

ABSTRACT

NARENDRAN RANJIT. Multiagent Referral Systems: Maintaining and Applying Trust and Expertise Models. (Under the direction of Dr. Munindar P. Singh).

This thesis addresses the challenge of assisting people to solve a problem by helping locate other people who might have the expertise required to solve that problem. To this end, the proposed system assigns an agent to each user. Further, it models a social network as a multiagent system based on the user's existing online identity. It provides heuristics by which an agent can estimate the expertise and trust to be placed in a user with whom the agent's user interacts. Our approach seeks to reduce the problem of finding an answer to a query to the problem of finding a path to a trustworthy expert who can answer the query. To accomplish this, our system uses referrals from one agent to another in the same fashion that word-of-mouth is used when people are looking for an expert. Our models of trust help establish the authenticity and veracity of the referrals and replies. This thesis describes the architecture and implementation of such a system.

**Multiagent Referral Systems: Maintaining and Applying Trust and
Expertise Models**

by

Narendran Ranjit

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Computer Science

Raleigh, NC

2007

Approved By:

Dr. Laurie Williams

Dr. Edward Gehringer

Dr. Munindar P. Singh
Chair of Advisory Committee

Dedication

You see things, and you say: 'Why'. But I dream things that never were; and I say, 'Why not?'

-George Bernard Shaw, "*Back to Methuselah*" (1921), part 1, act

Biography

Narendran Ranjit was born on the 25th of May, 1984, in Tuticorin, India. His interests are programming, photography, and reading. He got his Bachelor of Technology degree, majoring in Information Technology, from SSN College of Engineering, TN, India. He is currently a graduate student in the Department of Computer Science, North Carolina State University.

Acknowledgments

I would like to thank my adviser, Dr. Munindar Singh, for his guidance, Dr. Laurie Williams and Dr. Edward Gehringer for agreeing to serve on my committee, and my parents, for everything else.

Contents

List of Figures	vii
1 Introduction	1
1.1 Background	2
1.1.1 Modeling a social network	4
1.1.2 Finding experts in social networks	4
1.1.3 Enforcing trust and reputations in Social Networks	6
2 Heuristics	8
2.1 Modeling the social network	8
2.1.1 Identifying a data source	9
2.1.2 Modeling expertise	11
2.1.3 Modeling trust	13
2.2 Searching a social network	14
2.3 Identifying a referral chain	16
2.4 Rewards	20
3 System Architecture	22
3.1 Bootstrapper	23
3.1.1 Data Sources	23
3.1.2 Service Adaptors	24
3.1.3 XPath Processor	24
3.1.4 XSLT Generator	25
3.1.5 Content Aggregator	25
3.1.6 Expertise Calculator	25
3.1.7 Trust Calculator	26
3.1.8 Profile Builder	26
3.2 Context Analyzer Service	26
3.3 Data Store	26
3.4 Agent	27
3.4.1 Query Parser	27
3.4.2 Expertise Analyzer	28

3.4.3	Trust Analyzer	29
3.4.4	Agent Locator	29
3.4.5	Certification Authority	29
3.4.6	Message Builder	29
3.5	Messaging Framework	30
4	Discussion	31
4.1	GUI walkthrough	31
4.1.1	Bootstrapping	31
4.1.2	Performing searches	33
4.1.3	Issuing replies	34
4.1.4	Issuing referrals	35
5	Conclusion	36
5.1	Limitations	37
5.2	Future work	37
	Bibliography	38

List of Figures

2.1	An example of a social network	17
3.1	The Bootstrapper module	23
3.2	The Data Store	27
3.3	The Agent module	28
4.1	A browser with our toolbar installed	32
4.2	The bootstrapping interface	32
4.3	The import wizard	33
4.4	A search query	33
4.5	The “Pending Queries” menu	34
4.6	Answering a question	34
4.7	Replies for the query	35
4.8	The referrals menu	35

Chapter 1

Introduction

The traditional ways of getting a question answered have been to either search for the answer yourself from a repository of related information, or to ask someone else who might know the answer. The advent of the World Wide Web and the proliferation of ubiquitous search engines have made the option of personal research easier than ever by providing voluminous information at the click of a button, but there still has not emerged a practical alternative for the latter case, i.e., asking a knowledgeable person. While emails and instant messaging might have changed the mode and medium of communication, machines and technology have not had much success so far in providing effective replacements for inter-personal communication.

Despite the advances in information retrieval strategies over the World Wide Web, there are some queries that do not translate well to looking for solutions through a search engine. For example, consider a user, Alice, who plans to go for a dental check-up. She would like to find the “best” dentist around town. Alice uses a search engine to search for local dentists but she finds it tough to narrow down the responses to her particular locality. And even if she does, she might still be skeptical as not all great dentists have good Web pages and a Web page would not be an accurate indicator of the dentist’s professional capabilities.

The closest modern equivalent of the alternative scenario, i.e., asking someone else for the answer, has been through posting in online forums or bulletin boards. The simplified description of those services is that, typically, the user logs on to the Web site and posts

a question which appears on the Web site's bulletin board. The question may then be answered by some other *expert* user who is also a member registered with the service.

We argue that posting queries in forums may not result in particularly efficient results because online forums are, in a manner of speaking, defective by design. The problem is that the Internet, by supporting anonymity, complicates the ideal of making people accountable for their actions. The people answering questions, as well as the people asking them, are anonymous entities and as such there is no *social cost* incurred by either asking questions or providing misleading or false information. Also, online forums provide no mechanism to judge the expertise level of a user or any other indicator of the accuracy of the answers the user would provide.

The problem of enforcing accountability over anonymous entities over the Internet can be addressed to an extent by modeling the participants in the forum as a virtual *social network* which comprises of a series of inter-connected *nodes* and *edges*. The nodes in our case denote the users of the system and the edges indicate a relationship between the users. The other nodes directly connected to the user node, i.e., the nodes that user node can directly communicate with, are termed to be that user's friends.

Hence the problem of finding an answer to a question can be reduced to the problem of finding a path to a particular node on the network, i.e., locating an expert in a particular domain by navigating through a chain of referrals obtained from the user's friends in the network.

We aim to provide a practical solution to the problem of extracting reliable responses to user queries by developing a *multiagent referral system*, using baseline information gathered about the user's existing social network. We then propose to determine an strategy to efficiently identify experts in this social network and provide a way to establish a *referral chain* between the user and this expert based on the user queries. We also seek to establish a dynamic distributed *trust rating* for each user in the network, which would indicate one user's personal belief about another's veracity.

1.1 Background

For our purposes, an *agent* is a software module that performs operations on behalf of a user. Agents are capable of acting as autonomous entities and making independent

decisions. Each agent is self-contained and designed to perform actions to satisfy internal goals based upon its perceived environment.

Agents interact with each other to form *multiagent systems*. A multiagent system is a system composed of several agents, collectively capable of reaching goals that may be difficult to achieve by an individual agent or monolithic system. Each agent in a multiagent system is capable of responding to external events as well as interacting with other agents to perform certain tasks.

