ABSTRACT

WENTAO MO. A Referral-Based Recommender System for E-Commerce (Under the direction of Dr. Munindar P. Singh).

The thesis is intended to develop the technology and infrastructure to allow people to share knowledge with and learn from each other. A special kind of multiagent system, called *multiagent referral system (MARS)*, is proposed. In MARS, each user is assigned a software agent, and software agents help automate the process of expertise location by a series of "referral chains."

Unlike most previous approaches, our architecture is totally distributed and preserves the privacy and autonomy of their users. These agents learn models of each other in terms of expertise (ability to produce correct domain answers), and sociability (ability to produce accurate referrals).

A Referral-Based Recommender System for E-commerce

Master Thesis

Wentao Mo
Department of Computer Science
North Carolina State University
wmo@unity.ncsu.edu

Thesis Advisor: Dr. Munindar P. Singh

Committee Members:
Dr. Munindar P. Singh (Chair)
Dr. R. Michael Young
Dr. Mona Singh

BIOGRAPHY

Wentao Mo is a graduate student in Department of Computer Science, North Carolina State University. In 1989, he got his Bachelor of Science in Meteorology from Nanjing Institute of Meteorology, P. R. China. In 1992, he got his Master of Science in Climatological Numerical Modeling from Chinese Academy of Meteorological Science, Beijing, P. R. China. After his graduation, he worked in the Multimedia Center at Beijing Institute of Meteorology. His responsibilities were software development based on CD-ROM and Intranet, which aims at meteorologists' on-job training across China. In his graduate study from January 2000 at NC State, Wentao Mo focused on building a Multiagent Referral System (MARS).

ACKNOWLEDGEMENTS

During the development of this thesis project, I've received assistance from many people. First, I would like to express my gratitude to my academic advisor, Dr. Munindar P. Singh who introduced me to the field of multiagent system, supported my graduate work and sometimes pair-programmed with me. Special thanks to Dr. R. Michael Young and Dr. Mona Singh who kindly agreed to serve on my thesis defense committee.

Also I would like to express my gratitude to IBM Agent Building and Learning Environment (ABLE) team, especially to Dr. Joseph P. Bigus and Jeff Pilgrim, who provided discussion and support on the regular basis when I was integrating ABLE into MARS. Many thanks to Bill Shannon of Sun Microsystems, Inc., who provided me with technical support when I was implementing MARS's communication components by deploying Sun's JavaMail API.

I would like to thank other students in the Interaction, Media, and Commerce Lab: Bin Yu, who provided me with intensive valuable discussion and collaboration on MARS's prototype development; Richard Lyles from IBM, who directly contributed to the coding of the MARS learning component; Jie Xing, Zhengang Cheng and Pinar Yolum, who always gave me good suggestions.

Finally, many thanks are due to my family for their love, encouragement and patience.

This thesis is based upon work supported by the National Science Foundation under Grant number 0081742 and by a University Partnership Award from IBM Corporation. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation or IBM.

Contents

List of Figures

List of Tables

1. Introduction

Recommender systems are a category of software that make personalized recommendations of goods, services, and people. Among the initial applications were information spaces such as Usenet News [Konstan *et al.*, 1997], video recommendations [Shardanand & Maes, 1995], and Web pages [Balabanovic & Shoham, 1997, Terveen *et al.*, 1997]. Recommender systems are also used by many e-commerce sites (e.g. Amazon.com), and are finding their way to serious business applications [Schafer *et al*., 1999]. However, most recommender systems generate recommendations based on anonymous opinions. This may be fine if the recommendation is about something minor, such as which book to buy or which movie to see. When a person is making critical decision about his life or his business, it is unlikely for him to make the decision based on anonymous opinions. Instead, he wants advice from an expert among his friends. If he does not personally know such an expert, he wants a personal reference to such an expert via a series of trusted friends and colleagues – "circles of trust".

Referral systems have been known since late 1980s [Huhns, *et al*., 1987]. Recently, there has been a resurgence of interest in referral system, such as ReferralWeb [Kautz, *et al*., 1997], ContactFinder [Krulwich & Burkey, 1996], and Knowwho [Kanfer, *et al*., 1997]. Unfortunately, the existing referral systems suffer from some problems such as low accuracy of referrals, centralized maintenance of referral information, and lack of privacy and control by the users.

This thesis seeks to develop a special kind of multiagent system, called multiagent referral system, in which every user is assigned a software agent. The software agent is able to automate the process of query-answer-referral and help its user to learn the experience and knowledge of others [Shardanand & Maes, 1995, Singh, Yu & Venkatraman, 2001]. The system is intended to be accurate and dynamic in providing personalized recommendations for users. With more and more customers shopping on the Internet, there is more and more of the need to connect these customers, and to share their experience of shopping [Duhan, *et al*., 1997].

Compared with traditional recommender systems, referral systems have following advantages:

1. Referral systems can be designed to provide trusted recommendations. Here the referral chain is important in providing a reason to an expert who agrees to respond to an inquirer by explicating their relationship (e.g., they have mutual collaborator), and in providing a criterion for the inquirer to use in evaluating the trustworthiness of the expert.
2. Referral systems complement the traditional recommender systems. There are economic, social, and political reasons that much valuable information will never be available to the public via the Internet or any other network. In this situation the referral systems will help to search the social network for an expert with a *chain* of personal referrals from the inquirer to the expert.
3. Referral systems can access a larger community without fear of offense. Software agents shield the users from seeing many irrelevant messages.

1.1 Emergence of virtual communities

Research in multiagent systems uses social conceptions to describe the interactions between software agents in a multiagent world. Agents were ascribed to form "societies", "teams", "groups", "organizations" [Malone, 1990]. We would like to use the term of "communities" for this purpose. Let us review the idea.

Software agents are persistent computations that include percepts, reasoning, action, and communication [Russell & Norvig, 1995]. In our multiagent referral system, each user is assigned a software agent. The user agent will help its user maintain his/her profile, neighbor models, handle the interactions between the user and his/her agent such as helping the user send out a query and to whom the query will be sent, etc. In case the agent decides that its user does not have the expertise to deal with an incoming query, the agent need make some referrals automatically if any. All the decisions made by a software agent are based on the representations of profile and neighbor models, which are built during bootstrap and modified through the agent learning.

The software agents organize themselves into "communities". An online community (or a virtual community) is defined as a set of interacting participants [Singh, Yu & Venkatraman, 2001]. The participants here are people. More generally, they can be businesses or other organizations as well. Therefore the activities happened in these communities are not restricted to queries or answers to the queries. They can also be business services provided for a fee or some referrals in case the agents (users) who are involved are not able to provide the services directly.

Virtual communities build on real communities but may go beyond the "hidden" barriers, which are caused by personal, political, religious reasons. On the other hand, it takes less resource (e.g. time and people) to let the agents take over much of the grunt work for maintaining such extended communities than to do it totally manually. This is the value of our referral system, which is able to explore the hidden resource (e.g. experts' knowledge in some domains) that will otherwise be wasted.

2. Background: Important Utilities

2.1 IBM Agent Building and Learning Environment (ABLE)

ABLE is a toolkit for building intelligent agents that include both reasoning and learning [ABLE, 2000, Bigus, 2001]. The ABLE framework consists of three major parts: AbleBeans, which are defined according to JavaBeans; AbleAgents, which are a set of function-specific JavaBeans and constructed using one or more AbleBeans; Able Agent Editor, which is a GUI-based interactive development environment that assists in the construction of AbleAgents using AbleBean components.

AbleBeans include beans for reading and writing data from text files, for data transformation and scaling using templates, for rule-based inferencing using Boolean and fuzzy logic, and for neural network learning. The AbleAgents include agents for classification, clustering, prediction, and genetic search (Figure 2.1).

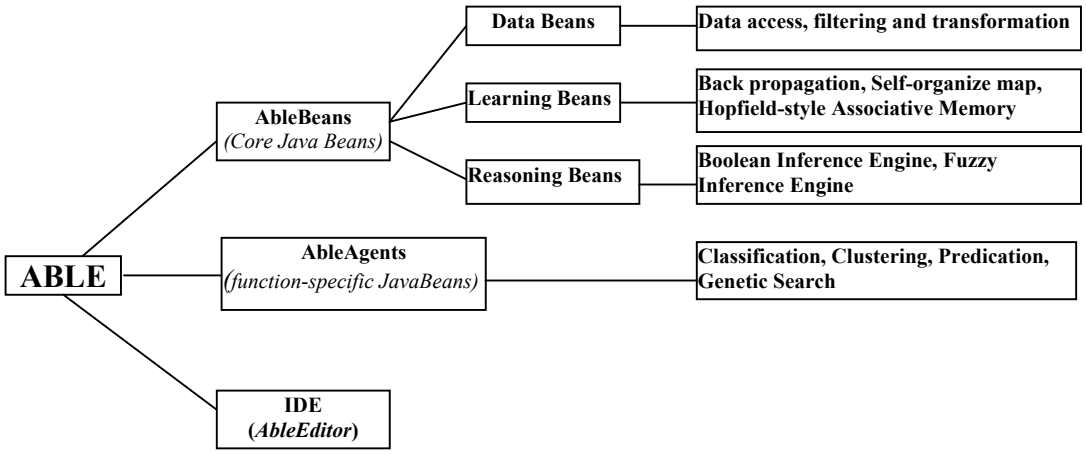

Figure 2.1 ABLE components

2.1.1 Agent reasoning

MARS deploys ABLE's forward chaining reasoning, in which resolution and Boolean rulesets have been used. Here let us review these related concepts.

2.1.2.1 Resolution

Resolution is a complete theorem-proving algorithm for first-order logic. It provides facts true or false by virtue of contradiction [Russell & Norvig, 1995]. A theorem X is true if and only if the negation of X ($\neg$ X) is not true. Let look at the following example:

1. $\neg$ cat(x) $\vee$ mammal(x)
2. cat(x)

Statement 1 means that if x is a cat then x is a mammal. Statement 2 means that x is a cat. To prove that x is a mammal, we have to show that statement "x is not a mammal" (statement 3) is not true:

3. $\neg$ mammal(x)

The resolvent of statement 1 and 2 is:

4. mammal(x)

Statements 3 and 4 cannot both be true. Therefore we have a contradiction and the assumption, $\neg$mammal(x), is false, and the alternative, mammal(x), must be true. If the clauses to be resolved are selected in systematic ways, resolution is able to find a contradiction if one exists, although it may take a long time to find it [Bigus *et al.*, 1998].

2.1.2.2 Boolean ruleset

The simplicity of If-then rules makes it become the most popular form of declarative knowledge representation used in artificial intelligence applications. Compared with knowledge represented in predicate logic, people are more comfortable reading the rules. Each rule can be viewed as a standalone piece of knowledge or unit of information in a knowledge base. New rules can be easily added, and simply creating or modifying individual rules can change existing knowledge.

ABLE Boolean inference engine includes forward-chaining algorithm and backward-chaining algorithm. Forward-chaining algorithm can be used to produce new facts (we have used this rule for MARS reasoning), and backward-chaining algorithm can deduce whether statements are true or not. Among the successful large-scale applications of artificial intelligence research, rule-based systems were one of the first ones. Typically, they consist of three major elements: a knowledge base, a working memory or database of derived facts and data, and an inference engine, which contains the reasoning logic used to process the rules and data.

To understand how it works, let us look at the following simple rule:

If animal = bird then feathers = true

By using forward-chaining algorithm, if we know that the animal is a bird (e.g. Tweety), then we can conclude that feathers is true and add that fact to our knowledge base.

A rule with all true antecedent clauses is said to be triggered or ready to fire. After a rule is fired, it is added as a fact to the working memory. A rule base may contain more than one rule that is ready to fire. The strategy of the inference engine will decide which one is fired.

A Boolean ruleset contains only Boolean logic. In other words, there are no fuzzy statements, or other types of logic statements allowed in a Boolean ruleset. Boolean rulesets can be used by any of the Boolean inference engines, for example the backward- and forward chaining inference engines.

