web. Answer Garden 2 also had mechanisms to enable groups of people to collaboratively build answers and information repositories over time, thus facilitating the question answering process.

Unlike the Expert Finder, the Answer Garden's primary goal is not to connect novices and experts, but to provide an accurate answer to the questioner. The implicit social circle (and how people in the community perceive each expert) seems to provide enough incentive for experts to give good answers to questions asked. However, the experts were predetermined for the first version of the system and it was non-trivial to add new experts to the system and take advantage of the fact that new people are always joining the community and bringing their knowledge (this was modified in a later version.) We try to keep the system as open as possible with the Expert Finder, so that new experts can join (and bring their expertise) at any time.

VIII.III.II Another Expert Finder

At MITRE, a project called Expert Finder has also been developed [Mattox, 98]. In their system, expertise is derived from number of mentions of a term either in corporate communications (newsletters) or in a published resume. Documents that mention the employee's name in conjunction with the keyword being sought are also taken into account. MITRE's Expert Finder works by taking a keyword phrase and passing it to an underlying search engine, which then returns a set of hyperlink pointers to the published documents about the topic. It then returns the names with the most published documents on the subject or that are most frequently mentioned in relation to the topic. These names are then matched against a database of MITRE employees.

This Expert Finder relies on a database of documents published by company employees and an employee database. It is a centralized system, which doesn't allow for inclusion of new expert easily and doesn't provide incentive mechanisms as we do.

VIII.IV Task-Based Recommendations

VIII.IV.I PHelpS

The Peer Help System, or PHelpS [Collins, 97], is an expert system to select peer helpers with whom a worker can interact in the workplace. For each user, a user model is created, containing several types of information. These are used when another user needs help, to aid the system in selecting a knowledgeable, available and appropriate set of helpers to provide the needed information. Besides selecting peer helpers, PHelpS also makes available previous help sessions as cases to be learned from. The concept of peer learning is important, since helpers can assist others better due to their knowledge of the workplace and acting as a peer helper reinforces learning and strengthens task knowledge. With the concept of peer learning, everyone in the organization becomes involved in training, which gives workers a sense of involvement and facilitates teamwork and team building.

PHelpS concerns itself with "just in time workplace training", that is, training that occurs on the job, in the context of a task. It is strongly connected to the task being undertaken by the workers: for each task there is an associated hierarchical set of steps, many of which involve committee assessments. This means that each task may span several days and more than one worker may be responsible for it throughout that time. The first steps in building the system involved analyzing the tasks and constructing task hierarchies for the system.

Another key item in the system is the PHelpS personal assistant: it enables workers to use these task-subtask hierarchies as checklists in recording steps completed within their tasks. When a step in the checklist is causing difficulty, the worker can get help either by browsing previous cases or asking the system to identify peer helpers for this problem. The system selects users from its database who are knowledgeable about the problem area and the task, are available to provide help, have not been recently overburdened with help requests and speak the same language (this was an issue in this system, since it was developed in Canada.) The helpee can inspect the potential helpers' user profiles so as to better choose whom from this list he or she wants to interact and a dialogue with the chosen worker is then initiated.

User models in PHelpS contain information relevant to the tasks (tasks users are able to perform, how well they can perform them in a fine and coarse-grained level, etc.), in addition to more generic information (age, gender, number of times they've provided help, etc) and temporal information (time taken for completion of a step, time taken to learn a task.) Initial knowledge acquisition is done through the filling of questionnaire. After that, the system tracks the user in order to maintain the profile up to date, as the worker checks off steps and tasks competed, the systems keeps a log of that and the time taken for each. The users, however, are ultimately responsible for maintaining their own profiles up to date, telling the system whether they are available to help on a certain topic or not.

PHelpS is an interesting system. Unlike ours, it's highly task-oriented, which allows it to follow a user's work patterns and check to see when he or she gets stuck. The inspectable user profiles is something we've adopted, but the initial requirement that users fill out (and later maintain) their profiles might prove to be a problem, since users might be reluctant to spend much time on such activities. We get around this situation by having the agent periodically update the profiles.