A *referral system* is a multiagent system where each agent is capable of producing “intelligent” referrals to other agents in the network, and can act upon referral information obtained from other agents.

An *expert* is a user who has domain knowledge about the user query. The goal of our system is to match a user with an expert based on the domain of the user’s query.

To be practically feasible, our approach for matching a user query with accurate responses from domain experts must address the following challenges:

- Modeling a social network based on information garnered from the user’s existing online identity along with information about his social network.
- Building personalized *expertise models* for each agent, to help in identifying experts in the social network and providing a path from the user to the expert.
- Building personalized *trust models* for each agent, to assign and calculate trust values dynamically for a user with respect to another user.

The ultimate goal of the trust and expertise models is to assist the agent in finding experts in the system and identifying an ideal *referral chain* from the user to that expert. A referral chain comprises a series of *links*, each link indicating a referral from one agent to another agent on its Friend List. Each agent in our system only has knowledge of the user’s direct friends without any idea of the underlying structure of the entire social network. Hence the only way for an agent to contact another agent that it does not already know of is through a referral from an intermediate agent. A referral chain could hence be considered as a series of introductions among strangers through common friends.

1.1.1 Modeling a social network

To create a social network model, one would need to extract information from the user about his acquaintances and their areas of expertise. The easiest way to do this would be assign the task to the user. For example, we might provide him a questionnaire asking him to name his friends and their interests and do the same for his friends and so on. This is the approach adopted by most of the currently popular social networking Web sties.

Another alternative would be to devise an agent that scouts for online *web-trails* left by the user and find information about him with little to no user input required. In our context, a web-trail refers to any activity performed by the user over the Internet for which there exists some kind of log or record. Considering the growth of the Internet, it is not unrealistic to assume that a significant number of people have already established a significant online presence. Typical web-trails would include email messages or a list of the web-sites the user is a member of. The key challenge to selecting a data source would to find one from which enough meaningful information could be extracted to be useful, while still protecting the user's privacy.

Foner proposes a system called Yenta which mines a user's email messages to build a user profile [Foner, 1997]. Users of the system are then clustered together based on similarity of their profiles. In this case, a user's friends are people who share similar interests rather than people whom he might actually know in the real world.

Schwartz and Wood also build a network model by parsing email logs at centralized servers, obtained from 62 independent participating sites [Schwartz and Wood, 1993]. They define a set of heuristic algorithms to uncover shared-interest relationships among people, based on the history of communications among them.

The ReferralWeb initiative by Kautz, Selman, and Shah discourages mining of email logs, under the concern for privacy, and uses data found about the user in the public domain; including links found on home pages, lists of co-authors in technical papers and exchanges between individuals found in netnews archives [Kautz et al., 1997].

1.1.2 Finding experts in social networks

Before approaching the problem of searching for an expert in a social network, it is prudent to establish that the social network is indeed, searchable. One of the clas-

asic case studies in this regard is the *Small World* experiment conducted by Travers and Milgram [Travers and Milgram, 1969]. The purpose of the experiment was to transport a package from Nebraska to Boston - the catch was that the package could only be transferred from each person to one of his acquaintances. Each person potentially has numerous acquaintances but is forced to choose one among them to carry forth the package. Studies by Muhammed and Watts showed that the next package-bearer was chosen primarily based on proximity or similarity of profession [Dodds et al., 2003]. Ultimately the package was delivered, and through this experiment, Travers and Milgram established that the social network is indeed searchable and that the average length of an acquaintance chain is six; leading to the popular phrase *six degrees of separation*.

Decker, Sycara and Williamson [Decker et al., 1997] propose a match-making system based on *middlemen*, which are third party agents whose goal is to match service requesters with consumers. Their solution is based on a centralized architecture.

Another solution to the problem would be to make use of omniscient recommender systems. This typically consists of an agent which observes the choices made by most users in the system, and recommends the most commonly used options to the user. Such a system has been proposed by Schafer, Konstan, and Riedl [Schafer et al., 1999].

Yu and Singh propose a referral system where queries are matched to users based on between the keywords present in the query and the keywords associated with each user denoting its *expertise* [Yu and Singh, 2003]. Their algorithm also takes into consideration *history* information associated with an agent's prior referral record.

Adamic et al. put forth a *Best Connected Search* algorithm which propagates a query by passing it to the most connected nodes in the network; in our context, to the users with the most friends [Adamic et al., 2001]. Adamic and Adar supplement this proposal, using simulation data obtained from parsing email logs from an organization, by adding two additional parameters to the criteria for choosing a node to forward the query to: relationship in the organizational hierarchy and geographical proximity [Adamic and Adar, 2003].

Weak Tie Search is a strategy by Granovetter [Granovetter, 1973]. He advocates forwarding queries based on the strength of the relationship among users in the network. Queries are always to be forwarded to the users whose relationship strength with the current user is the weakest.

Zhang and Ackerman perform simulation runs using performances of several ex-

pertise location algorithms, using the Enron email data-set as the source for their simulation [Zhang and Ackerman, 2005]. They base their evaluation criteria on factors like social cost incurred by the query and accuracy of the results among others.

1.1.3 Enforcing trust and reputations in Social Networks

Trust is an abstruse concept with multiple interpretations. The challenge of identifying the underlying factors which constitute trust and precisely quantifying those factors would be a task whose roots are more within the realm of philosophy and psychology than computer science. Even within computer science, the issue of trust has been studied and analyzed in several different contexts. In this section we review some of the major contributions.

Marsh formalizes trust as a computational model based on social and psychological factors [Marsh, 1994]. His complex model considers history and observed behaviors of entities to quantify reliability of future interactions. Sztompka defines trust in terms of beliefs and commitments; he postulates trust as the belief that an agent honors its commitments [P., 1999].

Kamvar et al. put forth an algorithm for calculating trust called the *EigenTrust* algorithm [Kamvar et al.]. *EigenTrust*, based on the PageRank algorithm used by Google, is primarily used in peer-to-peer networks. A global trust value for each peer is calculated by aggregating individual trust values from others peers, based on transaction history. Ziegler and Larsen propose an *AppleSeed* algorithm which is also based on calculating an EigenVector [Ziegler and Lausen, 2004]. The *AppleSeed* algorithm, however, provides localized trust ratings for peers.

Richardson et al. assign a trust rating to an edge between two nodes in a network and calculate trust between two nodes by aggregating values along the path between them [Matthew et al., 2003]. Gil and Ratnakar provide a similar solution for trusting information sources over the semantic web by aggregating individual feedback about the sources from other users [Gil and Ratnakar, 2002].

To make trust quantifiable, it would be convenient to represent it as a numerical value. Some approaches advocate a binary trust value, 0 indicating lack of trust and 1 indicating trust [Golbeck and Hendler, 2006]. The Semantic Web Trust project uses a scale from 1-10 [Golbeck and Hendler, 2004]. Richardson's approach defines trust within the

range 0,1 [Matthew et al., 2003].

Chapter 2

Heuristics

In this chapter we seek to outline some of the key challenges towards construction a multiagent referral system. We provide a brief review of existing research towards addressing those challenges and present the strategies adopted by our system towards overcoming those challenges.