2.1.2.3 MARS ruleset

ABLE now has seven different inference engines which all use the ABLE Rule Language (ARL). These vary from relatively lightweight engines to heavyweight artificial intelligence engines. As MARS uses forward chaining engines to do the reasoning, we provide the description on them only in the following:

Forward -- this lightweight engine processes if-then rules using forward chaining with specificity only for conflict resolution. Each rule is evaluated and if it is triggered, it is placed in the conflict set. A rule is selected from the conflict set based on the number of antecedent conditions. The one with the most clauses is chosen. If more than one rule is triggered and they have the same number of antecedent clauses, one is randomly chosen to fire. When the rule is fired, the consequent clause is evaluated, which usually results in an action being taken or a variable being assigned to a value. A variable assignment causes only those antecedent clauses and rules that reference the variable to be re-tested. The inference cycle is then repeated until no more rules can fire. Figure 2.2 shows a sample ruleset file of this forward chaining engine.

Forward2 - this medium weight inference engine uses a working memory and when/do pattern match rules. On each cycle, every rule and object is evaluated to determine the conflict set. A rule is selected from the conflict set based on the rule priority and the number of antecedent conditions. This engine produces results identical to the Forward3 engine, but has much less overhead when working with a small number of rules and/or small numbers of objects in working memory.

Forward3 - this heavy weight inference engine uses a working memory, a Rete-like pattern match network, and when/do pattern match rules. A pattern match network is used to test objects as they are added to the working memory. Partial matches are cached in the pattern match network. This inference engine produces identical results to the Forward2 engine, but is much more scalable

when there are large numbers of rules and/or large numbers of objects in the working memory.

```
Rulebase Query{
    InferenceMethod(Forward)

    Variables(
        user_expertise              Numeric(0.0)
        user_sociability            Numeric(0.0)
        expertise_threshold         Numeric(0.0)
        sociability_threshold       Numeric(0.0)
        action          Categorical("query" "decline")
    )

    InputVariables(user_expertise, user_sociability)
    OutputVariables(action)

    Rules(
      a1: expertise_threshold     = 0.5
      a2: sociability_threshold   = 0.8

      query1:
          if user_expertise > = expertise_threshold
          then
             action = "query"

      query2:
          if user_sociability > = sociability_threshold
          then
             action = "query"

     decline:
          if user_expertise < expertise_threshold and
             user_sociability < sociability_threshold
          then
             action = "decline"
    )
)
```

Figure 2.2 MARS ruleset file for query reasoning

2.2 JavaMail API

JavaMail is an API of Sun Microsystems released in February 2000 [Mani *et al*., 1998, JavaMail 1.1.3, 2000]. The JavaMail API, implemented as a Java platform standard extension, provides a set of abstract classes that model a mail system. The API provides a platform independent and protocol independent framework to build Java technology-based mail and messaging applications. The version we use is the JavaMail 1.1.3 implementation. It includes implementations of the core JavaMail API packages as well as implementations of the IMAP and SMTP service providers.

The JavaBeans Activation Framework extension (JAF) and POP3 provider are also needed along with the API. With the JAF, developers can take advantage of the standard services to determine the type of an arbitrary piece of data, encapsulate access to it, discover the operations available on it, and to instantiate the appropriate bean to perform said operation(s). The POP3 Provider implements a JavaMail interface to a POP3 service. Currently we are using JAF 1.0.1 and POP3 1.1.1 to implement MARS's communication parts.

The JavaMail layered architectural components are:

- The *Abstract Layer* declares classes, interfaces and abstract methods intended to support mail-handling functions that all mail systems support.
- The *Internet implementation layer* implements part of the abstract layer using Internet standards – RFC822 and MIME.
- JavaMail uses the JAF in order to encapsulate message data, and to handle commands intended to interact with that data.

JavaMail clients use the JavaMail API and Service Providers implement the JavaMail API. The layered design architecture allows clients use the same JavaMail API calls to send, receive and store a variety of messages using different data-types from different message stores and using different message transport protocols (Figure 2.3) [Mani *et al*., 1998].

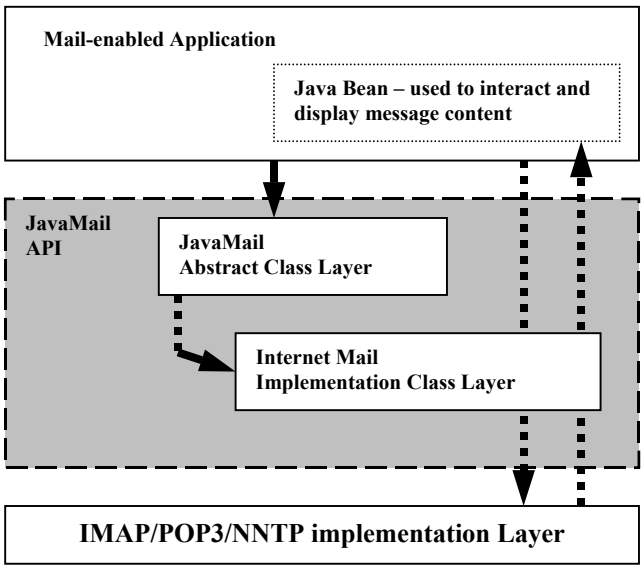

Figure 2.3 JavaMail layered architecture

The figure below shows major classes and interfaces comprising JavaMail API [Mani *et al*., 1998].

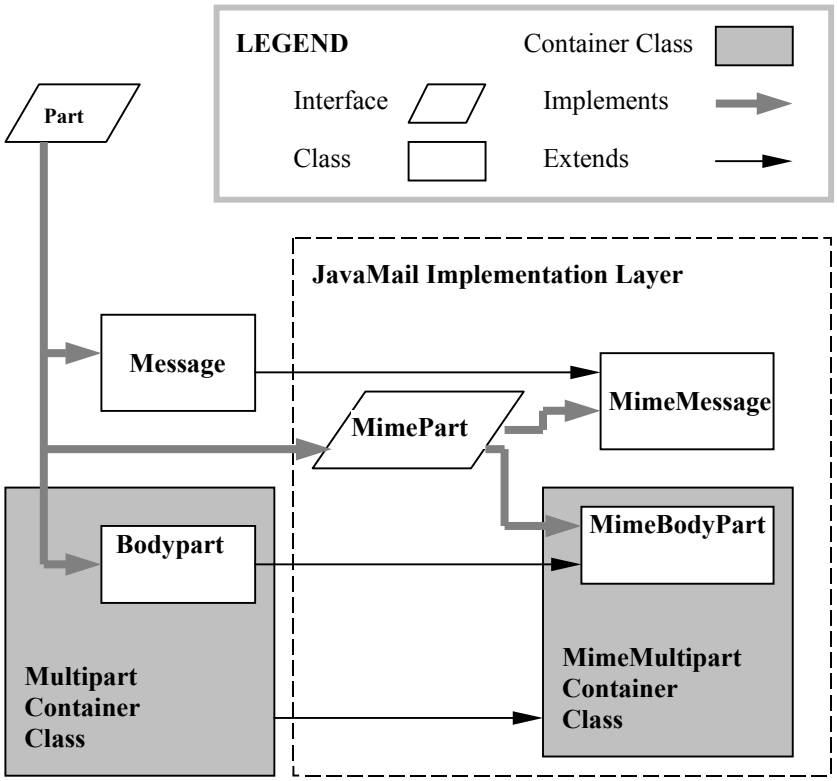

Figure 2.4 Major JavaMail API classes and interfaces

For a typical client application, the JavaMail API performs the following functions [Mani *et al*., 1998]:

- Create a mail message, which consists of a collection of header attributes and a block of data of some known data type as specified in the Content-Type header field. JavaMail uses the Part interface and the Message class to define a mail message. It uses the JAF-defined DataHandler object to contain data that is placed in the message.
- Create a Session object, which authenticates the user, and controls access to the message store and transport.
- Send the message to its recipient list.
- Retrieve a message from a message store.
- Execute a high-level command on a retrieved message.

For the current version of MARS, we have created 30 email accounts on salevista.com mail server for the alpha version testing. After a user downloads a MARS copy, an email account will automatically assigned to the agent. The agent will use that email account to communicate with other agents. The agent uses SMTP to send out a message and POP3 to retrieve message from the email server. The advantages of using email are: 1) to meet our need to build a totally distributed system; 2) it doesn't require other users be on-line when a user needs to send queries or make referrals to those users' agents or get answers from them.

3. Contributions of MARS

3.1 User modeling

One of the major concerns in MARS is how to model and maintain the knowledge/expertise learned from the user and other agents. To do this, we need to introduce the concept of ontology.

3.1.1 Ontology

Ontology is a specification of a conceptualization [Gruber, 1993]. We can regard ontology as a set of definitions of formal vocabulary. Practically, an ontological commitment is an agreement to use a vocabulary in a way that is consistent with respect to the theory specified by ontology. We build agents that commit to ontology. We design ontology so we can share knowledge with and among these agents.

For MARS, ontology is useful to represent a query domain and user's expertise. Currently in the MARS, we represent the ontology in the domain of Java programming, in which 18 fields in Java programming have been defined. It can be regarded as sub-tree of the root, which represents a more general domain (Figure 3.1). There is a program-independent file to store all the information about the ontology.

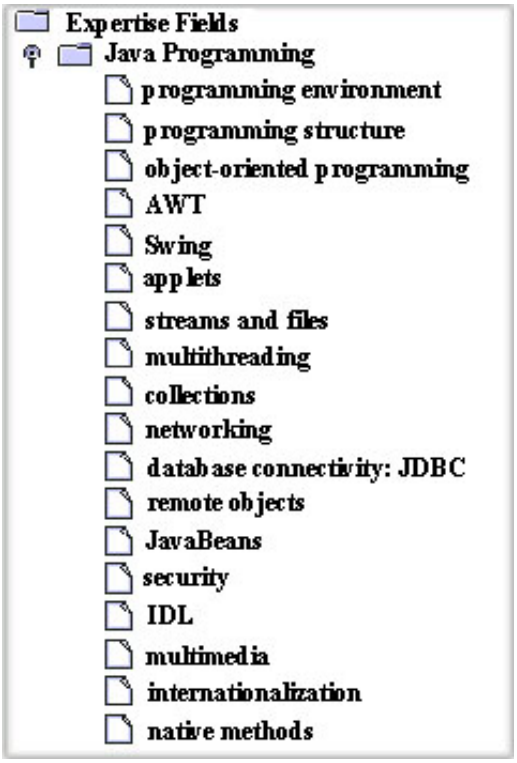

Figure 3.1 MARS ontology

3.1.2 Neighbor model

Here "neighbor" is defined as a list of the other agents whom the user's agent "knows". It's very important to keep a list of neighbor:

- When the user is sending a query, the user's agent must decide to whom from the neighbor list the query will be sent;
- When a query coming up and the user does not have the expertise to answer it, the agent also must decide to which agents in its neighbor list the query will be referred if any.

Each neighbor model contains the following information:

- DigitalID: A unique identified number assigned to each agent.
- Email address: The email address on salevista.com assigned to the agent.
- Name: User's full name.
- Expertise: Represented by a Hashtable the neighbor's expertise on different domain fields (each value is between 0 and 1).
- Sociability. A scalar represents the neighbor's sociability level (value between 0 and 1).

The neighbor models are invisible to the user. The user can maintain a close friend list instead. The close friend list is a sub-set of the neighbor models. When MARS runs at its first time, the agent's neighbor models are empty. MARS will automatically copy the close friend list to the neighbor models. After that, the agent dynamically maintains the neighbor models via agent learning. For example, let us consider the following scenario:

The user asks a question and then gets answer from someone who is neither in the close friend list nor in the neighbor model. If the user evaluates the answer as a good one, the agent who answers the query will be added into the neighbor models.

3.2 Multiagent learning in a social network

In the MARS system, each user has an agent representing him in the system. The agent for the user stores information about the user and his close friends (expertise, sociability, contact information etc.) during the bootstrap period of the system. After that, the agent will maintain and modify the information through the interactions between the user and his agent, the agent and other agents who represent other users (Figure 3.2).

In order to interact successfully with other agents, each agent has the following information:

- Its owner's expertise;
- The models of its neighbors, which include each neighbor's expertise and sociability.

