Agents to Assist in Finding Help

Adriana Vivacqua and Henry Lieberman
Media Laboratory
Massachusetts Institute of Technology
Cambridge, MA 02139 USA
+1 617 253 0315
lieber@media.mit.edu

ABSTRACT

When a novice needs help, often the best solution is to find a human expert who is capable of answering the novice's questions. But often, novices have difficulty characterizing their own questions and expertise and finding appropriate experts. Previous attempts to assist expertise location have provided matchmaking services, but leave the task of classifying knowledge and queries to be performed manually by the participants. We introduce *Expert Finder*, an agent that automatically classifies both novice and expert knowledge by autonomously analyzing documents created in the course of routine work. Expert Finder works in the domain of Java programming, where it relates a user's Java class usage to an independent domain model. User models are automatically generated that allow accurate matching of query to expert without either the novice or expert filling out skill questionnaires. Testing showed that automatically generated profiles matched well with experts' own evaluation of their skills, and we achieved a high rate of matching novice questions with appropriate experts.

Keywords
Expertise location, agents, matchmaking, Java, help systems.

INTRODUCTION

Meet Jen: Jen has been in the computer business for a while, doing systems analysis and consulting. She has wide experience in Cobol, mainframes and database programming, but little experience in Java, which her company has now decided to use.

Meet David: David is a hacker. He started programming at the age of 15, and 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 and perform transactions through their computers. The system uses database manipulation and a graphical user interface.

Given that Jen is a novice Java programmer, she has a hard time 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…

The hard way

Jen doesn't know what objects are available to connect her server side routines and database with the front end. She asks around the office, but nobody is familiar enough with the Java language to navigate JDBC objects and connections. She manages to access the database, defines the functionality that should be included in the front end, and now needs to know 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 testing the objects is a tedious and slow process. 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. The book 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.

She wades around the newsgroups, reads their FAQs, and posts a question. Disappointingly, she gets no answers. She finds that most of the newsgroups are tight communities where people tend to get off topic or carried away. She subscribes to a few mailing lists, but traffic is too high. 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. Sarah's boyfriend, David, is experienced in Java. Jen reluctantly sends him an email, to which David replies with a brief explanation and pointers to some websites about JDBC.

Enter the Expert Finder

Let's see how the same scenario goes with our Expert Finder system. Instead of asking around the office, Jen goes to her Expert Finder agent and enters a few keywords.

Expert Finder 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 the programs she wrote while studying with the “Learn Java in 21 Days” [5] book. Expert Finder verifies what constructs she has used, how often and how extensively, and compares those values to the usage levels for the rest of the participating community to establish her levels of expertise. Jen can see and edit her profile on the profile-editing window, and decides to publish all of it. 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.

java.io	10	Novice
java.util	15	Novice
System	20	Novice
elementAt	5	Novice
println	20	Novice

Table 1: Jen’s areas and levels of expertise

The agent communicates with other agents calculating their “suitability” by verifying which libraries and classes they know how to use. It picks out David (Table 2), because he has used the “java.sql” library and its objects.

Area	Usage	Expertise Level
java.io	46	Intermediate
java.util	45	Intermediate
Connection	11	Advanced
InputStream	5	Intermediate
CallableStatement	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.

His expertise is higher, but not too distant from Jen’s. Jen takes a look at David’s published profile, checks his “halo factor” (an indicator of how helpful he is to the community), and sends him a message:

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?

Thanks,

Jen

David verifies, based on Jen’s “halo factor”, that Jen is a new user and decides to answer her question:

Hi Jen,

To call stored procedures you should use a Callable Statement, which can be created with the *prepareCall* method of the Connection class.