2.1 Modeling the social network

To create a social network model for a user the essential information required would be about his interests, his acquaintances, and their areas of expertise. This information can either be obtained directly from the user [Orkut; Facebook] or derived by the system from the user's existing online identity [Foner, 1997; Kautz et al., 1997; Schwartz and Wood, 1993].

Having the user manually provide the required information would not be practical; the users are unlikely to spend the time and effort to manually furnish the information required, and any such information would always be prone to user error. An ideal data source for our system would be one which can provide our system with the following information either explicitly, or in a format from which this information can be inferred:

- The list of other users a user is connected to. We term this as his *Friend List*. We use the term *friend* broadly, using it to include the user's acquaintances as well.

- The areas of expertise of each user in his Friend List.
- The strength of association between users.

The Friend List is required for obvious reasons- to let the agent know who to talk to next on the network. This list is local for each user; each user has knowledge of only his friends and not who the friends of his friends are.

As a precursor to our goal of finding a domain expert in the system, we need to establish the initial domain areas of each user. We hence need to determine the areas of expertise for every other user in the Friend List. We use the term *expert* loosely, using it to denote any user who has some knowledge of a particular domain. Hence a domain could have multiple experts and a user can be an expert in multiple domains.

We recognize that, in a social network, not all relationships are to be considered equal. There is *strength of association* between two individuals [Granovetter, 1973]. For example, if Alice is connected to Bob within the network, it needs to be determined if Alice and Bob are close friends or just casual acquaintances. This strength of association varies and may not always be symmetrical. Alice could like Bob more than Bob likes Alice. This information becomes crucial while calculating the trust rating for each user, as will be shown in section 2.3. We act on the assumption that a user would tend to place a higher belief in the accuracy of a statement if it came from a close friend, as opposed to a statement from a casual acquaintance.

The key challenge to selecting such a data source would be to find one from which enough meaningful information could be extracted to be useful, while still protecting the user's privacy.

2.1.1 Identifying a data source

Yenta used the email messages of a user to build up his profile model [Foner, 1997] and Schwartz and Wood mined email logs from mail servers for this information [Schwartz and Wood, 1993] while ReferralWeb scouted for information in the public domain, from Web sites and publications [Kautz et al., 1997]

Scouting for information in the public domain is fraught with uncertainty. Names on the Internet are not unique and a simple search query using the first name and last name of the user would typically throw up hundreds of results, most of them irrelevant to

the current user. We would then need to make the system robust enough to weed out the spurious results and use only the proper matches. Incorporating such capabilities within our system would be outside the scope of our project and also would not provide enough benefits to be practical.

Emails could indeed prove to be a valuable information source. However, we decided against parsing emails from a central mail server on grounds of privacy; the choice of sharing information, even if it is anonymized and can in no way be tracked back to the sender, should lie entirely with the user. In other words, the user should make the decision to opt in to provide his data for analysis, and should not have to opt out of having our system make use of his data which we already have. Hence our system uses information gained from mining a user's email exchanges stored in his inbox, and only after the user explicitly requests it. No copies of the actual emails are stored anywhere, except the temporary information present in memory while parsing the emails.

Each unique "From" address field in the message header is assumed to be from a *friend* of the user. Major email providers weed out the spam even before the message reaches the user's inbox; hence our sample data would have relatively less noise. We also ignore emails that have more than three people mentioned under the "To" field in the header under the assumption that this would be likely to be chain mail. Our systems supports mining emails from the Yahoo! [Mail] and GMail [Gmail] email providers, though the user can easily add another provider of his choice later on if he wants to, provided that the provider allows third-party access to such data and the data is either in a format supported by our system or can be converted to a format understood by the system.

However, while analyzing emailing patterns might be useful for modeling the organizational hierarchy of an enterprise, they would not be an accurate indicator of that individual's social network. For example, I would be less likely to be communicating with my room-mate through email than I would with my professors or business colleagues. This has been validated through studies by Boase et al [Jeffrey Boase and Wellman, 2006]. Hence, while mining emails could prove to be a valuable repository of information, our system should ideally cross-reference information from other sources to supplement it.

The best possible data source that we could have used to model a social network is another social network. Our system hence allows the user to import his profile and friends from any existing social networking Web site he is a member of, again assuming the website allows third-party access to such data and the data is either in a format supported by our

system or can be converted to a format understood by the system.

2.1.2 Modeling expertise

Our system categorizes expertise as keyword-value pairs associated with each user; the keywords, also called *tags*, indicate the domain of the expertise and the value indicates the depth of knowledge in that domain and is hence an indicator of the degree of expertise of the user. A high value would indicate a user with extensive expertise pertaining to that domain and a low value would indicate that the user is barely acquainted with the domain.

We recognize that it may not be possible to exactly identify the expertise of each user from the limited information we have access to, and doing so is not our goal. Our goal is to initially provide our system with enough data to make an educated guess about this attribute based on the data available during the bootstrapping process and then subsequently refine this based on future interactions between the users to get a more accurate estimate. Tags may be added or removed, and the tag values may be modified based on subsequent interactions among the agents.

We derive the expertise information from messages sent or received by the user, obtained from the data source. Messages can be broadly classified under two types: messages that provide information and messages which request information. Messages which provide information are unlikely to provoke a significant reply, other than message acknowledgments or replies designed to remove ambiguity in the original message. For example, if Alice sends an email to all her friends informing them that she just got a promotion, it is unlikely to generate any replies other than congratulatory messages and analyzing those emails would not significantly add to the knowledge bank of our system. Hence for our analysis we deem messages which haven't elicited a reply to be insignificant for our purposes and consider only *conversations*, i.e., messages which have been replied to.

We assume that, if a user initiates a conversation with another upon a particular topic it implies that the sender considers the recipient to be an expert on that particular topic. Alice would not send Bob a message about programming unless Alice was under the impression that Bob knew enough about programming to reply to her message. For this scenario, our system should determine Bob to be an expert on "programming" and associate a tag named "programming" alongside his entry in the Friends List.

To determine a set of tags from messages we initially need to remove irrelevant

words from the message such as conjunctions, prepositions, and adjectives. We make use of the Yahoo! Context Search engine for this purpose [Search]. Given a message, the Web service returns a list of keywords found in that message. Our context search service, however, cannot distinguish between different variations of the same words; for example, programmer is treated in a different context from programming. Hence we needed to devise a method to equate different variations of the same word to the same tag in our expertise list.

One way to resolve such ambiguity would be to build an ontology of words and trace each occurrence of a word to the root from which it is derived. We decided against using external sources providing access to existing ontologies (e.g., WordNet) on the grounds of making our system as self-contained as possible, and removing dependencies on external Web services wherever possible. Building an ontology is not a trivial task, and it would not be practical to build one from scratch for purposes of this project. Hence our system differentiates between different variations of the same word by removing the suffix of the word and reducing it to its *stem*. We use Porter's suffix-stripping algorithm for this purpose [Porter, 1997]. This is the same approach followed by Yu and Singh in their proposed mechanism for searching social networks [Yu and Singh, 2003].