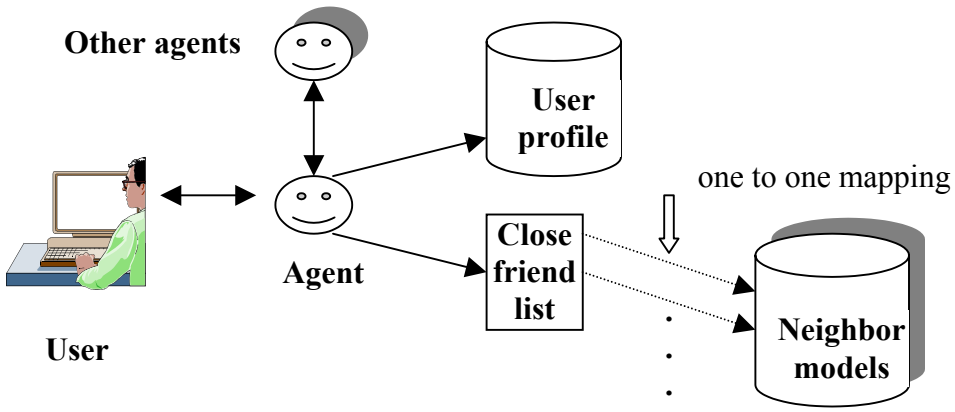

Figure 3.2 MARS data structure

After initialization, the models of its neighbors will be modified through the interactions between the agent and its user, and the interactions between the agent and other agents. When a good answer is obtained, the expertise of the agent who provides the answer will be increased, so will the sociability of the agents who make the referrals. When a bad answer is obtained, the revisions will be downward. Ultimately, what every agent is trying to learn about others is that their potential usefulness for finding the experts of the sort mostly often requested by its user.

3.2.1 Referral graph

One of most important features of MARS is that each agent is able to make referrals when the owner of the agent does not have the expertise to answer a query. The situation may continue quickly among MARS's agents. For a specific query, the referral chains may form a referral graph. Therefore the agent needs a referral graph to keep track of those referral chains corresponding to the query. A referral graph encodes how the computation spreads as a query originates from an agent and referrals are sent back to this agent.

Let's consider some facts on the referral graph:

Definition 1 A referral $R(a_i, a_j)$ returned from agent a_i is defined as $< a_i, a_j >$, where a_i is called the source, and is a_j called the destination of referral R.

When searching for a potential expert in a social network, usually the further distance an expert is from the inquirer, the less likely the expert will respond. Similarly, the more steps away from the inquirer, the less accurate of the referrals provided. In our work we set a bound 6 for the length of all referral chains.

Definition 2 Let Q be a query from agent a_i. We assume that, after a series of *l* referrals, agent a_j

returns an answer. The entire referral chain in this case is defined $< a_i, r_1, ..., r_{l-1}, a_j >$, as where $r_n (n = 1, ..., l-1)$ is the agent who makes a referral.

Definition 3 A referral graph ***G*** is defined as a directed graph (**A**, **R**), where **A** is a finite set of agents $\{a_1, ..., a_n\}$, and R is a set of referrals $\{r_1, ..., r_m\}$.

Obviously, not all agents are contacted at the same time. Some agents are contacted earlier, and others are contacted latter. Corresponding to a referral graph, some agents may be at lower level (closer to the source), and some agents may be at higher level (closer to the destination). We use the concept "depth" to represent the level of different agents in a referral graph, and the root (source) with a depth of zero.

Definition 4 Given a referral graph **G**, the depth of any agent a_i for **G** is defined as the length of the shortest path from the source a_i to the destination a_j, denoted as $depth(a_i)$, where a_i is the root of graph **G**.

Note that some of the referrals are not new to the referral graph, some referrals may be redundant, which cause a cycle or loop in a referral graph.

Definition 5 Closed path. A path $(X_l, ..., X_r)$ is said to be closed if it has the same starting and ending nodes, that is, $X_l = X_r$ [Castillo *et al.*, 1997].

Definition 6 Loop. A loop in a directed graph is a closed path in an undirected graph (Sample, Figure 3.4, A-B-D).

Definition 7 Cycle. A cycle is a closed directed path in a directed graph (Sample, Figure 3.5, B->C->D->B) [Castillo *et al.*, 1997].

Definition 8 Given a referral graph **G**, a referral $R =< a_i, a_j >$ is said to be redundant if and only if (1) agent a_i and a_j are some nodes in graph **G**, and (2) $depth(a_i) \geq depth(a_j)$.

Let's look at some examples for some of above definitions:

Normal Case (Figure 3.3)

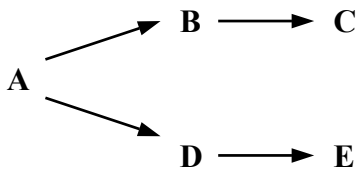

Figure 3.3 Referral Graph: Normal Case

In this case, **A** initializes the query process. The query has been sent out to its close friends **B** and **D**. Neither **B** nor **D** can answer the query. They refer **C** and **E** to **A** respectively. Finally **C** and **E** answer the query from **A**.

Later Referral (Figure 3.4)

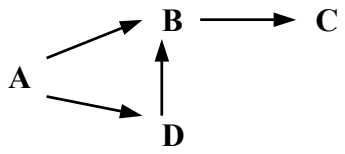

Figure 3.4 Referral Graph: Loop

In this case, **A** sends out the query to **B** and **D**. **B** refers **C** to **A** and **C** answers **A**'s question. Sometime later, **D** refers **B** to **A**. From this case, the can see the need to define a "close out" time. Anything that comes in after close out will be ignored. This is an example of loop. Loop is useful because it provides new information in the referral graph, i.e. the edge **D** to **B** in this case.

Cycle (Figure 3.5)

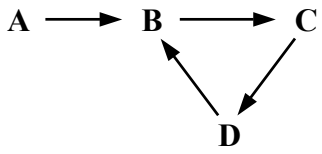

Figure 3.5 Referral Graph: Cycle

In this case, **A** asked **B**, **B** refers **C**, **C** refers **D**, and then **D** refers back to **B** (a circle). In our future work, we will keep loops in our referral graph but remove cycles.

3.2.2 Learning from feedback

When the user gets an answer to his query, he can evaluate to the answer. MARS therefore updates the corresponding agent's expertise and other agents' sociability. Figure 3.6 shows our algorithm for learning the expertise and sociability in a referral graph.

Assumption: An agent who gives the answer won't make a referral.

Suppose agent A_r is the requester agent, set Γ is the agents being visited. Then given a series of referral $\{r_1, r_2, \ldots, r_2\}$, for each referral $r_k = < A_i , A_j >$, agent A_r will update the expertise and sociability of other agents according to the following rules,

(1) if $A_j \notin \Gamma$ and A_j returns an answer, then
   1) append r_k to the referral graph, and A_j into Γ,
   2) evaluate the answer and update the expertise of agent A_j and the sociability of any agent to agent A_j;
   3) otherwise go to (2),

(2) if $A_j \notin \Gamma$ and A_j doesn't returns an answer, then just append r_k to the referral graph, and A_j into Γ, otherwise, go to (4)

(3) if $A_j \in \Gamma$ and *depth*(A_i) < *depth*(A_j), then just append r_k to the referral graph, and A_j into Γ, otherwise, go to (4)

(4) Otherwise, ignore the referral r_k.

Figure 3.6 Algorithm for learning the expertise and sociability in a referral graph

Definition 9 Given two learning rates α and β, where $0 < \alpha < 1$ and $0<\beta<1$, the update operation is defined as

(1) $S_{i+1} = S_i + \alpha - \alpha * S_i$ when a good answer is obtained;

(2) $S_{i+1} = S_i + \beta * S_i /(1 + \beta)$ when a bad answer is obtained;

(3) $S_{i+1} = S_i$ when a neutral answer is obtained (no change at all).

Where S_{i+1} stands for the new value of the variable (expertise or sociability),

S_i stands for the current value.

Figure 3.7 shows the learning curves for expertise and sociability. From the figure we can tell:
(1) If the answer is a good one, in the low value area, either expertise or sociability increases very fast while it increases very slowly in the high value area;
(2) If the answer is a bad one, in the high value area, either expertise or sociability decreases very fast while it decreases very slowly in the low value area.
Both above observations meet our intuition in the real world.

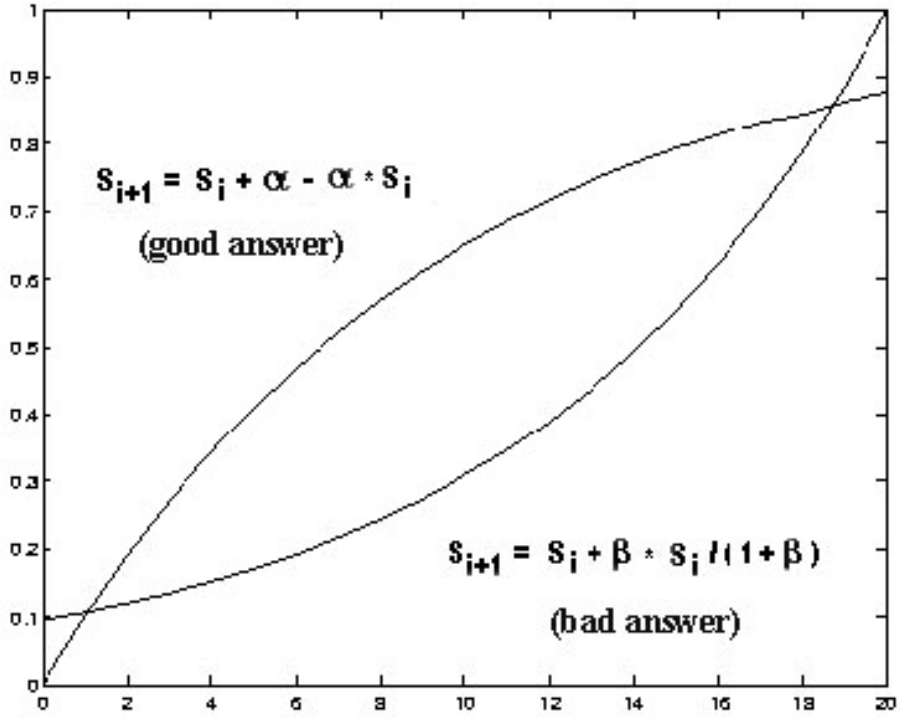

Figure 3.7 Learning curves
S: expertise or sociability value
α,β: experimental constants

Let Q be a query from agent a_i. Under **Definition 9**, we assume that, after a series of referrals, this leads to agent a_j who is able to answer the query. Let one of the referral chain in this case be $< a_i, r_1, ..., r_l, a_j >$ with the length *l*. Now a_i can refine the estimate of each intermediary agent toward certain queries, and use that to adapt the models that has of others. In this case, applying to $a_i's$ model, we use the operation in the definition to update $a_j's$ expertise and the sociability of the referral who is closest to a_j (in this case, it is r_l). We use the following formula to update other referrals' sociability:

$$S_i = \frac{S}{2^{l-i}}$$

where S is the updated sociability of the closest referral and i = 1,…, *l*-1.

Table 3.1 summarizes the strategy we use in updating the models that an agent has of other agents based on its success in obtaining a good answer or a referral to a good answer. A ↑ indicates that the corresponding estimate is revised upwards, whereas a ↓ indicates that it is revised downwards.

Situation		**Y's exp**	**Y's soc**	**Z's exp**	**Z's soc**
Y gives a good answer		↑	–	–	–
Y gives a bad answer		↓	–	–	–
Y refers to Z					
	Z gives a good answer	–	↑	↑	–
	Z gives a bad answer	–	↓	↓	–
	Z gives a good referral	–	↑	–	↑
	Z gives a bad referral	–	↓	–	↓

Table 3.1 Learning by X about Y and Z: when X asks Y; if Y refers to Z, then X asks Z

Our work is related to reinforcement learning [Kaelbling *et al.*, 1996], in which programming agents are guided by reward and punishment without needing to specify *how* the task is to be achieved. Q-learning [Chris, 1989] is the most popular and seems to be the most effective model-free algorithm in reinforcement learning for learning from delayed rewards. It does not, however, address any of the issues involved in an environment without explicit reward or punishment.

There is one important thing we need to point out here is that the update to the referral graph is different from Bayesian Network (BN). Let us consider this scenario: **A** asks **B**, **B** refers **C** and **C** refers **D** and **D** answers **A**, that is **A->B->C->D**. There are probably two ways to update neighbor models: (1) keep the computation in **A**, and propagate the results to **B** and **C**. In this way **B** and **C** will get the referral graph. This will raise two problems: privacy and computation complexity. (2) move the computation to all agents. In the above example, **D** propagates the answer to **A** through **B** and **C**. This is more like a Bayesian network. There are two problems for this method too: first, learning from inquirer agent will be slow. BN is good for a static structure, but social network is not. Second, it is hard to control when to stop the computation.

