Agents for Expertise Location

Adriana S. Vivacqua

Software Agents Group, MIT Media Lab
20 Ames St., 305F
adriana@media.mit.edu

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. We have chosen to tackle this problem. Our approach to addressing the problem of finding people who can help is to use software agents to assist the search for expertise and mediate the information exchange process. Issues of availability, user profiling, privacy and incentives are involved. We chose the Java Programming domain for initial implementation and testing of the system. Other researchers have taken stabs at this problem, most without the use of agent technology. We are building agents, called Expert Finders, to help people find help.

Introduction

A person may find him/herself in need of help even when performing routine tasks. Very often, help is hard to find. The person ends up doing of research in books, when it might have been more productive to talk to someone with the right expertise who could help solve the problem. Students, for example, often have to deal with courses that others have taken before. Another student would be a potential source for help when the professor was unavailable. The following scenario illustrates a common situation:

> Jen is a novice Java programmer. She has difficulty understanding the language, and learning all the existing packages and classes. However, she knows she needs to parse some text. She spends some time going through the manuals, online documentation and mailing list archives. She tries posting to a newsgroup but gets no response. While talking to a co-worker, she mentions her problem, to which he replies that using the StringTokenizer class would make it easier and shows her a piece of code he had previously written that uses this class.
>
> Now, imagine Jen had an agent she could ask to find someone who would be able and willing to help her. The agent then goes out on the network and finds people who would be well suited to her problem, given their areas and levels of expertise. It returns a list of people she might want to talk to. She contacts them, they tell her what class to use and show her some code to do it.

We're addressing the problem of *finding an expert who can assist a person given the knowledge they both have*. Often there is no expert close at hand, which makes the task harder. By extending the search to a larger, networked community, more experts become available, even if there's no physical proximity. The Expert Finder is our attempt at tackling the issues present when trying to obtain help remotely. We have constrained our domain to Java programming, but are trying to keep the project as domain independent as possible.

Assisting the Help-Seeking Process

When a person doesn't know what to do, he or she usually looks for someone who 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 created 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); and
- adjusting its own profiling mechanisms according to user feedback

We have been experimenting with an initial implementation and some of these ideas. We've gotten a better view of how the system should work after implementing the first version.

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 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. This is an interesting domain, since it is reasonably complex and allows us to build a useful tool for actual programmers to use and test the agents.

Approach

Given the aforementioned problems and scenarios, we have envisioned 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 environment to encourage collaboration over the network, thus allowing a community to leverage from the knowledge existing inside people's heads.

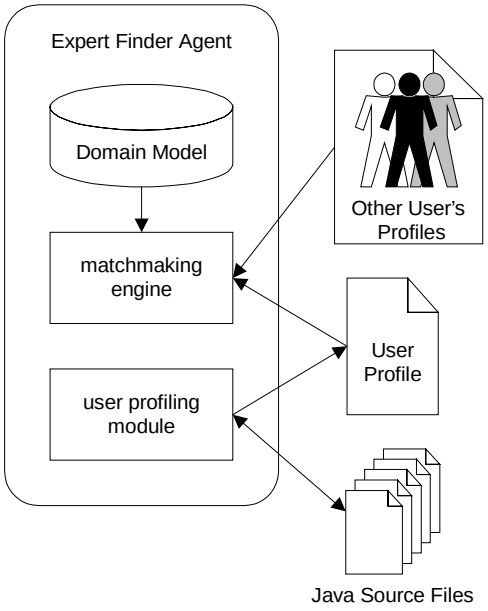

Figure 1: Agent's Internals: Each agent has (1) a profiling module, which builds the user's profile from his/her Java files; (2) a matchmaking engine, which consults and compares other user's profiles and (3) a domain similarity model, used for matchmaking purposes

Each agent's goal is *to determine the best expert in the existing community to help a person.* Each user has an agent that has knowledge about his or her areas and levels of expertise, taken 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 1 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 we want all agents to be the same, 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 moment and a novice in the next), 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. We want to create an environment that also allows this to happen.