Once we obtain the keyword list, we still need to assign weights to individual keywords. This step is required to determine the strength of expertise; Bob who has sent Alice a single message about ipods has to be given a lower expertise rating than Charlie who corresponds regularly with Alice about iPods. Various algorithms have been suggested for term weighting in Information Retrieval using different techniques ranging from on linear algebra [Berry et al., 1995] to probabilistic models [Hofmann, 2001]. For our system we chose the *TF-IDF* (term frequency-inverse document frequency) approach suggested by Salton and Buckley [Salton and Buckley, 1987] as it is both practical and generated accurate results in our test cases.

According the TF-IDF model, the weight vector for a person p who is an expert in N domains, i.e., who possesses an expertise vector with N tags, is $v_p = (w_1, w_2, \dots, w_N)$

$$\text{where } w_i = \frac{tf_i \log\left(\frac{P}{pf_i}\right)}{\sqrt{\sum_{t=1}^n \left[\left(tf_t \log\left(\frac{P}{pf_t}\right) \right)^2 \right]}}$$

tf_i = number of times a term i is mentioned by a person

pf_i = number of people mentioning the term i

P = total number of people

The tf_i term in the numerator indicates information local for each person. The $\log\left(\frac{P}{pf_i}\right)$ term denotes the inverse document frequency and is an aggregation of global information from a collection of people. In our case, this indicates the popularity of a term among a group of people. The product of tf_i and $\log\left(\frac{P}{pf_i}\right)$ is normalized to prevent a bias towards longer documents. The intuition behind the equation is that frequently used words are assigned lesser weights compared to rarely used words.

A high weight in TF-IDF is reached by a high term frequency (in the given document) and a low document frequency of the term in the whole collection of documents.

In addition to modelling the expertise of the user’s friends, we would also need to determine the user’s interests. In this regard, we assume the user receives messages about a topic because he is considered an expert on that topic by the sender. Hence the expertise vector for the user would be derived from the messages received by the user neglecting the messages which he initiates.

2.1.3 Modeling trust

While deriving the user’s Friend List from the data sources identified in section 2.1.1 is a trivial task, easily accomplished by reading message headers from mails or accessing contact lists from social networking Web sites, the process of determining the *relationship strength* from such data requires performing some additional operations. We consider relationship strength to indicate the trust between two users. In this context, we focus on the social aspects of trust, and hence limit its definition to simply indicate *the degree of confidence or faith*.

Trust between users is generally *asymmetric*. Bob does not have to trust Alice as much as Alice trusts Bob. Also, trust is largely *personalized* and having a single global trust value for a user would not accurately reflect the real-world scenario. Alice is not obligated to trust Charles just because everyone trusts him completely. Alice might distrust him based on her personal opinion which would overrule popular perception. Trust is also *intransitive*. Even if Alice’s friend Bob trusts Charles, Alice might still not trust him. We seek to capture and preserve these properties of trust in our model. As in the previous case, our heuristics for determining relationship strength are intended as more of an initial fishing expedition rather than as a failsafe method.

We model trust based on the assumption that people are more likely to be friends with other people with similar interests. Contrary to the pop-psychology perception that “opposites attract”, studies have established that we are attracted to other people based on our similarities rather than our differences [Buston and Emlen, 2003]. Hence, relationship strength is modeled by the system based on the degree of similarity between users, which is determined from the expertise model created for our system as described in the previous section. The Yenta system also follows the same principle, by grouping similar users within the same cluster [Foner, 1997].

We calculate the similarity between vectors, i.e., between the user’s expertise vector and the expertise vector of everyone else on his Friend List. Calculating similarity between two vectors can be accomplished by calculating the cosine of the angle between them. Hence, given the expertise vector for the user $U_j = (w_{j1}, w_{j2}, \dots, w_{jn})$ and the expertise vector for a friend $F_i = (w_{i1}, w_{i2}, \dots, w_{in})$ the similarity between the user and the friend can be calculated by the following equation:

$$sim(U_j, F_i) = \frac{\sum_{t=1}^n w_{jt}w_{it}}{\sqrt{\sum_{t=1}^n (w_{jt})^2 \sum_{t=1}^n (w_{it})^2}}$$

2.2 Searching a social network

In this section we present a distributed algorithm for locating experts in the social network. Given a search query a user’s agent should either return a list of referrals to other agents in the system who might be potential experts or it should return a reply from an expert. The only information each agent has access to, besides the search query, is its Friend List which contains a list of all the agents it can communicate with directly, the strength of association, and tags denoting the areas of expertise for each user.

The Best Connected Search (BCS) approach advocates forwarding a query by passing it to the users in the network having the most friends [Adamic et al., 2001] while the Weak Tie Search (WTS) algorithm forwards the query to the user whose relationship strength with the current user is the weakest [Granovetter, 1973]. Yu and Singh’s strategy forwards based on the similarity between the expertise vector of the user’s friends and the query vector [Yu and Singh, 2003]. The High Density Search (HDS) algorithm forwards queries to the user with the least number of uncommon friends [Wasserman et al., 1994].

While choosing an optimal search strategy, speed and computational complexity

should not be the only factors determining effectiveness of the strategy; the psychological and social costs associated with the query propagation strategy should also be considered. *Social cost* refers to the intangible effort made by a user in performing a service for another user. It assumes that people always perform services in expectation of future returns. For example, if Bob answers ten different questions from Alice, there is a social cost incurred by answering each query and Bob is hence entitled to expect Alice to answer questions he might have in the future. If Alice refuses to answer, then Bob might also stop answering all future questions from Alice.

In our system, all referrals are generated by the agent on behalf of the user and there is hence no social cost associated with a referral. However, the responsibility of actually sending a reply is on the user and each reply to a question hence incurs a social cost.

Of the various search strategies, Yu and Singh’s approach seems to be the most attractive. Messages are forwarded based on perceived expertise hence the probability that the query will elicit a reply rather than a referral would be higher than in the case of randomly selecting a node to forward a query to. Even if a query is not answered, there is some information gained, namely the expertise value for that user will be decreased. This observation is borne out by simulation runs performed by Zhang and Ackerman where they found this strategy to have a high success rate, outperformed only by HDS and BCS [Zhang and Ackerman, 2005]. HDS and BCS algorithms, however, require knowledge of the underlying network structure, namely, information pertaining to friends of friends which we do not have access to. Hence we adopt a simplified version of Yu and Singh’s approach, where we define similarity to be the number of keywords the query vector has in common with the expertise vector for each user. We term this the *Similarity Search strategy*.

One of the problems with similarity search is that it assumes knowledge of a lot of information of both the query and the expertise list of each friend. Such information may not always be available from our input sources. Hence, in cases where there is no match between keywords in the query and keywords in the expertise list for any user, an alternate search strategy has to be adopted. Assuming no additional knowledge of the social network besides our friend list, the alternative strategies that present themselves are forwarding the query to the users with the highest relationship strength or the more unintuitive option of forwarding it to the users with the lowest relationship strength.