3.3 Message format

A typical MARS message format is:

originID	originName	qid	msgType	domain	toID	toName	content	date

originID: ID for the agent who initializes the query.
originName: The name for the agent who initializes the query.
qid: Unique query ID created for each query.
msgType: Message type – such as query, answer, referral.
domain: The domain field related to a query.
toID: ID for the destination agent.
toName: Name for the destination agent.
content: Message content.
date: The date when the query is sent.

In the following examples, we will show how the above format fits to different cases (we will not show content and date in the following examples because they all are the same).

Case 1: Send out a query

myDigitalID	myName	qid	query	query domain field	receiver's digitalID	receiver's name

Case 2: Receive a query, then answer it

Query received:

Sender's DigitalID	Sender's name	qid	query	query domain field	my digitalID	my name

The user has the expertise to answer the query so the agent displays the query to the user. If the user replies the query, an answer will be sent:

myDigitalID	myName	qid	answer	domain	sender's digitalID	sender's name

Case 3: Receive a query, refer it to close friends

If the user doesn't have the expertise to answer the query but some of his/her close friends have, a referral message will be sent back to the originator of the query automatically:

referral's ID	referral's name	qid	referral	domain	close friends' ID list	name list

Case 4: Receive a query, ignore it

Neither the user doesn't have the expertise nor his/her close friends have.

Case 5: Receive a referral, then send the queries to those have been referred

Referral message received:

referral's ID	referral's name	qid	referral	domain	close friends' ID list	name list

The agent automatically sends out the query to those whom are referred. For each query message:

my digitalID, referral's digitalID	my name, referral's name	qid	requery	domain	new receiver's digitalID	new receiver's name

Case 6: Receive an answer

my digitalID	my name	qid	answer	domain	answerer's digitalID	answerer's name

3.4 Internal architecture of each agent

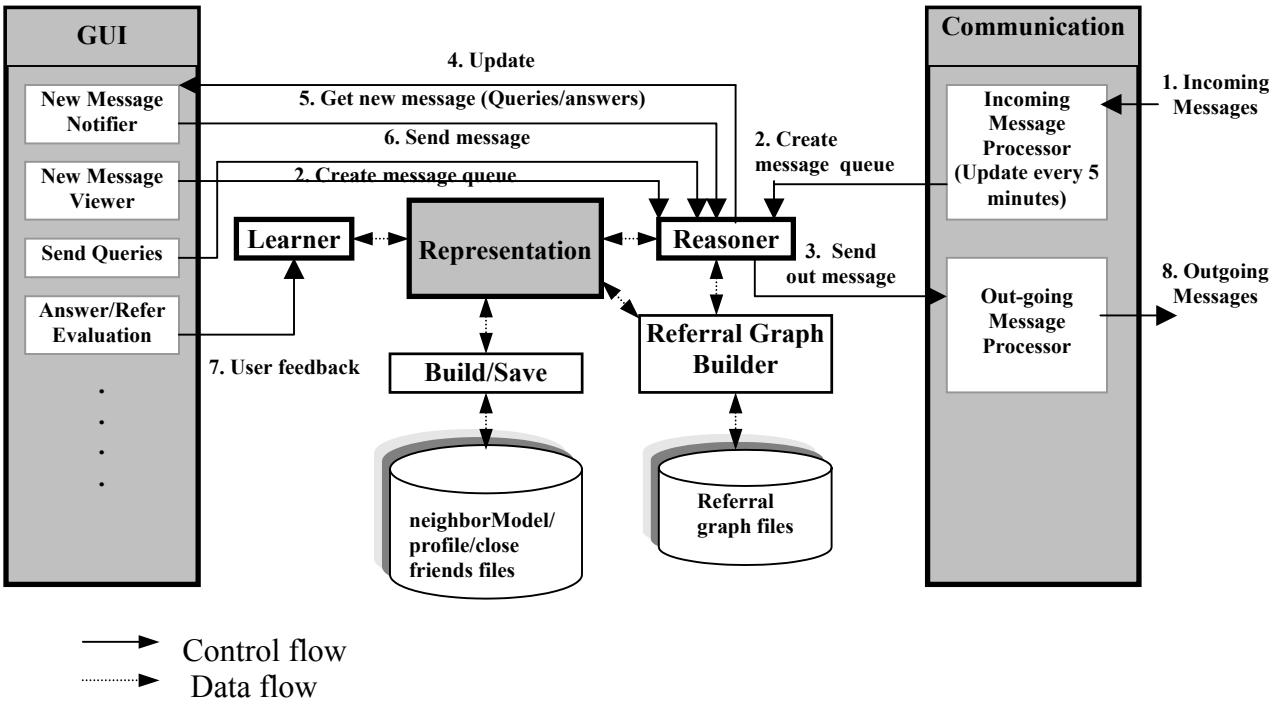

Figure 3.8 Internal architecture of MARS agent

MARS is a prototype system for information gathering in a person-to-person social network. Our purpose is to develop an accurate, dynamic, and evolving multiagent system that can achieve the effect of the informal social networks that exist in an organization or community. It is composed of the following modules (illustrated in Figure 3.8):

GUI: MARS interfaces that enable the interactions between the user and MARS, i.e., transforming the queries and responses between the Reasoner, Learner and the user.

Reasoner: MARS reasoning components deploying ABLE forward chaining reasoning. It has a major auxiliary component – "Referral graph builder" which is responsible for creating the corresponding referral graph files and saving change to those files. Figure 3.9 shows the relation between MARS's Reasoner and ABLE. When MARS main GUI starts, an instance of Reasoner is initialized. Every time MARS needs to do the reasoning job, it calls the Reasoner instance with corresponding value. The Reasoner then calls ABLE forward chaining reasoning components and gets the reasoning result from ABLE. Finally MARS will use the result to determine what next action it will take.

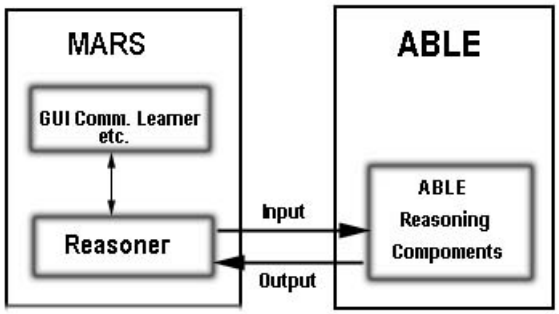

Figure 3.9 MARS and ABLE

Learner: MARS learning components to update user's neighbor model, including the update of expertise and sociability.
Representation: Data structure representing user profile, neighbor models, message queue, and referral graph etc. It has a major auxiliary component – "Build/Save" that maintains the corresponding files.
Communication: The components dealing with sending and receiving message between agents.

Now let us have a close look at the control flow of the architecture (Figure 3.8):

1. "Incoming message processor" of the communication components is implemented by using a Timer, which is activated every 5 minutes. When the Timer is activated, it will check if there is any new message coming up. Go to step 2 if there is any new message coming up.
   Note: the user also can control this step manually by clicking the "New message viewer" button on the GUI.
2. Create a new message queue. At this stage, the Reasoner will do the following things:
   (1) Check each message in the message queue. If the message is a query that the user can answer, or an answer to the user's query, the message will be kept in the message queue;
   (2) If the message is a referral for one of the user's queries, send out the query again to the agent whom is referred (execute step 3).
   (3) If the message is a query that the user does not have the expertise to answer, the Reasoner will check the agent's neighbor list to see if there is any neighbor who has the expertise to answer the query. If yes, send out a referral message to the agent who initializes the query (execute step 3); if not, skip it.
   In case of (2) and (3), after the agent send out the corresponding messages, the Reasoner will remove the message from the message queue. This will results in keeping in the message queue only answers to user's queries and queries to the user that the user has the expertise to answer.
3. Forward a message to the "Out-going message processor" of the communication.
4. This happens when step 1. and step 2. happen sequentially. At this stage, the GUI part will take care of popping up a message window notifies the user of new message.
5. After the user responses to "New message notifier", a "Message displayer" GUI will be shown up which is able to display the message queue, and answer, refer, or reject a message.
6. The user clicks a "Send message" button. The Reasoner will determine to which neighbor the query goes. At this stage, a specific referral graph file will be created for the query.

7. When the user gets an answer to his/her query, he can use the corresponding GUI to make an evaluation to the answer. The Learner will be activated at the time. It will update the expertise and sociability of corresponding neighbors' with the assistance of user's feedback and the referral graph.
8. The "Out-going message processor" sends out a message.

3.5 Implementation

MARS's implementation involves several programming toolkits and utilities. These are IBM Research Center's ABLE 1.2a, Sun's JavaMail API, POP3, and JavaBean Framework. The MARS system is implemented on Windows NT/98 by using Java 1.2. From the internal architecture, we know MARS is composed of five major modules: GUI, Reasoner, Representation, Learner, and Communication. We have discussed Communication in 2.2 and Representation in last section (3.4). We'll discuss GUI in Part 4. In this section, we focus on Reasoner and Learner, and their relations with other components.

Java is a purely object-oriented (OO) language. From software engineering's point of view, based on our architectural design, we need to do the object-oriented analysis (OOA) and object-oriented design (OOD) before actual coding [Pressman 2001]. The design method, Unified Modeling Language (UML), is a good fit into the object-oriented world [Fowler & Scott, 2000]. In the following description, we will use collaboration diagrams and class diagrams to describe the Reasoner and Learner in details (All the diagrams are drawn using Rational Rose).

3.5.1 Collaboration Diagrams

Collaboration deals with the interaction among two or more classes. Collaborations are useful when we want to refer to a particular interaction. In this section, we want to use collaboration diagrams to describe the major situations Reasoner and Learner to deal with.

Case 1: Sent out a query.

On the MARS main GUI, user can type in his/her query and choose a domain field related to the query. Then the user clicks the "Send" button to send out the query. There are five classes involved in this case:

- MainGUI: a class extends from Jframe of JFC Swing to display the major MARS GUI window.
- Reasoner: MARS reasoning class.
- Communication: msgSender class. Communication contains msgReceiver and msgSender classes. For the purpose of clarity (especially in the sequence diagram), we use Communication to represent them.
- ReferralGraph: referral graph class.
- Actioner: a class to connect all the other classes.

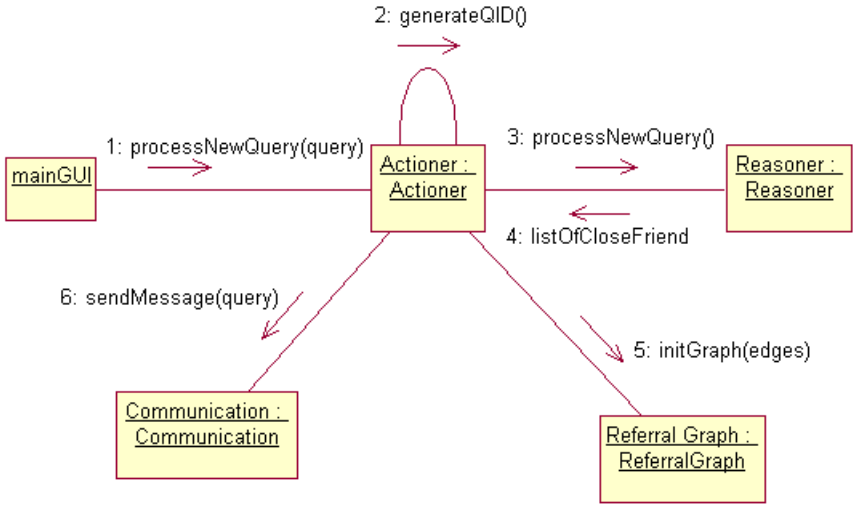

Figure 3.10 Collaboration Diagram: Send out a query

Let us look at the control flow (Figure 3.10):

1. After the user clicks the "Send" button, in the event handler of mainGUI, an instance of Actioner class is created and its method, processNewQuery(query) , is called;
2. Actioner identifies that it is a new query and therefore generates a new query ID (QID) by using the method "generateQID()";
3. Actioner passes the query domain field value to Reasoner;
4. Reasoner returns a list of close friends who have the expertise that meets the domain field;
5. A referral graph corresponding to the QID is created and the edges from the user to his/her close friends are added to the graph;
6. MsgSender send out the query messages.

Case 2: Received a query, then answered it.

A query from another agent is coming up. User's agent identifies that the user has the expertise to answer the query and displays the query to the user.