VIII.V Social Capital

The notion of social capital has been around for a while now [SOCNET, 97]. As it is defined by Putnam [Putnam, 93], Social Capital is what makes the difference between a group of people and a community: features of social organizations (such as networks, norms, trust) that facilitate coordination and cooperation for mutual benefit. It is important to note that, in the absence of coordination and mutual commitment, everyone defects and nobody gets anything from the community.

In his study of small Italian cities, he noted that the most successful ones had a strong tradition of civic engagement (voter turnout, newspaper readership, membership in societies and clubs) and citizens in these regions trusted one another to act fairly and obey the law. Community leaders were honest and committed to equality and the networks are organized horizontally (instead of hierarchically.) He found that these civic networks foster generalized reciprocity, that is, a person does something for another at one moment in the expectation that, down the road, the recipient or someone else will return the favor. In these settings, the existence of close social ties facilitates gossip, which is a way of cultivating reputation (a basis for trust in the society.)

We try to use some of these principles in the Expert Finder, through the "Halo Factor" indicator. We allow users to view others' halo factors, in the hope that that will create the necessary reputation for a given person and serve as incentive for others to help when seeing someone who is useful to the community in need of assistant.

Chapter IX: References

[6DOS] – Six Degrees of Separation - http://www.6dos.com/

[Abuzz, 97] - Abuzz, 97 - The Adaptive Knowledge Exchange – Abuzz White Paper – http://www.abuzz.com/home/white_papers.htm

[Ackerman & Malone, 90] – Ackerman, M & Malone, T – Answer Garden: A Tool for Growing Organizational Memory – proceedings of the ACM Conference on Office Information Systems, Cambridge, MA, April 1990

[Ackerman & McDonald, 96] – Ackerman, M. & McDonald, D. – Answer Garden 2: Merging Organizational Memory with Collaborative Help – Proceedings of the ACM Conference on Computer-Supported Cooperative Work (CSCW'96), November 1996, pp.97-105

[Ackerman & Palen, 96] – Ackerman, M & Palen, L., 96 – The Zephyr Help Instance: Promoting Ongoing Activity in a CSCW System – CHI96

[Ackerman, 94] – Ackerman, M – Augmenting the Organizational Memory: A Field

Study of Answer Garden – proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW'94), pp. 243-252.

[Burke, 97] – Burke et al – Question Answering from Frequently–Asked Question Files: Experiences with the FAQ Finder System. – Technical Report TR-97-05, University of Chicago, Computer Science Department, June, 1997

[Collins, 97] – Collins, J.A, et al. - Inspectable User Models for Just in Time Workplace Training – User Modelling: Proceedings of the 6th Int. Conference, Springer, NY, 1997

[Ericsson & Charness, 97] – Ericsson, K & Charness, N. – Cognitive and Developmental Factors in Expert Performance, in Expertise in Context, Feltovich, Ford & Hoffman (eds.), MIT Press, 1997

[Evard, 97] – Evard, M. - "Twenty Heads are Better Than One: Communities of Children as Virtual Experts". PhD Thesis, MIT Media Lab, Cambridge, 1997.

[Experts, 97a] – Experts Exchange – Experts Exchange FAQ - http://www.experts-exchange.com/info/faq.htm

[Experts, 97b] – Experts Exchange – Experts Exchange Gold Site! - http://www.experts-exchange.com/info/gold.htm

[Feltovich, Ford & Hoffman, 97] - Feltovich, P. Ford, K. & Hoffman, R – A Preliminary Tour of Human and Machine Expertise in Context, in Expertise in Context, Feltovich, Ford & Hoffman (eds.), MIT Press, 1997

[Foner, 97] – Foner, L. – Yenta: A Multi-Agent, referral-based Matchmaking System – The First International Conference on Autonomous Agents, Marina Del Rey, CA, 1997