While forwarding to people with the highest strength, i.e., people we trust a lot,

might appear the more attractive option at first glance, we found a WTS to be the more effective option because of the following reasons :

- By definition, the difference between a friend and an acquaintance is that we would typically know a friend better than we would an acquaintance, i.e., we would be likely to possess more knowledge about a friend than an acquaintance. Based on that assumption, it follows that we would hence be less likely to know the areas of expertise of acquaintances than those of close friends. It is hence probable that the acquaintances might possess domain information the user is not aware of and hence it would make more sense to forward the queries to them.
- The query is also likely to propagate further if forwarded through acquaintances. This is because our acquaintances (weak ties) are less likely to be socially involved with each other than our close friends (strong ties) [Granovetter, 1973].

2.3 Identifying a referral chain

In this section we describe how the agents use the trust and expertise models created by the system, as described in the previous sections, to identify a referral chain from the user to the expert. We provide an overview of the protocols used by our system for the propagation of queries, replies, and referrals throughout the network while preserving the properties of the trust model defined in the previous section. We then identify vulnerabilities in our protocol which could be exploited by a malicious user to compromise the system and discuss solutions to overcome those vulnerabilities.

Once the user initiates a query, which can be any question phrased in natural language, the agent extracts the keywords from the query and chooses a select few agents from its Friend List to forward it to, based on either the Similarity Search algorithm or the Weak Tie Search algorithm whichever it deems to be appropriate, in the hopes of finding an expert who can answer the query. We illustrate this process using the social network shown in figure 2.3.

The silhouettes depict users of the system. The lines in between two users indicate existence of a relationship between them. From the diagram, Alice knows Bob, but does not directly know Diana. The numbers on each line denote the strength of the relationship;

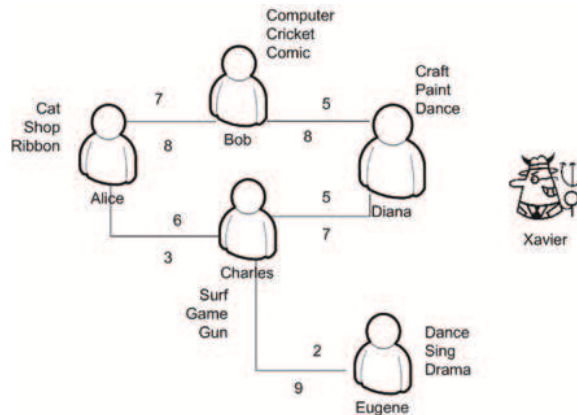


Figure 2.1: An example of a social network

the number below the line indicates strength of the relationship from left to right and the number over the line indicates the strength from right to left. In this example, Alice trusts Bob “8” on a scale of 1 to 9 and Bob trusts Alice “7”. To get a better intuition behind the connotation of the numbers, it can be taken to imply that Alice believes Bob will come up with a good answer about 80 percent of the time, for example. Beside each user, their expertise list is depicted. Details about the weights assigned to each expertise are omitted for the sake of simplicity.

Once an agent receives a query it can choose between ignoring it, answering it or referring the querying agent to other users in its Friend List who might be experts in the domain of the query. This choice is made by the agent after analyzing its expertise model. If it finds a match between the query vector and its own expertise vector, the agent flags the query for attention by the user. The user then reads the question and either ignores it or provides a reply which is then propagated back to the initial agent. If no match is found the query is again forwarded based on Similarity Search or Weak Tie Search.

Theoretically, such a query could propagate through the system forever. Hence, there has to be some *time-to-live (TTL)* associated with the query after which it should no longer be forwarded. Usually TTL is measured in terms of number of hops, where a hop refers to each intermediate node a query is passed to. For example, the Gnutella P2P network uses a default TTL value of 7, which it decrements after each hop.

In our system, we argue that the TTL value would be better used as a function of

the trust value of each user rather than the hop count. Ideally queries propagated through a strongly trusted chain should have greater precedence over queries traveling down a path of lweak trust. Queries from Alice, who is highly trusted by her peers, should be given greater precedence over queries from Xavier who has a bad reputation. Also, we need to calculate the implied trust between users who are not directly connected.

While the trust between directly connected users can be obtained using the calculations mentioned the previous section, calculating the trust between users who are not directly connected would require a different approach. For example, if Alice sends Charles a query about dancing, Charles would refer her to Diana and Eugene. As trust is intransitive, Charles would also need to tell Alice how much she can trust answers from them, i.e., he has to indicate in some way that he considers Eugene to be more reliable than Diana. Besides that, Charles also has to let Diana and Eugene know how important he considers Alice’s query to be.

We propose a simple calculation to derive transitive trust. If A_i, A_j and A_k are users in the network and if there exists a trust value t_1 for the edge between A_i and A_j , and a value t_2 for the edge between A_j and A_k , and there is no direct edge connecting A_i and A_k , we define the trust t_3 between A_i and A_k as : $t_3 = (t_1 \times t_2)/10$

We use this representation of the implied trust as the TTL for each query and term it the *priority value*. The priority value is initially assigned by the user initiating the query and is decremented at each intermediate node as the query propagates over the network. Each intermediate agent decreases the priority value based on how much it trusts the initiator of the query.

The user initiating the query would naturally provide it the highest priority possible. For example, for the social network depicted in Figure 2.3 consider Alice initiates a query and sends the query message to Bob with a priority value of 9. Bob would reduce the priority of the query to 7 and reply back with a referral to Diana. Alice would then get in touch with Diana, providing her with the reference from Bob. Diana interprets the priority value to imply that she can trust Alice “7” on a scale of 1 to 9. She could then reply back to Alice or refer her to another user and so on.

The message is dropped by an agent if the priority value decreases to 1 or any user-defined minimum threshold. A paranoid user could set a threshold of 7, indicating that he would not reply to any messages whose priority number was less than 7. But by adopting such a policy, he would be forgoing a chance to expand his social network which

would in turn imply a smaller number of people willing to reply to his queries because of the higher social cost involved.

Such a scheme would imply that the maximum length of a referral chain is nine, which happens if all the intermediate nodes have a high reputation with each other. As it has been established that the average length of a relationship chain is six [Travers and Milgram, 1969], this would not adversely affect the user's chances of getting a reply to his query. Also, each user would be judged by the company he keeps; if all Xavier's friends have a trust value of 1 with their friends his queries will not propagate through more than one level of the network.

However, the scheme outlined so far would not be robust. It is open to attack by *impersonation* and *message tampering*. Impersonation happens when one user takes over the identity of another. Xavier could contact Diana pretending to be Bob and get her to answer questions she normally wouldn't. Another possible attack could be one user modifying the contents of the message before sending. When Alice gets a referral to Charles from Bob, she could simply reset the priority value of the message, i.e., the referral, to a higher value before forwarding it.

To avoid these kinds of attacks each message would need to be digitally signed and encrypted. We use *public key encryption* for this purpose. Public key cryptography involves a pair of keys—a public key and a private key. The private key is kept secret, while the public key may be widely distributed. The keys are related mathematically, but the private key cannot be practically derived from the public key. A message encrypted with the public key can be decrypted only with the corresponding private key.