There are five major classes involved in this case (network is used to describe the general concept, not represents a class in the architecture):

1. Communication: both msgReceiver and msgSender involved.
2. Actioner.
3. Reasoner.
4. DisplayQuery: the GUI class displaying incoming queries and answers.
5. AnswerGUI: the GUI class dealing with answering a query.

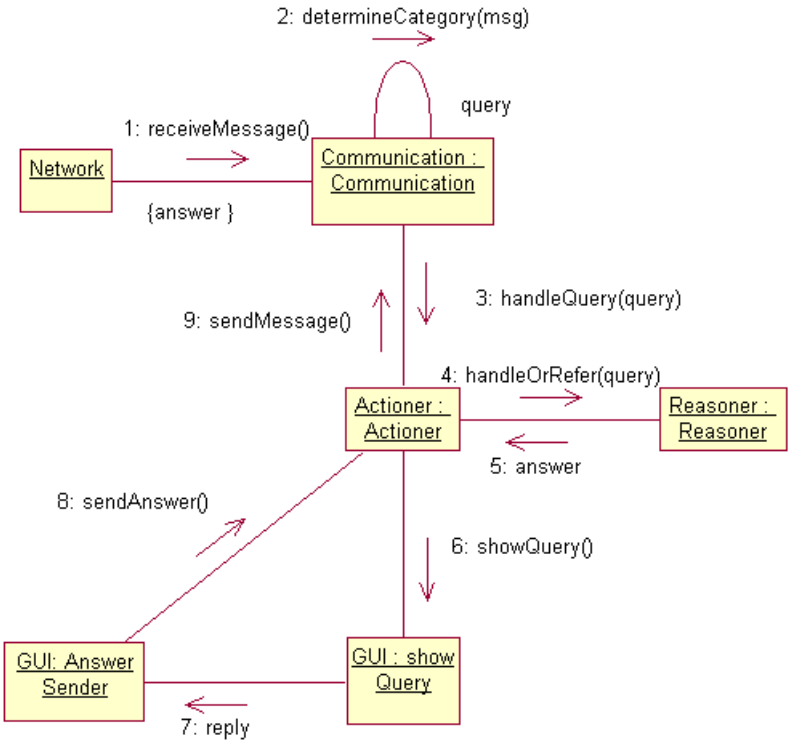

Figure 3.11 Collaboration Diagram: Answer a query

The control flow (Figure 3.11):

1. Either it is initialized by the message Listener timer or user mouse click event, the msgReceiver class is called;
2. msgReceiver identifies that it is a query according to the message format;
3. msgReceiver passes the value to Actioner;
4. Then Actioner passes it to Reasoner;
5. Reasoner's Reasoning result is that an "answer" should be provided;
6. Class DisplayQuery is created and shown;
7. If user click the "Reply" button, the class AnswerQuery is created and shown;
8. Replying query action is passes to Actioner;
9. Actioner passes it to msgSender and msgSender sends out the answer message.

Case 3: Received a query, referred it to close friends.

A query is coming up. The user's agent checks to see that the user doesn't have the expertise to answer the query. Then it sends a message to the agent who initializes the query with a list of its close friends.

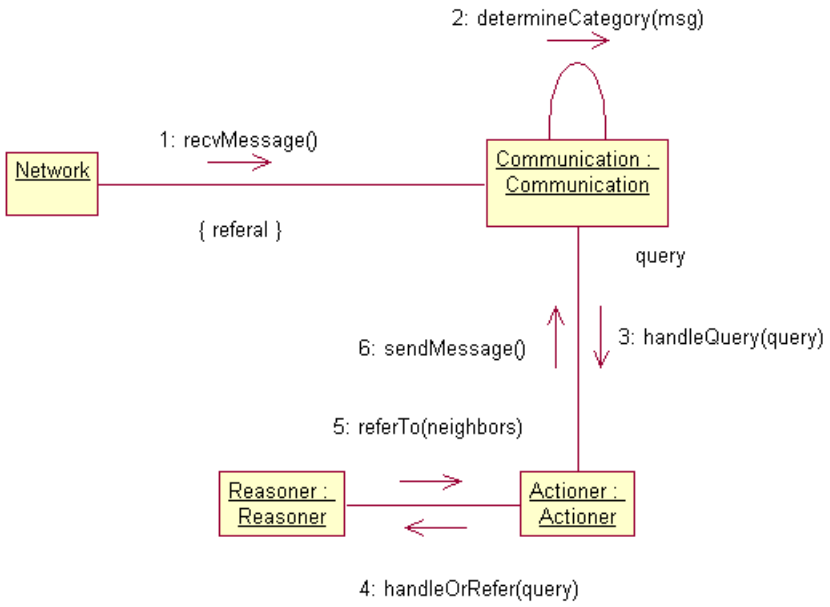

Figure 3.12 Collaboration Diagram: Make referrals

There are three major classes involved in this case:

1. Communication;
2. Actioner;
3. Reasoner.

The control flow (Figure 3.12):

1. Either it is initialized by the timer or user mouse click event, the msgReceiver class is called;
2. msgReceiver identifies that it is a query from the message format;
3. msgReceiver passes the value to Actioner;
4. Actioner directs it to the Reasoner;
5. Reasoner's reasoning result is "referral" and a list of close friends;
6. Actioner passes the referral message to msgSender and MsgSender sends out the message.

Case 4: Received a query, ignored it (neither answer nor refer).

A query is coming up. The user's agent identifies that neither the user has the expertise to answer the query nor does any close friends. The agent will take no action.

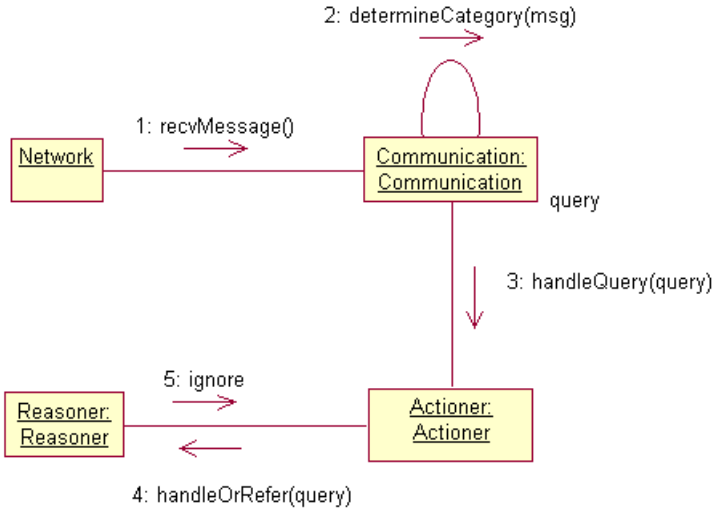

Figure 3.13 Collaboration Diagram: Ignore a query

There are three major classes involved in this case:

1. Communication;
2. Actioner;
3. Reasoner.

The control flow is (Figure 3.13):

1. Either it is initialized by the timer or user mouse click event, the msgReceiver class is called;
2. msgReceiver identifies that it is a query from the message format;
3. msgReceiver passes the value to Actioner;
4. Actioner directs it to the Reasoner;
5. Reasoner's reasoning result is "ignore" and stop.

Case 5: Received a referral, then sent the query to those whom were referred.

The agent who sends out a query receives a referral regarding the query. It retrieves information from the referral message and then sends out the query to those agents who are referred.

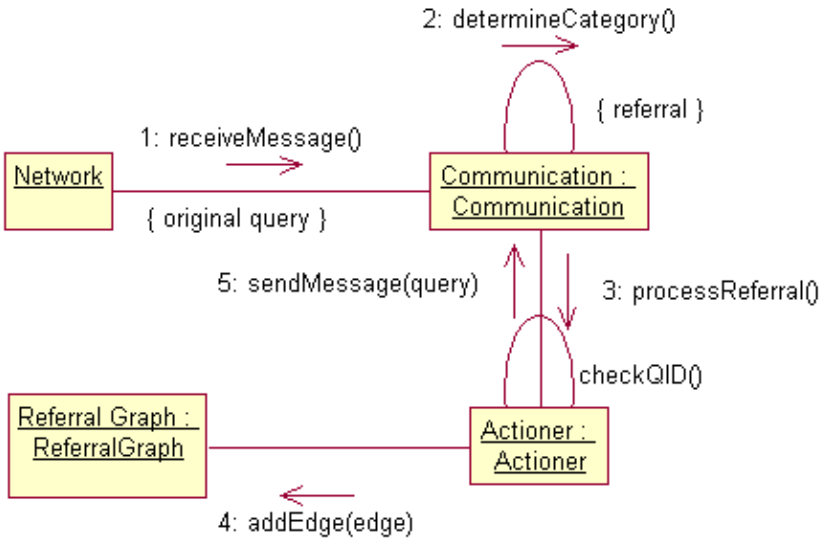

Figure 3.14 Collaboration Diagram: Send queries to whom are referred

There are three major classes involved in this case:

1. Communication;
2. Actioner;
3. ReferralGraph.

The control flow is (Figure 3.14):

1. Either it is initialized by the timer or user mouse click event, the msgReceiver class is called;
2. msgReceiver identifies that it is a referral message from message format;
3. msgReceiver passes the value to Actioner, who retrieves QID from the message;
4. The class ReferralGraph is loaded. The edges from the agent who makes the referrals to the agents who are referred are added to the referral graph;
5. Actioner passes the referral information to the msgSender and msgSender sends out the message.

Case 6: Received an answer.

The agent receives an answer to its query and displays the answer to the user. The user can then evaluate the answer.

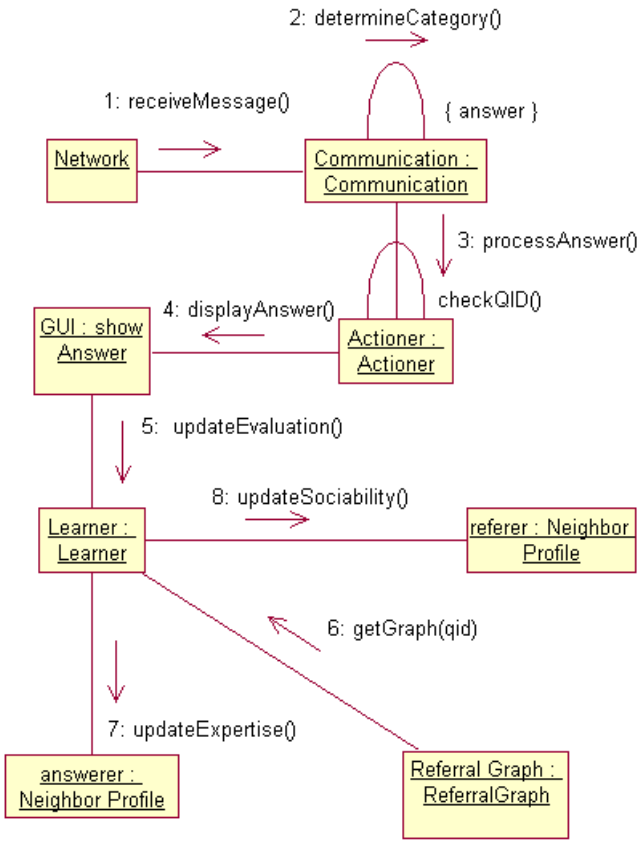

Figure 3. 15 Collaboration Diagram: Get an answer

There are seven major classes involved in this case:

1. Communication;
2. Actioner;
3. DisplayAnswer and EvaluationGUI;
4. Leaner: the class deals with MARS's learning tasks;
5. Answerer: An instance of class neighborProfile;
6. Referer: Another instance of class neighborProfile;
7. ReferralGraph.

The control flow is (Figure 3.15):

1. Either it is initialized by the timer or user mouse click event, the msgReceiver class is called;
2. msgReceiver identifies that it is an answer from the message format;
3. msgReceiver passes the information to Actioner, who retrieves QID from it;
4. The window of displayAnswer is shown and EvaluationGUI is shown after the user responses;
5. Learner is created;
6. Learner retrieves the corresponding referral graph;
7. According to user's feedback, Learner updates the answer's expertise;
8. According to user's feedback, Learner updates referrals' sociability.

3.5.2 Reasoning and Learning Class Diagram

A class diagram describes the types of objects in the system and the various kinds of static relationships that exist among them. There are two principal kinds of static relationships:

- Associations (for example, a student may borrow a number of books)
- Subtypes (a student is a kind of person)

Class diagrams also show the attributes and operations of a class and the constraints that apply to the way objects are connected.