Here’s a little snippet 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);
```

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

David

With Expert Finder, Jen obtained David’s help much faster than she would have otherwise.

Approach

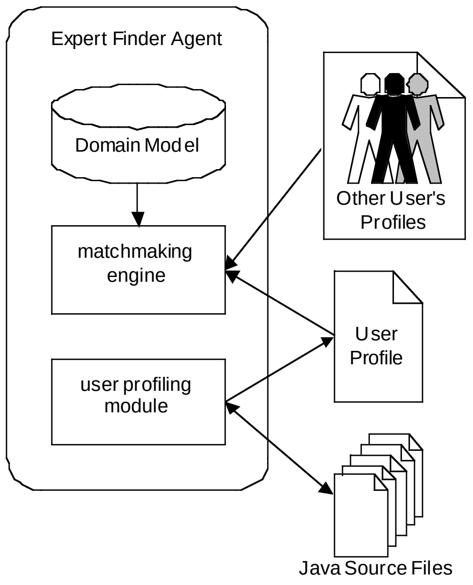


Figure 1: An agent’s Internals: Each agent has (1) a profiling module, which builds the user’s profile from his/her Java files; (2) a matchmaking engine, which consults and compares other user’s profiles and (3) a domain similarity model, used for matchmaking purposes

Figure 1 shows one agent’s internal structure. It is important to note that there are no specialized agents for experts and novices. It often happens that a person might be an expert in one area and a novice in another.

Domain Similarity Model

Our system uses a *similarity model* for the Java domain, because an expert whose knowledge lies in a more general or more specific category or related topic to the novice's requirements might still be a good candidate to provide help. In a sophisticated domain like Java programming, there are many overlapping relationships between the knowledge elements. Rather than burden users with the task of manually browsing subject category hierarchies, and judging relevance, we move that task onto the agent.

Even if the agent is not perfectly accurate in its similarity assessment, the agent's model constrains the search space enormously and results in more relevant recommendations. We also provide browsers and editors for the domain model, and for user profiles, allowing any deficiencies in our prior knowledge to be corrected manually.

The Java Programming Domain

Constructs in Java are hierarchically structured into classes and subclasses and organized in packages according to purpose or usage. Many classes also provide an extra hint: the "See also:" entry, which lists related classes, methods or packages. We assigned 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.

- **Sub/Superclass relationships**: a subclass is fairly similar to its superclass (inheriting methods and properties), but a superclass is less similar to its subclass, since the latter may contain resources not available in the former. For example, the class *Container* is a subclass of class *Component*: it inherits 131 methods and 5 fields. However, *Container* also defines 52 of its own methods. Code: *SUB* or SUP.
- **Package coincidence:** Packages group classes by what they are used for. Package *java.awt* contains classes used for graphic interface construction, such as buttons, list boxes, drop-down menus, etc. A person who knows how to use these classes is someone who knows how to build graphical interfaces. Code *PAK*.
- **"See also" entry:** this is a hint which links to other classes that might work similarly or share a purpose. 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 parameterized: the model holds the different relations, not the numbers.

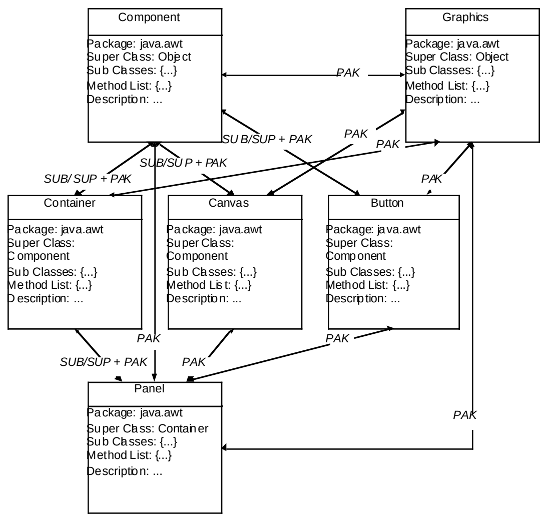
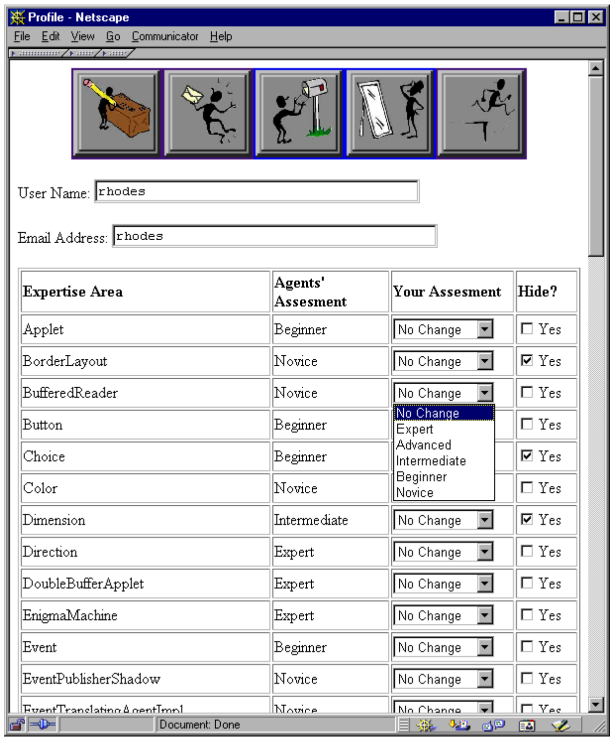


Figure 2: Similarity model for the Java domain (partially shown.)

Building Profiles

Automatic profiling is important, given that, in general, people dislike filling long forms about their skills. An automated method also reduces the possibility of inaccuracy due to people's opinions of themselves. Another advantage is that automated profiles are dynamic, whereas people rarely update interest or skill questionnaires. 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 (novice - beginner - intermediate - advanced - expert) and a flag noting whether or not this information is to be disclosed. Hidden information will still be used in calculations of expertise for a given query. A user might change his or her profile at any time.



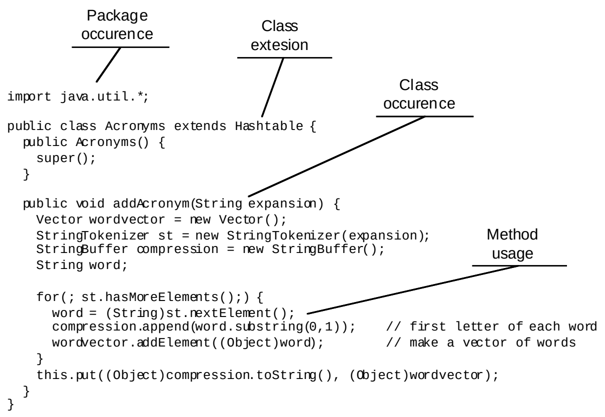
User Name: rhodes

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 3: 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.

Assessing a user's areas and levels of expertise is done through analysis of his or her Java source files and parsing them, analyzing:

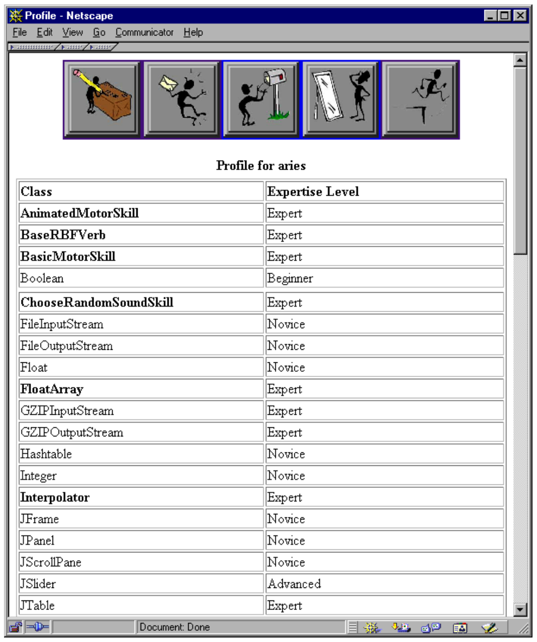