Going back to the previous scenario,

> Jen's agent reads through her Java source files, verifying that she knows a lot about graphics procedures, a little bit about string manipulation but not much about file processing or parsing. When she asks for help with string parsing, the agent starts contacting other agents, asking for their user's profiles. It finds two possible candidates, people who are available at the moment and have the necessary knowledge. Steve knows a lot about file processing and I/O, and some about parsing. Rick knows a lot about parsing files, strings and about the language in general. The agent then suggests both men, putting Steve first, given that he is at a level of expertise closer to Jen's.

Building Profiles

An important part of the system is having profiles that accurately reflect what the users know and their levels of expertise. 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 Expert Finder agents. These profiles are recalculated periodically.

The agent creates the user's profile by reading through the user's Java source files and parsing them, analyzing what classes, methods and libraries are used and comparing 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 on each. Expertise level is currently determined by taking the number of times the user uses each class and dividing by the overall class usage. We're now incorporating average class usage 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. We are also considering using software metrics techniques to evaluate the quality of the code.

Matching Needs and Profiles

A domain knowledge model is used to identify similarities between different areas or constructs, so as to allow the agents to recommend someone even when an expert in one particular field isn't available. We expect to be able to give suggestions of people that may be able to answer questions given the related knowledge they have.

The domain model was built using the Java language model and documentation itself. Thus, we have taken relationships between classes and used these to determine levels of similarity between classes. We currently have a model based on sub/superclass relationships and package coincidence that maps how classes relate to each other. It is similar to a keyword dictionary, except it has weights. For instance, one of the entries might read:

BufferedInputStream -- BufferedOutputStream (5); BufferedReader (7)

Meaning that using the BufferedInputStream class is similar to using the BufferedReader class (more so than using the BufferedOutputStream class). We are now expanding our model (still using the documentation) and taking into account the "see also" links provided by Sun (good indicator of related topics) and parsing the text descriptions to encompass more abstract notions such as graphics or math.

A match is made by first taking the questioner's query and finding similar topics in the domain model. With this set of topics, 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. The agent returns a list of potential helpers, preferably people who are around the same level of expertise as the questioner. We believe that the best person to help is not always the topmost expert, but someone who knows a bit more than you do.

Motivation and Incentives

The issue of motivation is crucial for the system to function. Some ideas that have been explored involve "commercializing" information, as in electronic commerce systems. Several systems give users "credits", with which they can buy answers to their questions [6DOS], [Experts], [Abuzz]. However, some studies have shown that people will answer questions for personal reasons, such as the "being helpful" or "showing off" [Ackerman & Palen, 96]. Rather than jumping into the marketplace bandwagon straight away, we're turning to psychologists' works to learn what motivates prosocial behavior. According to Grzelak [Grzelak & Derlega, 82], cooperative behavior is behavior that maximizes both the individual's and other's interests. We're trying to create an environment where people cooperate more freely, creating a community for knowledge exchange.

We assume that every person who uses the system is also willing to help others when necessary. One of the interfaces shows users on the answering side how much the questioners have volunteered to the community. We hope this will motivate experts to help people who also contribute to the system, and create a sense of Social Capital as understood by Putnam [Putnam, 93], where people give and take from the community, rather than from individuals. The agents are able to determine how helpful a person is in the community by tracking their interactions (questions received and answered). This information is shown when necessary. We don't want to create a sense of reward for the information; we want it to be freely volunteered.

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. 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.

Architecture

For simplicity's sake, we're using a registry server, which holds a list of all active agents. Each agent registers its presence with the server when it first starts. This server also holds the most recently updated domain model, which can be transmitted to the agents periodically. In addition, it keeps track of "global values" such as number of users and overall class usage (to calculate the profiles) and message exchange in the system.

We are aware that this method isn't optimal, but we're trying to concentrate on other problems first. In the near future, we may switch to using a totally decentralized architecture such as Yenta's [Foner, 97] or ReferralWeb's [Kautz, Milewski & Selman, 96], although we don't want to rely on the existing social networks too much.