Figure 3.16 shows the class diagram, which describes the Reasoner, Learner, other classes related to them, and the relationships between these classes. It provides another view of understanding the relationships between the classes we have discussed. We can notice again that Actioner is a class to connect most of the other classes related to the reasoning and learning components.

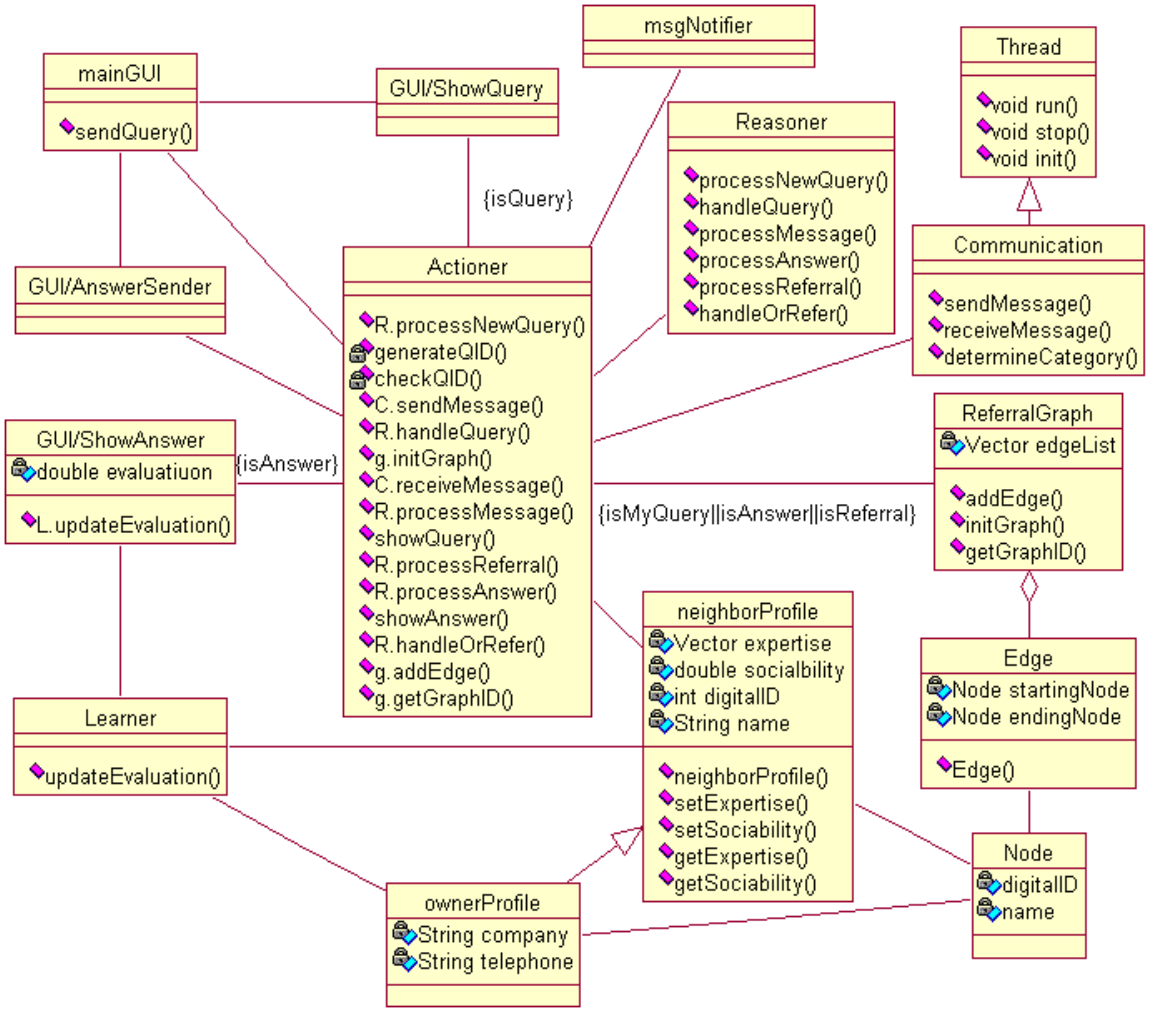

Figure 3.16 Class diagram

4. MARS alpha version

Currently MARS is at its alpha version. It includes the basic functions such as query, referral, answer, and some functions related to them. The implementation involves the Graphic User Interface (GUI) designs, which raise some Human-Computer Interaction (HCI) concerns.

4.1 Design decisions

When we began to develop the prototype system of a MARS's user agent, we followed the following fundamental HCI design guidelines [Preece *et al.*, 1994]:

- **Visibility**: We should make things visible in our designs. Let's take a look at the MARS main GUI (Figure 4.1). You can see everything we need to do can be accessed from this window. You may send out a query, get new messages, maintain user's own profile and modify user's close friend list on this window.
- **Affordance**: The perceived and actual properties of thing, primarily those fundamental properties that determine just how the thing could possibly be used. Affordances provide strong clues to the operations of things. For examples, you can click a button to send out a query or you can look up a query field from the domain tree.
- **Mapping**: The relationship between two things. Natural mapping, by which taking advantage of physical analogies and cultural standards, leads to immediate understanding. We tried to achieve this by using suitable caption for each button, label that explains the function of the text field.
- **Feedback**: Sending back to the user information about what action has actually been done and what result has been accomplished. After the user clicks the "Send" button, text field will display to whom the query has been sent.

As I chose small-sized window and therefore we can't put everything in one window, the issues of visibility and mapping are becoming critically important. I tried to solve the problems to add some comment labels besides the functioning buttons. Also I have to carefully choose the wording on the captions of the buttons. On some cases I have to increase the widths of the buttons to provide enough caption explaining words. All these seem to be very helpful.

To achieve the above guidelines, our iterative design process is:

(1) According to the motivation of MARS, in the future MARS will be used in a cell phone or a PDA. So we decide to choose a small screen design. At this stage, we have studied some PDA interfaces (such as Motorola PageWriter 2000) and some instant messengers (e.g. AOL Messenger, Microsoft Messenger, ICQ etc.).
(2) Initial scenario design. We began to draw our designs on paper. Because we choose the small screen and want to make everything visible on the small screen, it's very helpful to draw all the scenarios and how to make them accessible from a main window.
(3) Early usability evaluation. We put the paper designs on computer using PhotoShop in order to get the closer look and feel to the actual application. We invited some users to test the usability of these designs. Some changes have been made during the interaction with the users.
(4) JFC Swing Prototype. We began to write the Java code to implement our refined designs.

(5) Pilot testing. When we complete the coding, we asked some users to download MARS. We have got valuable feedback after they actually ran the MARS.
(6) Repeat step (4) to step (5).
(7) Final Design. We get our current "final design".

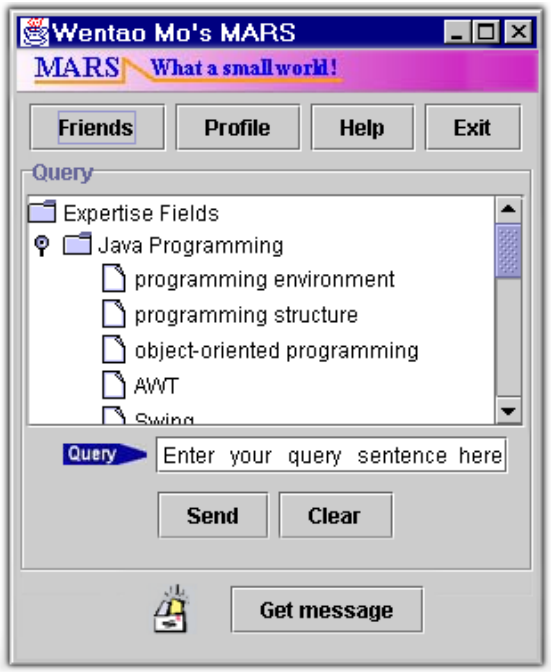

Figure 4.1 MARS main GUI

4.2 MARS in action

Currently we have setup a web site to download MARS alpha version (Figure 4.2). The version runs on MS Windows environment using Java 1.2 or above.

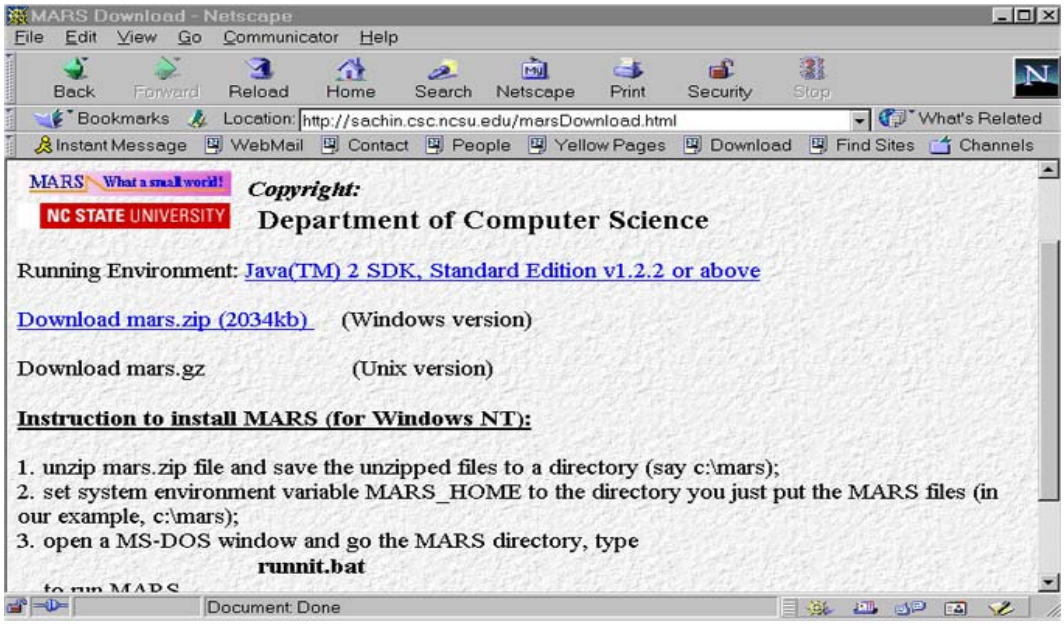

Figure 4.2 MARS download web site

For the first time to use MARS, the user will be asked to input some information related to the user profile along with a password the user chooses. After the sign-up, the user is able to log into MARS (Figure 4.1). Now let's take a look at MARS's major functions.

- User profile (Figure 4.3): Display user's contact information and expertise. The profile is corresponding to ownerProfile in the class diagram. The user during the sign-up provides the data. He/she can modify his/her own profile via this function.

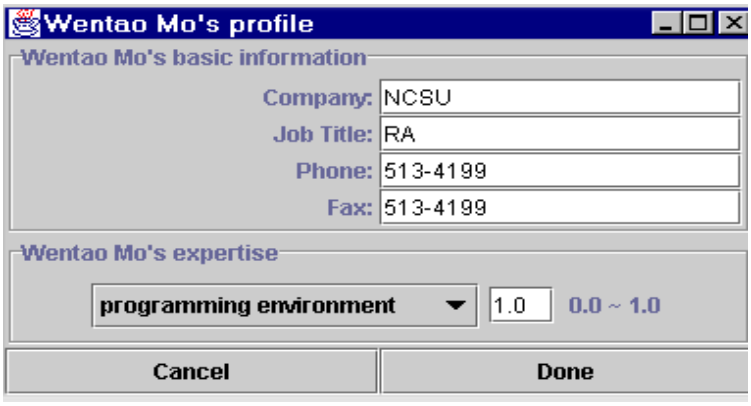

Figure 4.3 User profile

- Add close friends (Figure 4.4): Display user's close friend list, the expertise and sociability for each friend. User can add new friends to the list as well as remove friends from the list. The friend list is corresponding to the neighborProfile in the class diagram.