```
import java.util.*;

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 4: Example code and items analyzed in it.

- **Libraries**: which libraries are being used? How often? Libraries are declared once, usually at the beginning of a file.
- **Classes**: which classes are used? How often? Classes are declared, instantiated and used throughout the file. Classes can also be subclassed, which indicates a deeper knowledge of the class. 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.
- **Methods**: knowing which methods are being used helps us further determine how much he or she knows about a class: Are only a few methods used over and over again? How extensively is the class used?

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) [9], 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

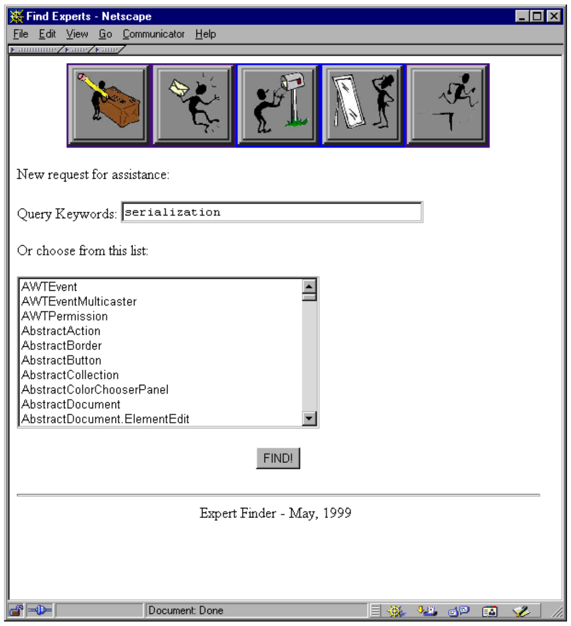
Figure 5: Viewing other users' profiles: the items in bold represent classes that have been subclassed. "Hidden" classes are not shown.