Hence, in our system this would imply that if two users are friends, each should know the other's public key. When Alice sends a query request to Bob, and Bob replies back with a referral to Diana, this reply should be encrypted with Diana's public key and hence cannot be decrypted or tampered with without possession of her private key. Also, Xavier cannot counterfeit messages as he is not Diana's friend and hence does not have access to her public key which would be required to encrypt the messages.

While encryption and decryption using the public key and a private key can be accomplished locally by each agent, there is typically a central certification agent in charge of allocating public keys to everyone in the network. This would typically be done by a certification authority though distributed versions of public-key encryption have also been proposed [Frankel and Yung, 1998]. We do not address the problem of initial distribution

of keys and hence our system assumes that friends in our network already have a public key and private key assigned to them.

2.4 Rewards

Our system would need to define a mechanism wherein agents who provide accurate replies or referrals which lead to replies are rewarded, and agents who either fail to produce a reply or provide an incorrect reply are penalized, in order to maintain the integrity of its trust and expertise models.

Ideally, after an acceptable reply is elicited from an expert for a query initiated by the user, the rewards should propagate to each intermediate agent involved in that referral chain. However, in our system, each agent only stores information about the user’s direct friends. Hence it cannot update trust and expertise values for users who are not in its Friend List.

A possible solution to this problem would be to have the agent add all the intermediate nodes in the referral chain to its Friend List, and then assign trust and expertise values to those agents. But we decided against this approach for the following reasons:

- This could ultimately lead to a scenario where each user’s Friend List includes every other user in the social network, and propagating a query would be akin to broadcasting it to every agent available.
- We assume friendship to be a bi-directional relationship, i.e., Alice cannot have Bob in her Friend List unless Bob also has Alice in her Friend List. We also model contacts in our system based on real-world relationships. Consider a situation where Bob refers Alice to Diana who replies to her question. Diana might not want to be added to Alice’s social network just because she did her a service. They may well become friends and add each other to their contact lists later on, but this should be a choice made by the user, not the agent, and the unpredictability of that event would preclude us from using that assumption to build our rewards scheme upon.

Another solution could be to have the querying agent send “Thank you” notes down the referral chain after receiving an acceptable reply. This note would inform the intermediate nodes of the success of the transaction, and they would in turn increase their

personal trust value of the person they referred. While this solution appears more attractive, this could be manipulated by a malicious node which sends spurious Thank You notes to artificially increase the trust value of the target.

We propose a simplified scheme where if t denotes the existing trust value of the neighbor agent, i.e., the agent closest to the initiating agent in the referral chain, its trust value is increased by $10 - t$ percent for a good answer and is penalized for a bad answer by decreasing its trust rating by t percent. The maximum value a trust rating can take is 9 and an agent with a trust value of 9 is not rewarded any further. It might be argued that this would make the agent lose the incentive to provide results in the future, but as this trust is personalized, the other agent would have no way of knowing its trust value.

Chapter 3

System Architecture

This section describes the architecture of our proposed system. After a brief description of each component involved, we provide a description of how each component in the architecture interacts with the other components to achieve the desired functionality of the system as a whole.

The Bootstrapper is responsible for creating a *profile* of the user and modeling his social network using baseline information obtained from external sources. The profile contains the user's Friend List as well as the trust and expertise models. Bootstrapping is a one-time process initially performed when the user registers with our system. However, the components of this module can also be subsequently invoked if the user desires to add information from other sources into his profile. The bootstrapper builds the profile and stores it in the *datastore*. The Agent represents an independent module assigned to perform all operations on behalf on the user such as queries, responses, and referrals. It performs these tasks based on information it infers from the user's profile present in the datastore. The messaging framework provides a means for inter-agent communication. The Context Extraction service is a third-party Web service used by our system to extract context keywords from messages.

3.1 Bootstrapper

The goal of this module is to create a profile for the user. It creates a representation of the user's social network based on the user's existing online identity. It also generates trust and expertise models using baseline information obtained from external user-defined data sources. Figure 3.1 illustrates the components which make up the Bootstrapper.

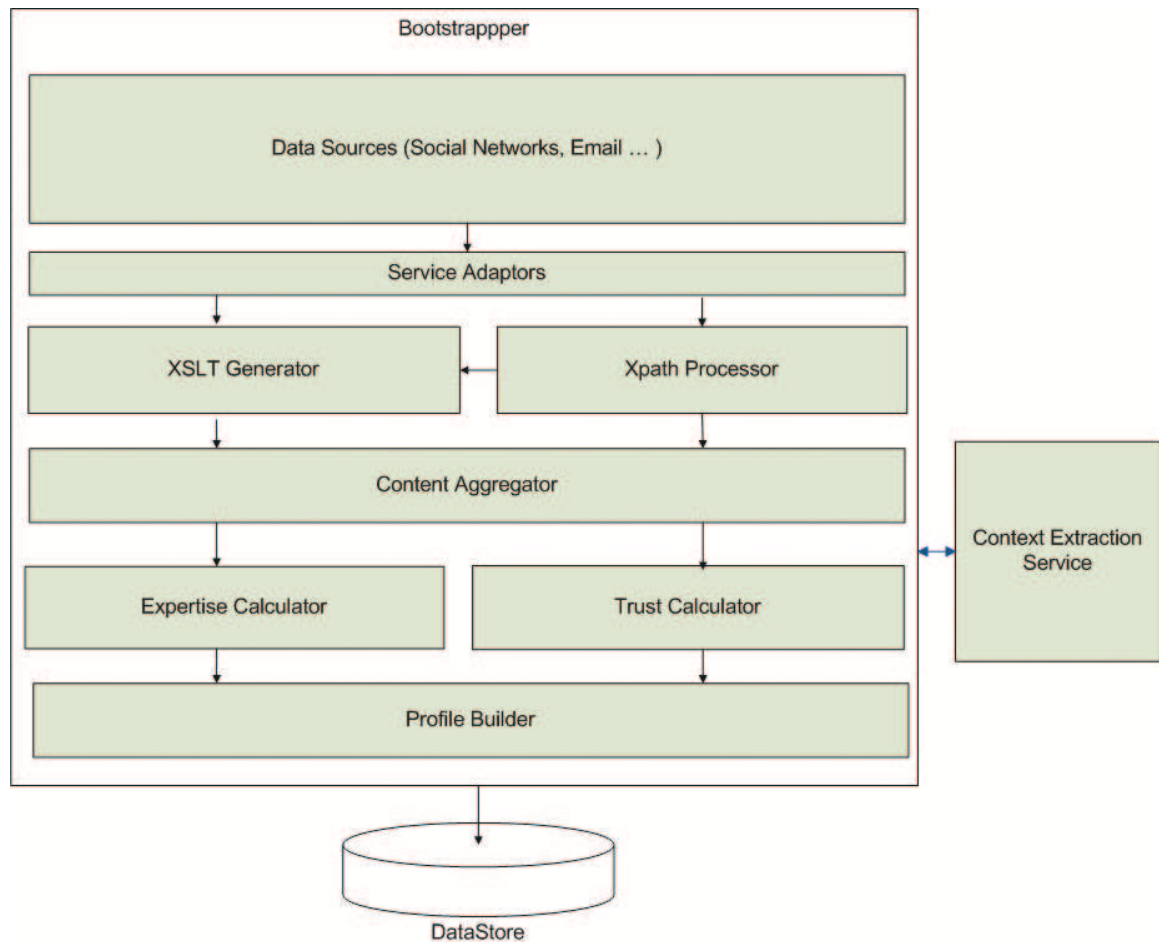


Figure 3.1: The Bootstrapper module

3.1.1 Data Sources

Our system can mine a variety of existing sources to gather information about the user's network. The sources may be, but are not limited to, social networks and email mes-

sages. Any source can be used as long as it can be represented through a XML document which conforms, either directly or upon application of an arbitrary XSLT template, to the XML schema specified by our system. XSLT (Extensible Stylesheet Language Transformation) is an XML-based language used for the transformation of XML documents. The schema used by our system has been constructed after analyzing data feeds from various sources like Facebook, Gmail and Yahoo! mail and we expect it to be easily adaptable for use with most other such sources.