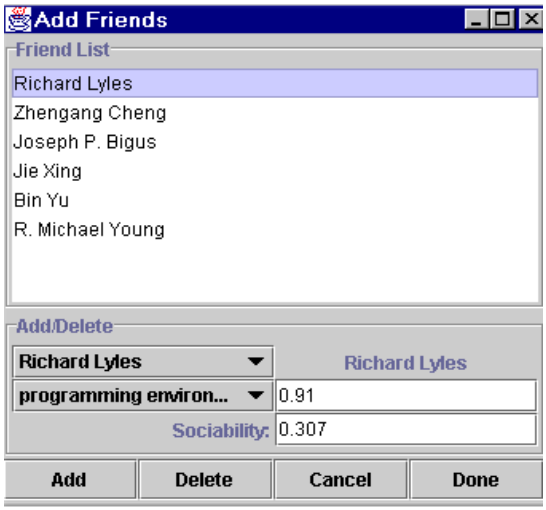

Figure 4.4 Add close friends

- Send out a query: After setting up close friend list, the user is able to ask any question he/she may have. First, the user needs to select a domain field to which the query belongs. Then he/she can type in the query and clicks the "Send" button. The query will send to those who have the expertise to answer (from the user agent's point of view).

- Get message (Figure 4.5): The window will be shown up when there are new queries and/or answers coming up. The user can use the button "Previous" and "Next" to browse the message in the message queue. For a query, the user can choose reply, refer or reject; for an answer, the user can evaluates the answer.

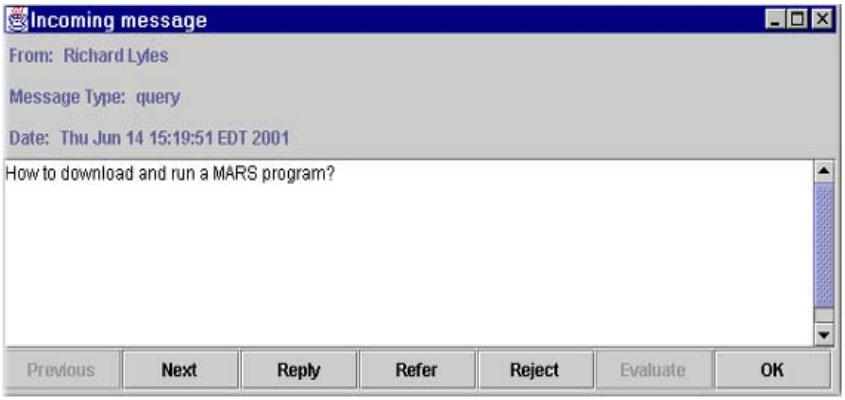

Figure 4.5 Get message

- Evaluate an answer (Figure 4.6): for each answer, the user has three choices: good, neutral, and bad. This is corresponding to the Learner in the class diagram. MARS agent uses the user's feedback here to update its neighbor models.

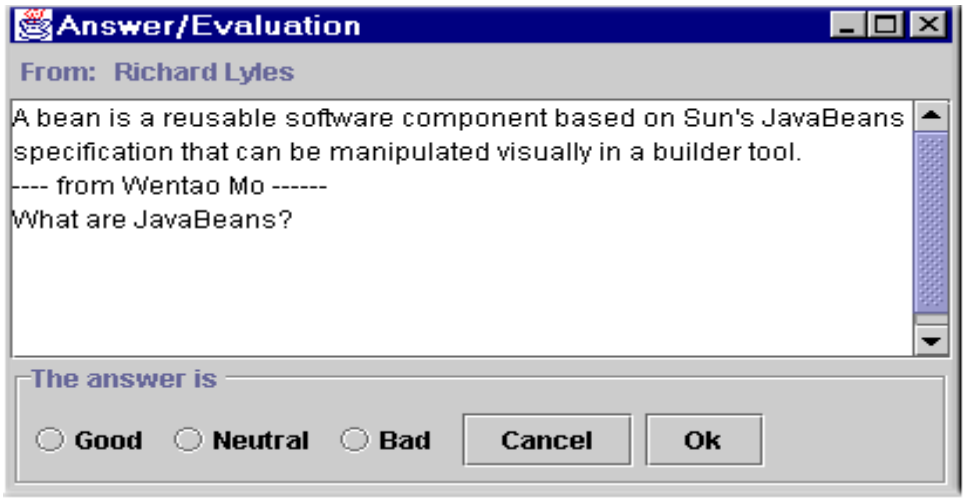

Figure 4.6 Evaluate an answer

- Help (Figure 4.7): The MARS application can activate a browser and direct the internet address to MARS web site's MARS manual, which helps users how to use MARS fully.

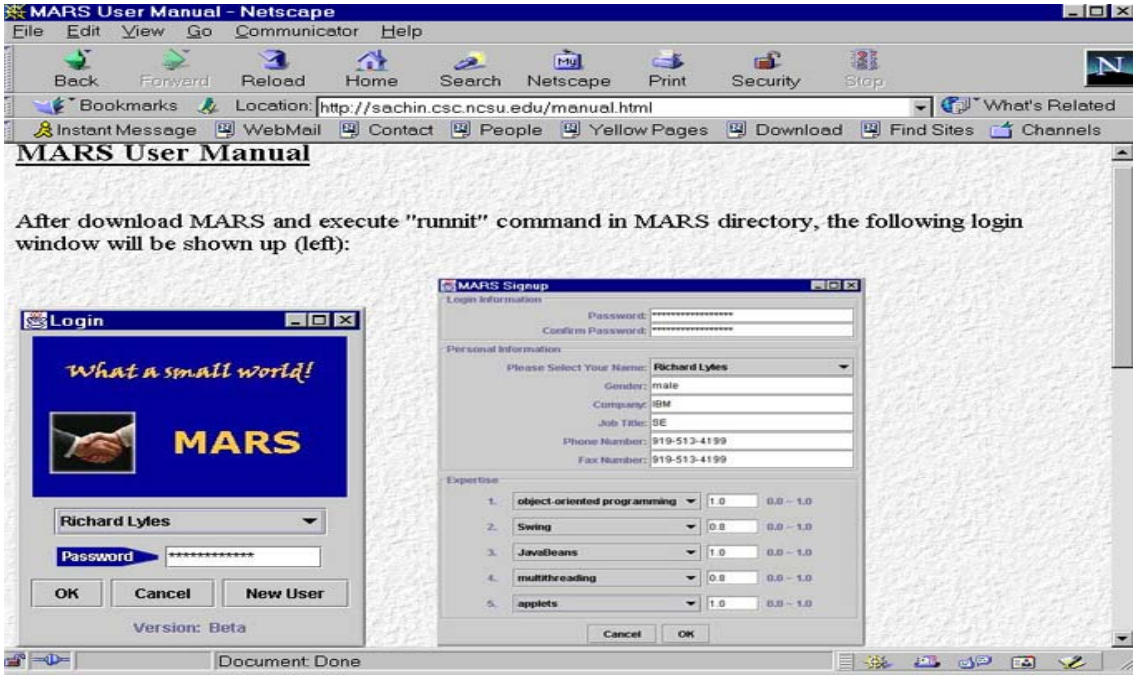

Figure 4.7 User Manual

5. Related work

MARS is a special kind of recommender system. Many different kinds of recommender systems were developed during the 1990s. They can be classified into four categories: content-based filtering (find items liked by the customer who bought similar items before) [Krulwich & Burkey, 1996, Lang, 1995, Lieberman, 1997], collaborative-filtering (find items liked by people who are similar to the customer) [Hill *et al.*, 1995, Konstan, 1997, Shardanand & Maes, 1995], matchmaker systems (cluster the agents together with the similar interests) [Foner, 1997, Kuokka & Harada, 1995, Lashkari, 1994, Maes, 1994] and referral systems (find someone with the needed information by referrals) [Kanfer *et al.*, 1997, Kautz *et al.*, 1997, Krulwich & Burkey, 1996]. Table 5.1 summarizes the characteristics of different approaches to recommender system.

	Recommendations	**Anonymous?**	**Comparison**
Content-based approach	netnews, articles	Yes	items with user profile
Collaborative filtering	netnews, articles	Yes	people with people
Matchmaker system	people	No	people with people
Referral system	people	No	People with people

Table 5.1 The characteristics of different approaches to recommender system

5.1 Content-based and Collaborative Filtering

The content-based approach to recommendation deploys the technology of information retrieval (IR). It recommends items because they are similar to those that a user has liked in the past. Text documents are recommended based on a comparison between their content and a user profile. Examples of such systems are InfoFinder [Krulwich & Burkey, 1996], NewsWeeder [Lang, 1995], and Letizia [Lieberman, 1997].

InfoFinder asks the users to supply some sample documents, and then searches for related documents. The search will return the URLs of the identified papers by electronic mail. The sample documents provide the agent with information about the frequency of words inside the desired papers, and also patterns or common phrases that can be important to do a better ranking of the final documents.

NewsWeeder is a netnews-filtering system that allows the user to rate each article read from one to five, through a Mosaic-based interface. The system then learns a user profile based on these ratings and can use this profile to find unread news that may interest the user.

Letizia is a user interface agent that assists a user browsing the World Wide Web. As the user operates a conventional Web browser such as Netscape, the agent tracks user behavior and attempts to anticipate items of interest by doing concurrent, autonomous exploration of links from the user's

current position. The agent automates a browsing strategy consisting of a best-first search augmented by heuristics inferring user interest from browsing behavior.

But pure content-based systems have several shortcomings. Generally, only a very shallow analysis of certain kinds of content can be supplied. In some domains the items are not amenable to any useful feature extraction methods with current technology.

The collaborative filtering (CF) approach recommends items other similar users have liked. Collaborative recommendation addresses the shortcomings of content-based systems. Automatic CF systems such as GroupLens [Konstan *et al.*, 1997] provide predictions with little or no user effort. Later systems such as Bellcore video recommender [Hill *et al.*, 1995] and Ringo [Shardanand & Maes, 1995] became widely used sources of advice on movies and music respectively.

However, CF approach introduces other problems. If the number of users is small relative to the volume of information in the system, then there is a danger of the coverage of ratings becoming very sparse, thinning the collection of recommendable items. A second problem arises when a user's tastes are unusual compared to the rest of the population. There will not be sufficiently many similar users, leading to poor recommendations.

Hybrid recommender systems using content-based and collaborative filtering approaches, such as Fab [Balabanovic & Shoham, 1997], avoid the limitations mentioned for content-based and collaborative system. Fab maintains user profiles of interest in web pages using information-filtering techniques, but uses collaborative filtering techniques to identify profiles with similar tastes. It then can recommend documents across user profiles. Users are recommended items both when the items score highly against their own profile, and when the items were rated highly by a user with a similar interest.

Compared with MARS, all these approaches hide the identity of the sources of the recommendations. Such anonymous opinions may be suitable for choosing a book or movie, but not for serious decisions.

5.2 Matchmaker Systems

In a centralized matchmaker system, a central server maintains information about user interests and the users connect to the server (in both cases, with web browsers) to discover whether they have a match. Webhound/Webdoggie [Lashkari *et al.*, 1994] and Firefly [Maes, 1994] are typical examples of centralized matchmakers.

Kuokka and Harada [1995] describe a system that matches advertisements and requests from users and hence serves as a brokering service. Also a centralized server, their system assumes a highly structured representation of user interests.

Yenta [Foner, 1997] is a decentralized matchmaker system, which tries to find other people through referrals and introductions and trying to cluster them with similar interests. There is no mechanism specifically for finding customers, so that it is harder to find someone who has enough knowledge to help.

Compared with these approaches, MARS is a totally distributed system. It does not force to cluster the agents together with the similar interests, although virtual communities with same interests will be formed via agents' interactions. These virtual communities may dynamically be changed aligned to the changes of users' interests.

5.3 Referral Systems

ReferralWeb [Kautz *et al.*, 97] is a centralized referral system, in which co-occurrence of names in close proximity on World Wide Web pages is used to suggest direct person-to-person relationship. However, it is questionable whether a co-citation matrix accurately reflects similar interests.

Several other referral systems also appeared around 1996. One is ContactFinder [Krulwich & Burkey, 1996], which is an agent that reads messages posted on bulletin boards, extracting topic areas using a set heuristic. ContactFinder posts a referral to a person when it ContactFinder persons are identified by their postings, and hence that person's communication partners are not considered. Compared with MARS, ContactFinder is limited to a company's internal community (it has been developed at Andersen Consulting's Center for Strategic Technology Research), assuming that the experts always like to answer queries from their colleagues. On the other hand, MARS takes advantage of the relationships between people and therefore it is not restricted to some small groups or companies.

The Knowwho email agent maps the user's social network by reading through his or her email messages [Kanfer *et al.*, 1997]. It then determines who is best suited to answer the user's question. Email, however, is not always a good indicator of expertise or experience. MARS is using a neighbor's profile information as a supplement, which can be obtained during interactions between agents.