Matching Needs and Profiles

Given a query, related topics are taken from the model and added to the query, thus expanding it. It is then compared to other users' profiles. A query can be formulated as:

- **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.
- **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.
- **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 6. If a user selects items from the list, it is reasonable to assume that he or she 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 (packages) is more important. If the user uses a combination of both, both relations can be used, although functional similarity takes 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.)



New request for assistance:

Query Keywords: serialization

Or choose from this list:

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

FIND!

Expert Finder - May, 1999

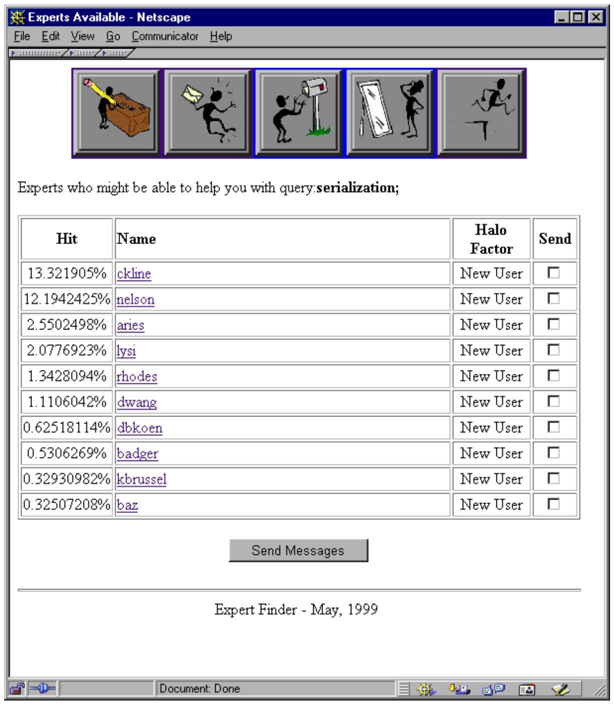
Document: Done

Figure 6: Query screen – a user may choose an item from the list or enter keywords.

A match is made by first finding similar topics in the domain model. The agent then goes on to contact other agents, computing a vector match between its user's needs and other users' expertise. The agent returns a list of potential helpers. We compute "fitness values" for all of the users, including the questioner. We then take the *n* with closest (but higher) fitness values. 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. First, because the topmost expert is most likely to be unavailable or uninterested in novice questions. But, more importantly, experts and novices have different mental models, as noted by [3] so we are more likely to bring together two people who have similar mental models.

Figure 7 shows the screen where users can view a response to their query, listing the experts available.


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	☐

Send Messages

Expert Finder - May, 1999

Document: Done

Figure 7: Expert list screen – experts are ranked by appropriateness to a given query.

Incentives

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 (the *halo factor*). The halo factor of a person is the percentage of questions answered from those received ([Qa/Qr]*100). It is displayed every time a person sends or answers a question, motivating both the questioner and responder. 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 don't want to 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.) As the system keeps track of questions sent and received, we can more evenly distribute questions when there are multiple experts available.

Interface Overview

A button bar (Figure 8) on the top of each page gives each user the options: making a new query, viewing responses, viewing questions, editing the profile and logging out.

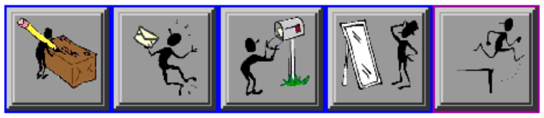
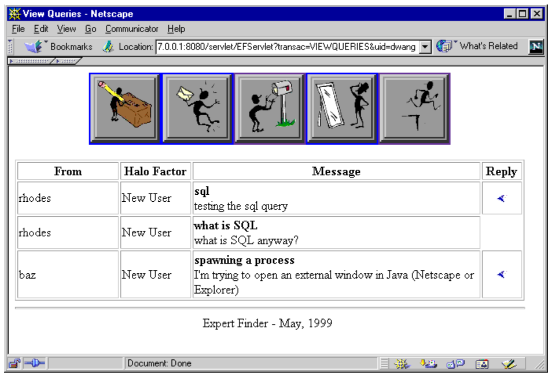


Figure 8: Expert Finder button bar. Left to right: Query, View Responses, View Questions, Edit Profile, Logout.