3.1.2 Service Adaptors

This component establishes the nature of the input source specified, be it from a social network or email, and also the input type. The input could be either a XML document or a Resource Description Framework (RDF) document. It provides the user a hook to use to plug in that information into our system; i.e., it creates a one-to-one mapping between elements in the XML document provided by the source and entities in the XML schema specified by the system. This mapping is determined primarily through information provided by the user. For example, the user is asked to provide the name of the elements which corresponds to his jabber id in his XML document. This element is then mapped on to the 'JabberID' entity in the XML Schema.

3.1.3 XPath Processor

XPath (XML Path Language) is an expression language for addressing portions of an XML document, or for computing values (strings, numbers, or boolean values) based on the content of an XML document. The XPath language is based on a tree representation of the XML document, and provides the ability to navigate around the tree, selecting nodes by a variety of criteria.

The XPath processor generates the XPath expressions required to uniquely identify the elements that have to be mapped on to each other. In the previous example of having the user point out the equivalent of the Jabber ID field in his XML document, the processor generates an Xpath expression to remove ambiguity and uniquely identify the target element.

3.1.4 XSLT Generator

Once the input source has been validated and the input is fed through the adaptor, it has to be transformed into a format that can be understood by our system. This transformation is done by applying an XSLT template upon the document. This component takes the XPath expressions representing the input mappings and uses it to dynamically generate an appropriate XSLT template which can then be applied upon the XML document to transform it to a format readable by the system.

3.1.5 Content Aggregator

The content aggregator combines information from all the different sources specified by the user into a single local store, i.e., it aggregates the transformations obtained by applying the generated XSLT templates upon the source document. For example, if the user has specified FaceBook and Gmail as his information sources, two different XSLT templates are generated by the XSLT generator component. The content aggregator component combines output of these files into a single user model.

3.1.6 Expertise Calculator

We denote areas of expertise as *tags* associated with each user. A tag is a (relevant) keyword or term associated with or assigned to a piece of information, thus describing the item and enabling keyword-based classification of information it is applied to. Tagging is usually associated with the semantic web, and is typically used in dynamic, flexible, automatically generated Internet taxonomies for online resources. In our context, the resources are the users and tags are metadata associated with each user describing his knowledge, albeit in a limited scope.

Expertise information is stored as keyword-value pairs in our system, the tag and the value associated with that tag which indicates the strength of expertise. The expertise calculator extracts domain information from the user's messages and uses that information to determine the tags. The tag value is calculated by the TFIDF calculation detailed in section 2.1.2.

3.1.7 Trust Calculator

While the expertise value can be used to reflect the depth of the user's knowledge about the topic, the trust value reflects one user's belief about another user's reliability. The initial calculation of trust is done assuming trust to be a measure of the similarity between people. Hence, the trust between users is calculated based on the similarity of their expertise vectors as detailed in Section 2.1.3.

3.1.8 Profile Builder

The profile builder gathers the generated trust and expertise models, as well as the user's Friend List, and serializes that information for storage. This profile is later made use of by the agents to assist in performing searches, providing referrals and replies.

3.2 Context Analyzer Service

This module takes in a sentence or a string of words as input and returns a set of keywords denoting the *context* of the string, i.e., it strips out prepositions, conjunctions and verbs from the sample text and returns the nouns. The context analyzer service is made use of by both the bootstrapper module and the agent module. The bootstrapper module uses this service to initially derive the expertise vector for each user based on their mail exchanges. The agent uses it to resolve a user query into a query vector which can then be compared with the expertise vector.

We make use of Y!Q, an external Web service, for our purposes [Search].

3.3 Data Store

The Data Store forms a repository for storing the Friend List and the trust and expertise models as illustrated in Figure 3.2. The information stored is local to each agent. The Data Store is created by the Bootstrapper module and is made use of by the agent to assist in making decisions. The agent also updates the Data Store based on knowledge gained.

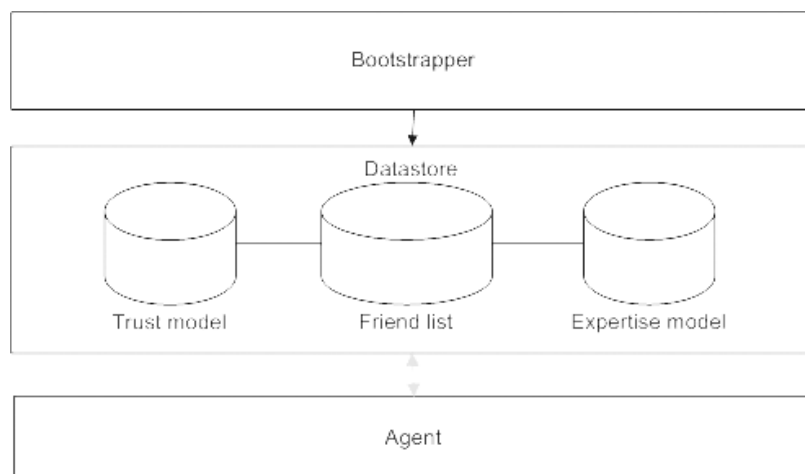


Figure 3.2: The Data Store

The Data Store is stored as a *FOAF* (Friend-Of-A-Friend) document. FOAF is a definition of an RDF vocabulary developed for expressing metadata about people, their interests, relationships and activities [specification]. The trust and expertise models are represented as FOAF resources.

3.4 Agent

An agent performs searches, elicits replies, and provides referrals on behalf of the user it represents. It makes decisions based on data obtained from the datastore, and can collaborate with other agents in the system. The agent may also modify the datastore based on the outcome of its decisions. Figure 3.3 illustrates the integral components of an individual agent in our system.