6. Conclusion and future work

This thesis presents a multiagent referral system application. It discusses the ideas, design, and implementation of MARS as well as the valuable experience gained.

Social networks are a natural way for people to go about seeking information [Gladwell, 1999]. One reason to believe that referral system would be useful is that it basically models the manner in which information gathering actually works, while allowing more people to be contacted without causing unnecessary interruption.

Typically a user is only aware of a portion of the social network to which he belongs. Referral systems of the sort developed here not only help a user find experts, but also help users bridge the gap typically found between the community to which he belongs and other communities. A single agent at the intersection of two communities can rapidly help them refer to each other. By evolving into a larger community, the user can discover connections to people and information that would originally lay hidden over the horizon.

This thesis contributes to both computer science and e-commerce. The computer science contribution is the extension of the standalone multiagent systems with people, and the application of these techniques to *CSCW (Computer-Supported Cooperative Work)* knowledge management and *virtual communities in cyberspace*.

The e-commerce contributions lie chiefly in the development of the *personalized e-commerce environment* and the analysis of the model of social decision making. If we succeed in the endeavor, the work will be a significant contribution to the next generation shopbot [Clark, 2000, Doorenbos, 1997].

From what we have achieved so far, there are enhancements we can foresee. In the following discussion, we will show you some immediate improvements.

6.1 Complete the overall architecture of the system

A registration server will be introduced to help the agents new to the MARS community, or in case the agent does not have any helpful neighbors for its query or no response for all agents being asked. The agent will periodically update its profile information on the server. At the same time, the agent can decide if it wants its profile information visible to the community.

The overall architecture of the system will be made up of 3 tiers: Web browser/Java application, application server and database server (Figure 6.1).

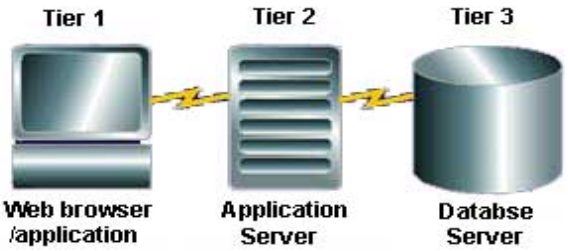

Figure 6.1 Overall architecture of the system

The first tier will use a Web browser or a MARS application. For the first time, the user will need to register with the server by visiting MARS web site (the user's profile is saved on the database). After the user downloads MARS, the MARS application will connect to the Application Server and retrieve or update the corresponding information.

The second tier will be implemented with a Web server running Java servlets. The Java servlet is able to access the database (via JDBC) and return an HTML page listing the data or store the information in Enterprise JavaBeans (EJB).

The third tier will be the back-end database server. The Java servlet can use information in the database provided that a JDBC driver exists. In our situation, we will use MS-Access so we can use the JDBC-ODBC driver that is bundled with the Java Development Kit versions 1.1 and higher.

6.2 Trust and Reputation Management

Trust is important wherever agents must interact [Castelfranchi *et al.*, 2000, Shapiro, 1987]. We consider the important case of interactions in electronic communities, where the agents assist and represent principal entities, such as people and businesses. We plan to use a social mechanism of reputation management, which aims at avoiding interactions with undesirable participants. Social mechanisms complement hard security techniques (such as passwords and digital certificates), which only guarantee that a party is authenticated and authorized, but do not ensure that it exercises its authorization in a way that is desirable to others [Yu & Singh, 2000]. Social mechanisms are even more important when trusted third parties are not available. Our specific approach to

reputation management will lead to a decentralized society in which agents help each other weed out undesirable players.

6.3 Modeling User's Expertise with Vector Space Model

Currently we are using scalar to represent user's expertise in a specific domain field. We have deployed Certainty Factor model to update the value according to user's feedback. In the future work we plan to use three vectors to adapt to the dynamic change of user's expertise.

The ISTK software package from InfoSeek Inc. was used to compute the similarity between a query and profile vector. ISTK first stems all words in any given document (e.g., removes prefixes and suffixes), computes an inverse-frequency metric for each word in the document, and computes a vector which describes the document based on these.

When used to index a large collection of documents for each colleague, ISTK normally takes a query, computes the vector associated with query and dot products the resulting query vector with neighbormodel personal vector. Dot products that are above the threshold are reported.

6.4 Evaluation

It's important to evaluate the MARS system in the real world. We will use two ways to do this: First, we can implement a simulation system, which uses MARS's referral graph, Reasoner and Learner to test if the system is able to locate desired experts via a series of referral chains. The advantage of this method is that we can simulate the situations that there are as many users as we want.

Another way to do the evaluation is to set up two groups of users. One group is using MARS and another group is using the web to find experts in a specific domain. In this way, we want to see if the group using MARS is able to locate more experts than another group and what improvements need to make to the system.

REFERENCES

[Konstan *et al.*, 1997] John Konstan, Brad Miller, David Maltz, Jonathan L. Herlocker, Lee R. Gordon, and John Riedl, 1997. Grouplens: Applying collaborative filtering to usenet news. *Communications of the ACM*, 40(3): 77-87.

[Shardanand & Maes, 1995] Upendra Shardanand and Pattie Maes, 1995. Social information filtering: Algorithms for automating "word mouth". In *Proceedings of ACM CHI'95 Conference on Human Factors in Computing Systems*, 210-217.

[Balabanovic & Shoham, 1997] Mark Balabanovic and Yoav Shoham, 1997. Fab: Content-based, collaborative recommendation. *Communications of the ACM*, 40(3): 66-72.

[Terveen *et al.*, 1997] Loren Terveen, Will Hill, Brian Amento, David McDonald, and Josh Creter, 1997. Phoaks: A system for sharing recommendations. *Communications of the ACM*, 40(3): 59-62.

[Schafer *et al*., 1999] Ben J. Schafer, Joseph Konstan, and John Riedl. Recommender systems in e-commerce. In *Proceedings of the ACM Conference on Electronic Commerce*.

[Huhns, *et al.*, 1987] Michael N. Huhns, Uttam Mukhopadhyay, Larry M. Stephens, and Ronald D. Bonnell. DAI for document retrieval: The MINDS project. In Michael N. Huhns, editor, Distributed Artificial Intelligence, pages 249-283. Pitman/Morgan Kaufmann, London, 1987.

[Kautz *et al*., 1997] Henry Kautz, Bart Selman, and Mehul Shah, 1997. ReferralWeb: Combining social networks and collaborative filtering. *Communications of the ACM*, 40(3): 63-65, March.

[Krulwich & Burkey, 1996] Bruce Krulwich and Chad Burkey, 1996. The ContactFinder: Answering bulletin board questions with referrals. In *Proceedings of the National Conference on Artificial Intelligence*, 10-15.

[Kanfer *et al.,* 1997] Alaina Kanfer, J. Sweet, and Anne E. Schlosser, 1997. Humanizing the net: Social navigation with a "know-who" email agent. In *Proceedings of the 3rd Conference on Human Factors and the Web*.

[Singh, Yu & Venkatraman, 2001] Munindar P. Singh, Bin Yu, and Mahadevan Venkatraman, 2001. Community-based service location. *Communications of the ACM*. To appear.

[Duhan *et al.*, 1997] Dale F. Duhan, Scott D. Johnson, James B. Wilcox, and Gilbert D. Harrell, 1997. Influences on consumer use of word-of-mouth recommendation sources. *Journal of the Academy of Marketing Science*, 25(4): 283-295.

[Malone, 1990] Thomas W. Malone 1990. Organizing Information Processing Systems: Parallels Between Human Organizations and Computer Systems, in: W. Zachary, S. Robertson, J. Black (Eds.), *Cognition, Cooperation, and Computation*, Ablex Publishing Corporation, Norwood, NJ, 56-83.

[ABLE, 2000] IBM's Agent Building and Learning Environment. http://www.alphaworks.ibm.com/tech/able

[Bigus, 2001] Joseph P. Bigus, 2001. The Agent Building and Learning Environment (ABLE white paper).

[Russell & Norvig, 1995] Stuart Russell & Peter Norvig, Artificial Intelligence: A Modern Approach, New Jersey: Prentice-Hall, 1995.

[Bigus *et al.*, 1998] Joseph P. Bigus and Jennifer Bigus, Constructing Intelligent Agents with Java, New York: Wiley, 1998.

[Mani *et al.*, 1998] John Mani, Bill Shannon, Max Spivak, Kapono Carter and Chrise Cotton, JavaMail API Design Specification (Version 1.1), Sun Microsystems, Inc., 1998.

[JavaMail 1.1.3, 2000] JavaMail™ API.

http://java.sun.com/products/javamail/javamail-1_1_3.html

[Gruber, 1993] T. R. Gruber. A translation approach to portable ontologies. *Knowledge Acquisition*, 5(2): 199-220, 1993.

[Castillo *et al.*, 1997] Enrique Castillo, Jose Manuel Gutierrez, and Ali S. Hadi, Expert Systems and Probabilistic Network Models, New York: Springer-Verlag, 1997.

[Kaelbling *et al.*, 1996] Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4: 237-285, 1996.

[Chris, 1989] Watkins Chrise. *Learning from Delayed Rewards*. PhD thesis, University of Cambridge, 1989.

[Pressman, 2001] Roger S. Pressman, Software Engineering: A Practitioner's Approach (Fifth Edition), New York: McGraw-Hill, 2001.

[Fowler & Scott, 2000] Martin Fowler and Kendall Scott, UML Distilled: A Brief Guide to the Standard Object Modeling Language (Second Edition), New Jersey: Addision-Wesley, 2000.

[Preece *et al.*, 1994] Jenny Preece, Yvonne Rogers, Helen Sharp, David Benyon, Simon Holland, and Tom Carey, Human-Computer Interaction, New Jersey: Addision-Wesley, 1994.

[Krulwich & Burkey, 1996] Bruce Krulwich and Chad Burkey, 1996. Learning user information interests through extraction of semantically significant phrases. In *Proceedings of the AAAI Spring Symposium on Machine Learning in Information Access*.

[Lang, 1995] Ken Lang, 1995. Newsweeder: Learning to filter netnews. In *Proceedings of the 12th International Conference on Machine Learning*, 331-339.

[Lieberman, 1997] Henry Lieberman, 1997. Autonomous interfaces agents. *In Proceedings of ACM CHI'97*, 67-74.

[Hill *et al.*, 1995] Will Hill, Larry Stead, Mark Rosenstein, and George Furnas, 1995. Recommending and evaluating choices in a virtual community of use. In *Proceedings of ACM CHI''5 Conference on Human Factors in Computing Systems*.

[Foner, 1997] Lenny Foner, 1997. Yenta: A multi-agent, referral-based matchmaking system. In *Proceedings of the 1st International Conference on Autonomous Agents*, 301-307.

[Kuokka & Harada, 1995] Daniel Kuokka and Larry Harada, 1995. Matchmaking for information agents. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 672-678.

[Lashkari *et al.*, 1994] Yezdi Lashkari, Max Metral, and Pattie Maes, 1994. Collaborative interfaces agents. *In Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI)*.

[Maes, 1994] Pattie Maes, 1994. Agents that reduce work and information overload. *Communications of the ACM*, 37(7).

[Gladwell, 1999] Malcolm Gladwell, Six degrees of Lois Weisberg, *New Yorker*, 52-63, January 1999. January 11 issue.

[Clark, 2000] David Clark, Shopbots become agents for bussiness change. *IEEE Computer*, 33(2): 18-21, 2000.

[Doorenbos *et al.*, 1997]. Robert Doorenbos, Oren Etzioni, and Daniel Weld, A scalable comparison-shopping agent for the world-wide web. In *Proceedings of the Conference on Autonomous Agents*, 1997.

[Castelfranchi *et al.*, 2000] Cristiano Castelfranchi, Rino Falcone, Barak Sadighi Firozabadi, and YaoHua Tan, Guest editorial: Special issue on "deception, fraud and trust in agent societies". *Applied Artificial Intelligence*, 14: 763-768, 2000.

[Shapiro, 1987] Susan P. Shapiro, The social control of impersonal trust, *The American Journal of Sociology*, 93(3): 623-658, 1987.

[Yu & Singh, 2000] Bin Yu and Munindar P. Singh, A Social Mechanism of Reputation Management in Electronic Communities, *Proceedings of Fourth International Workshop on Cooperative Information Agents*, pages 154-165, 2000.