A user can edit his or her profile on the profile-editing screen, shown previously in Figure 3. The queries are submitted to the system via the query interface, Figure 6.

The results of the query are then shown in the result screen, Figure 7. 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. Still on the result screen, the user can select experts and click "Send Message" to go to the message composition screen .

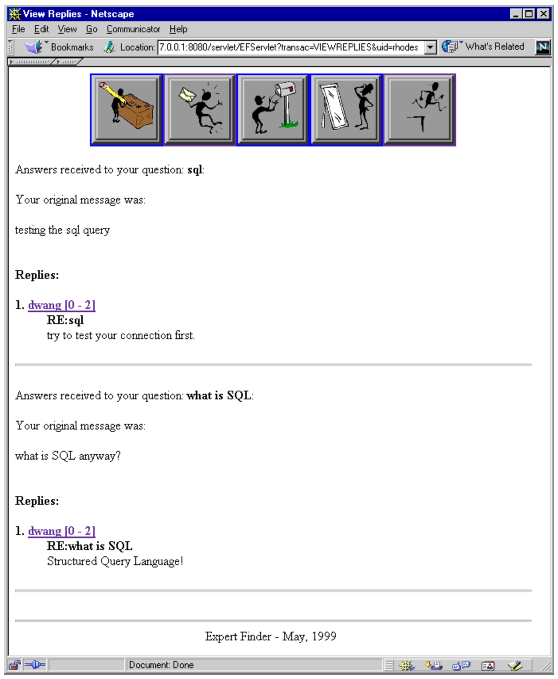
An expert can view questions sent to him or her, and compose a reply, Figure 9. He or she can view responses as on Figure 10.


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 9: 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 10: Viewing answers received to one's questions.

Evaluation

As an evaluation for this work, we built a prototype system, generated profiles for 10 users, and ran 20 queries through the system. We independently determined whether the experts suggested by the system would be able to answer those questions through a questionnaire. Questions 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?").

Possible answers from the experts were "I can answer", "I couldn't answer this" and "Not flat out, but I would know where to look". 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.

Profiling

To test how well the profiling module worked, we generated profiles for 10 users and had them edit them. We then took the original and edited profiles and checked to see how many items were altered and by how much. Users' profiles are kept in files divided into:

- **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.
- **Agent's calculations:** this is the expertise level the agent calculated for the user.
- **User values:** User's corrections to the agent's calculations, and values to be hidden from other users.

On average, it seems users edited about 50% of their profiles. The number of changes ranged from 9% for the least altered profile to 63% for the most altered. About one third of all changes were decreases.

On commonly used classes such as *Hashtable*, users felt they were very knowledgeable even though their profile indicated otherwise. Many experts were using this class and what we calculate for the profiles is what percentage of the total usage belongs to each of those experts. 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.

The decreases for the most part happened when there was only one user who used a given construct, and was therefore deemed the expert. If nobody else is using this class, the user is responsible for 100% of its usage in the community.

31% of changes were 1 step changes (for instance, novice to beginner), 33% 2 step changes, 26% 3 step changes and 10% 4 step changes. These numbers seem to indicate that the agent's calculations weren't so far off the mark.

Matchmaking

Overall, the system performed well, always placing at least one expert who had had said he or she 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.)

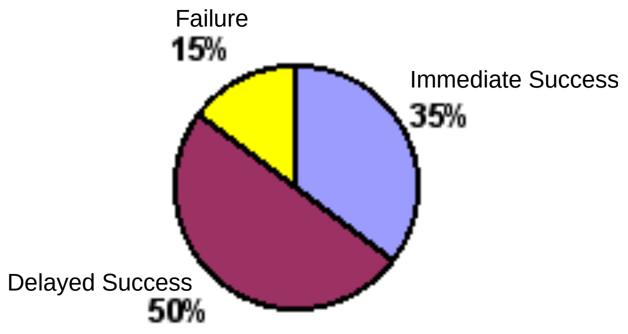


Figure 11: Distribution of Success/Failure cases.