As it is now, the client side holds the user's profile and a copy of the domain similarity model. It is also able to 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 is illustrated in Figure 2.

Related Work

We've been reading from many different fields for this work. There are interesting projects covering several areas, which relate to ours. Work on the field of community computing [Ishida, Nishida & Hattori, 98] is just starting, and seems very relevant. Works on this field concern mostly building and/or supporting networked communities, which is what we're trying to do: build communities of experts who will exchange information, leveraging the knowledge existing in the community.

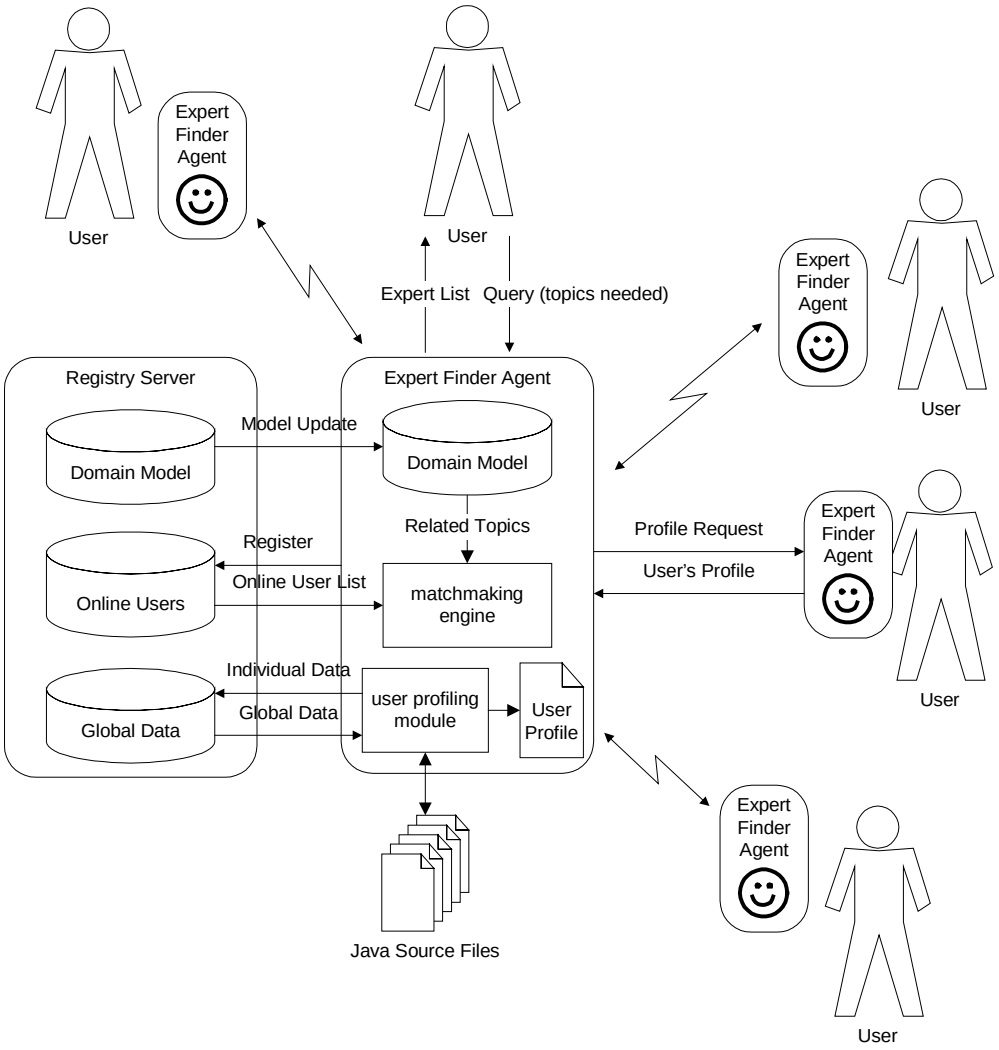

Figure 2: General Architecture: A user queries his/her agent. The agent consults the registry server and obtains a list of all currently active agents. It then goes on to contact each of these, requesting their users' profiles. It takes the profile, the query, related topics and its own user's profile and looks for good matches.