[Foner, 99] – Foner, L. - Political Artifacts and Personal Privacy: The Yenta Multi-Agent

Distributed Matchmaking System – Ph.D. dissertation – MIT Media Laboratory, May 1999

[Gross & McMullen, 82] – Gross, A. & McMullen, P. – The Help-Seeking Process – In Cooperation and Helping Behavior, Theories and Research, Academic Press, NY, 1982

[Grzelak & Derlega, 82] – Grzelak, J. & Derlega, V. – Cooperation and Helping Behavior; An Introduction – Cooperation and Helping Behavior, Theories and Research, Academic Press, NY, 1982

[Ishida, Nishida & Hattori, 98] – Ishida, T., Nishida, T. & Hattori, F. – Overview of Community Computing – In Community Computing: Collaboration over Global Networks, Ishida, T., ed., John Wiley & Sons, 1998

[Jessup, 97] – Jessup, L. (ed.) – Enterprise-wide Brainstorming on the Web: The case of Experts-Exchange – Section: Profiles of Innovative Groupware Uses – http://php.indiana.edu/~ljessup/gwcent4.html

[Lemay & Perkins, 97] – Lemay, L. & Perkins, C. – Teach Yourself Java in 21 Days, second edition – Sams, 1997

[Kanfer, Sweet & Schlosser, 97] – Kanfer, A., Sweet, J. & Schlosser, A. – Humanizing the Net: Social Navigation with a "Know-Who" Email Agent – Proceedings of the 3rd Conference on Human Factors and the Web – Denver, CO, 1997 - http://www.uswest.com/web-conference/proceedings/kanfer.html

[Kautz & Selman, 98] – Kautz, H. & Selman, B. – Creating Models of Real-World Communities with ReferralWeb – Recommender Systems: papers from the 1998 AAAI workshop; Technical Report WS-98-08; AAAI Press, Menlo Park, California, 1998

[Kautz, Selman & Milewski, 96] – Kautz, H., Selman B. & Milewski, A. – Agent Amplified Communication – Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96), Portland, OR, 1996

[Kautz, Selman & Shah, 97a] – Kautz, H., Selman, B. & Shah, M. – ReferralWeb: Combining Social Networks and Collaborative Filtering – Communications of the ACM vol 40, no. 3, March 1997

[Kautz, Selman. & Shah, 97b] – Kautz, H., Selman, B. & Shah, M. – The Hidden Web – AI Magazine, Summer, 1997

[Maes, 94] – Maes, P. – Agents that Reduce Work and Information Overload – Communications of the ACM, July 1995, Volume 37 (7)

[Mattox, 98] – Mattox, D., Maybury, M. & Morey, D. - Enterprise Expert and Knowledge Discovery – MITRE Corporation - 1998

[Putnam, 93] - Putnam, R.D. - The Prosperous Community: Social Capital and Public Life, The American Prospect # 13 (1993) - http://epn.org/prospect/13/13putn.html

[Salton, 88] - Salton, G. - Automatic Text Processing: The Transformation, Analysis and Retrieval of Information by Computer. - Addison-Wesley, Reading, MA, 1988.

[Seifert, et. al] – Seifert, C.; Patalano, A; Hammond, K. & Converse, T. – Experience and Expertise: The role of Memory in Planning for Opportunities in Expertise in Context, Feltovich, Ford & Hoffman (eds.), MIT Press, 1997

[SixDegrees, 99] – Six Degrees About Pages - http://www.sixdegrees.com

[SOCNET, 97] – Archives of the SOCNET (Social Networks) mailing list - http://www.heinz.cmu.edu/project/INSNA/arc_soc/capital.html

[Sun, 98] – Sun Microsystems - Java 2 Platform, Standard Edition, v1.2.2 API Specification - http://java.sun.com/products/jdk/1.2/docs/api/overview-summary.html