The system performed better for people with at least a little knowledge. Since the system recommends people at a level of expertise close to that of the questioner, if the questioner had little or no expertise, the system did not always recommend people well suited to answer.

For queries that were more specific, the system performed well. Taking the top 3 experts found (not recommended) for specific queries, we have 52% said they could answer the question, 19% said they could look it up and 29% said they could not. Analyzing the failure cases, 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.

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.

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.

Future Work

Profile Building

More accurate profile building is a major area for future work. Accuracy can be improved by enlisting more sources of information and taking into account other factors such as history. We could perform more complex code analysis, which might reveal more about one's programming style, abilities and efficiency. We could also use such techniques as collaborative filtering to rate expertise.

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. As Seifert [10] notes, expertise comes with experience, and memory plays an important part.

Making Expert Finder more proactive

The most immediate next step for Expert Finder would be making it more proactive. A context-aware agent built directly into the development environment could try to figure out the user needs help by watching error messages as he or she writes the program. It could also be done by detecting when the user goes to the help system.

The agent could also 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 samples 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.

Related Work

Information Marketplaces

Experts-Exchange

Experts-Exchange [4] uses a predetermined expertise directory, under which questions and answers are posted. It uses a credit system to provide incentive. 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.

Referral Systems

ReferralWeb

In ReferralWeb [7] a person may look for a chain between him/herself and another individual; 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"). The system uses the co-occurrence of names in close proximity in public documents as evidence of a relationship. Documents used to obtain this information were links on home pages; co-authorship on papers; etc.

ReferralWeb lacks a domain model or automatic profile construction, but Expert Finder might also benefit from ReferralWeb's social network techniques, since people prefer to ask questions of others who have pre-existing social relationships with them.

Yenta

Yenta [5] is a matchmaking agent that derives users' interest profiles from their email and newsgroup messages. Yenta aims to introduce people who share general interests rather than matching for a specific question or topic, and again has no domain model. Yenta is notable for its fully decentralized structure, which also could benefit Expert Finder.

Information Repositories

Answer Garden

Answer Garden, [1] is a system designed to help in situations such as a help desk. It provides A branching network of diagnostic questions through which experts can navigate to match the novice's question. A similar question already in the network may yield the answer, or a new Q&A pair can be saved for future reference. The network can also be edited.

Answer Garden and similar systems look for the contents of the answer rather than the expert, which is harder in some cases, and forgoes the ancillary advantages of locating an expert who might serve as a resource in the future.

Another "Expert Finder"

A MITRE project also called Expert Finder [8] derives expertise estimation from number of mentions in Web-available newsletters, resumés, employee databases and other information. It is a centralized system, which doesn't allow for inclusion of new experts easily and doesn't provide incentive mechanisms as we do. Recent versions are incorporating more proactive elements, bringing it closer in spirit to Expert Finder.

Task-Based Recommendations

PHelpS

The Peer Help System, or PHelpS [2] tracks users who are doing step-by-step tasks, and if a novice runs into difficulty, it matches them with another user who has successfully completed the same or similar sequence of steps. Unlike our system, 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.

Conclusion

We have presented Expert Finder, a user-interface agent that assists a novice user in finding experts to answer a question by matchmaking between profiles automatically constructed by scanning Java programs written by both the novice and the expert. Tests show that the agent does reasonably well compared to human judgment, and Expert Finder obviates the need for skill questionnaires that are daunting to user and hard to maintain over time.

REFERENCES

1. [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
2. [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
3. [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
4. [Experts, 97a] – Experts Exchange – Experts Exchange FAQ - http://www.experts-exchange.com/info/faq.htm
5. [Foner, 97] – Foner, L. – Yenta: A Multi-Agent, referral-based Matchmaking System – The First International Conference on Autonomous Agents, Marina Del Rey, CA, 1997
6. [Lemay & Perkins, 97] – Lemay, L. & Perkins, C. – Teach Yourself Java in 21 Days, second edition – Sams, 1997
7. [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
8. [Mattox, 98] – Mattox, D., Maybury, M. & Morey, D. - Enterprise Expert and Knowledge Discovery – MITRE Corporation - 1998
9. [Salton, 88] - Salton, G. - Automatic Text Processing: The Transformation, Analysis and Retrieval of Information by Computer. - Addison-Wesley, Reading, MA, 1988.
10. [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