Existing (commercial) on-line systems have taken the approach of using directories, and have experts classify themselves in those. In Experts Exchange [Experts] and Beehive [Abuzz], novices can go to these directories and send messages to groups of people in given categories. In these cases, novices don't know who will be answering their questions or their level of expertise. Plus, these systems use an economic incentive model, and we're proposing alternative incentive models.

The project most similar to ours is probably henry Kautz's ReferralWeb [Kautz, Milewski & Selman, 96]. Referralweb uses papers written, citations and email to determine the user's areas and levels of expertise. It then navigates the established social network to find an expert. We are looking to use different information sources (such as source code) and explore the possibilities in these. We are also trying not to rely so much on the existing social networks, opening up possibilities for creation of new communities. And, in certain cases, there won't be an expert in your community who will know what you need, so there will be a need for outside experts to come in. we are looking for ways to facilitate this process.

The Answer Garden [Ackerman & Malone, 90], [Ackerman, 94] is a system that gathers knowledge from questions and answers sent to experts, forming an organizational database. In this system there is and ontology used to categorize questions and a predefined set of experts that is hard to change. Users must navigate the ontology of questions to get to the information, and the experts are at the "leaf nodes", when there is no information in the structure, questions may be sent to an expert.

Yenta [Foner, 97] is a matchmaking agent, which derives users' interest profiles from their email messages and tries

to find other people with similar interests through referrals, performing introductions and trying to form communities. 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 also uses email as a basis to building profiles (which we believe is not good enough) and has no indication of expertise levels.

PHelpS [Collins, 97] is an expert system to select peer helpers with whom a worker can interact in the workplace, allows the users to ask questions on tasks. It is, however, a closed system, in that only workers of that system will use it, and it has a very specific task model that enables it to find the experts, by tracking which users have done that task before.

As far as we know, only one system has tried to build agents to aid in the search process [Kanfer, Sweet & Schlosser, 97]. 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 from that. Email, however, is not always a good indicator of expertise, but rather, of interest.

We've also been reading from the psychology field, about cooperative and helping behavior, hoping to find incentive mechanisms for our system that not the commerce-like "credits" systems. Some of the relevant papers describe what is and why collaborative behavior comes about [Gross & McMullen, 82], [Grzelak & Derlega, 82].

There are many other systems that are somehow related to our system, especially in the CSCW (Computer Supported Collaborative Work) domain. To our knowledge, few have used agents for this purpose before and even less concern themselves with building communities and fostering collaboration. We hope to keep working with this in mind and trying to find ways to create groups and exchange knowledge over a network.

Conclusion

There is still work to be done on our initial implementation, as we have mentioned in previous sections. We expect to soon finish this version and deploy it for testing purposes. We believe this is an important problem, and one that exists in several different domains. We are confident that using agents for this is purpose is a worthy approach, since it opens up many possibilities (decentralizing the system, automatic profiling, negotiation behaviors, scheduling etc). We have received very positive feedback from people in the programming community and other domains. We will proceed with our line of research and hope to have a new version out soon.

References

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

[Abuzz] - 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. In proceedings of the ACM Conference on Office Information Systems, Cambridge, MA, April 1990

[Ackerman, 94] – Ackerman, M – Augmenting the Organizational Memory: A Field Study of Answer Garden. In proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW'94), pp. 243-252.

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

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

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

[Experts] – Experts Exchange homepage - http://www.experts-exchange.com/

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

[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

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

[Kautz, Milewski & Selman, 96] - Kautz, H. Milewski A. & Selman, B – Agent amplified Communication. In Proceedings of AAAI-96

[Putnam, 93] - Putnam, R.D. - The Prosperous Community: Social Capital and Public Life. In 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.