3.4.1 Query Parser

A query is any question, framed in natural language, provided by the user. For a query to be compared with a user's expertise vector, the keywords present in the query have to be extracted and the query statement has to be transformed into a vector representation. The keywords are extracted by sending the query statement to the context

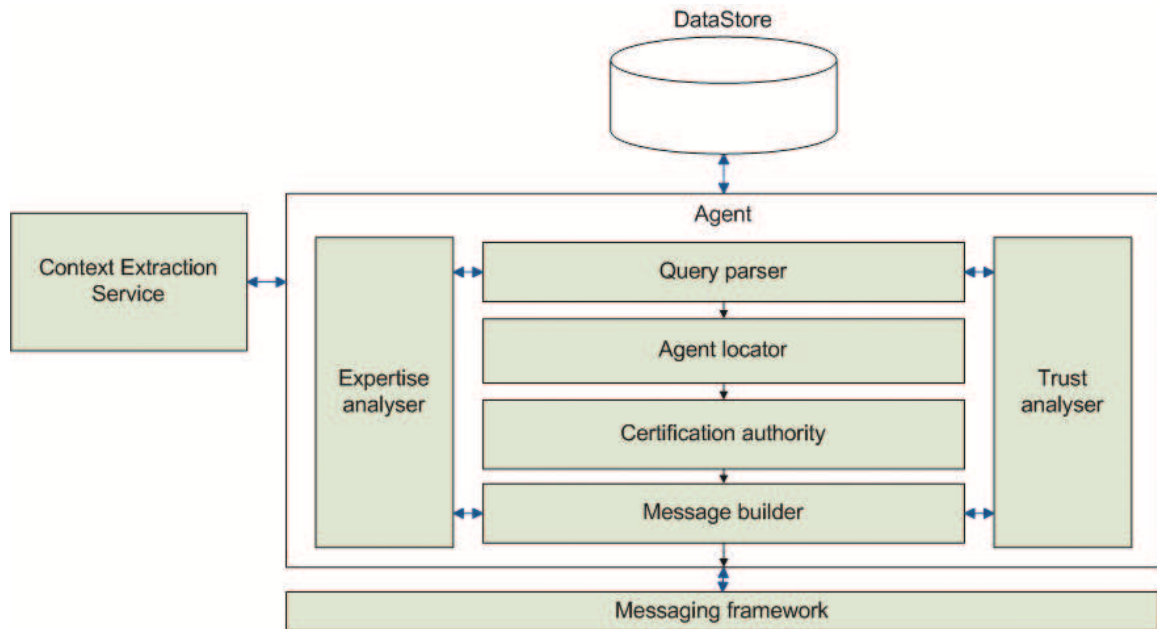


Figure 3.3: The Agent module

extraction service, and stripped to their stems using Porter’s algorithm as explained in section 2.1.2. These keywords are then enclosed in a message, along with the natural language query statement, before transmitting it to other agents in the system in the expectation of a reply.

When an agent receives a query, it initially compares the keywords with its own expertise vector. If a match is found, it presents the actual query statement (in natural language) to the user who might either reply or ignore it.

3.4.2 Expertise Analyzer

When an agent receives a query that it cannot answer based on its own expertise, it needs to provide a reference to a select few other agents on its Friend List. This selection is done by checking the expertise of every other agent on its list, based on information obtained from the datastore, and analyzing it for similarities to the query vector. This function is accomplished by the agent’s expertise analyzer component.

This also updates the expertise model present in the datastore based on response to queries to more accurately reflect the expertise of the users in its Friend List.

3.4.3 Trust Analyzer

When no friends are found whose expertise vectors match the query vector, a Weak Tie Search is performed. To perform this search, this component obtains the trust models from the datasource to assist in selecting the agents to forward the query to. This component is also responsible for calculating the implied trust value as outlined in section 2.3. It updates the trust model in the datasource to accurately reflect on the user's current relationships.

3.4.4 Agent Locator

Each agent in a multiagent system has a unique ID, called its Jabber ID. All messages sent through the system are directed at the Jabber IDs of the target user. The agent locator uses the Jabber IDs to keep track of other agents who are currently online, i.e., the agents who would be available for responding to any messages. This is accomplished by periodically querying our instance of a Jabber server and obtaining presence information. Hence, the agent locator is responsible for observing the current status of the user's friends in the network, whether online or offline.

3.4.5 Certification Authority

For any two directly connected users in the network, each must possess the other's public key. Each message from a source to a recipient would be encrypted using the public key of the recipient; the message would be unreadable to any other user who did not have access to the recipient's private key.

The certification authority is responsible for encoding the message with the public key and also decoding incoming messages to verify their authenticity. While encoding and decoding messages may be done independently by each agent, our system assumes use of a central certification agent to initially assign the public keys to each agent.

3.4.6 Message Builder

Each agent has a jabber id to which all messages are sent. A message is a XML document which comprises of various 'headers' in addition to the actual message content.

The headers contain information about the sender, the priority of the message and information about the referrer as can be seen from the sample messages provided.

This component is responsible generating the XML message and setting the values in the headers and also for parsing incoming messages to extract relevant information.

3.5 Messaging Framework

The framework comprises the underlying protocols required for inter-agent communication. It provides *instant messaging* and *presence* information. Instant messaging is used for queries, replies and referrals. Presence is used to keep track of the availability of each agent in the network. This layer is also responsible for maintaining sessions and routing messages.

Our framework makes extensive use of the *Jabber* protocol. The term Jabber is widely used to refer to a set of open protocols for streaming XML elements between any two points on a network, and to the technologies built using those protocols. Jabber is commonly used in conjunction with the *XMPP* protocol [RFC-3921]. The Extensible Messaging and Presence Protocol (XMPP) is an open XML technology for real-time communication, which powers a wide range of applications including instant messaging, presence, media negotiation, white boarding, collaboration, lightweight middleware, content syndication, and generalized XML routing.

Chapter 4

Discussion

The motivation behind our approach towards solving the issues of trust and expertise identification in social networks is not just to define heuristics but to develop a practical solution which can be used by an average user. Hence, it would make sense to evaluate our system based on the User Interface and adherence to HCI guidelines besides the accuracy of the results.

4.1 GUI walkthrough

In this section we provide a walkthrough over the process of bootstrapping, making queries and viewing replies.

We implemented our system as an extension for the Mozilla Firefox Web browser. An extension is an installable enhancement that enables developers to provide additional functionality to the browser. Our extension, when installed, adds an additional toolbar to the browser as shown in figure 4.1.

4.1.1 Bootstrapping

The system initially has to build a profile for the user before it can make use of the profile information for eliciting requests and replies. This is accomplished by the

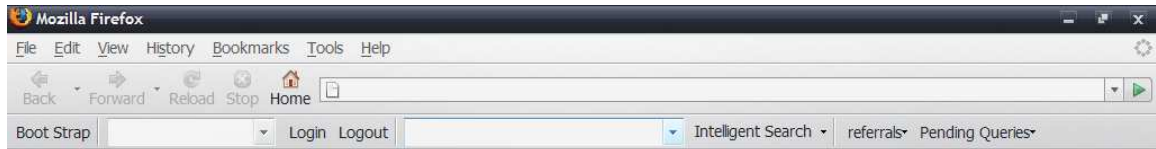


Figure 4.1: A browser with our toolbar installed

bootstrapper module whose internal architecture is described in section 4.1. Figure 4.1.1 depicts the initial “Welcome Screen” of the bootstrapping module.

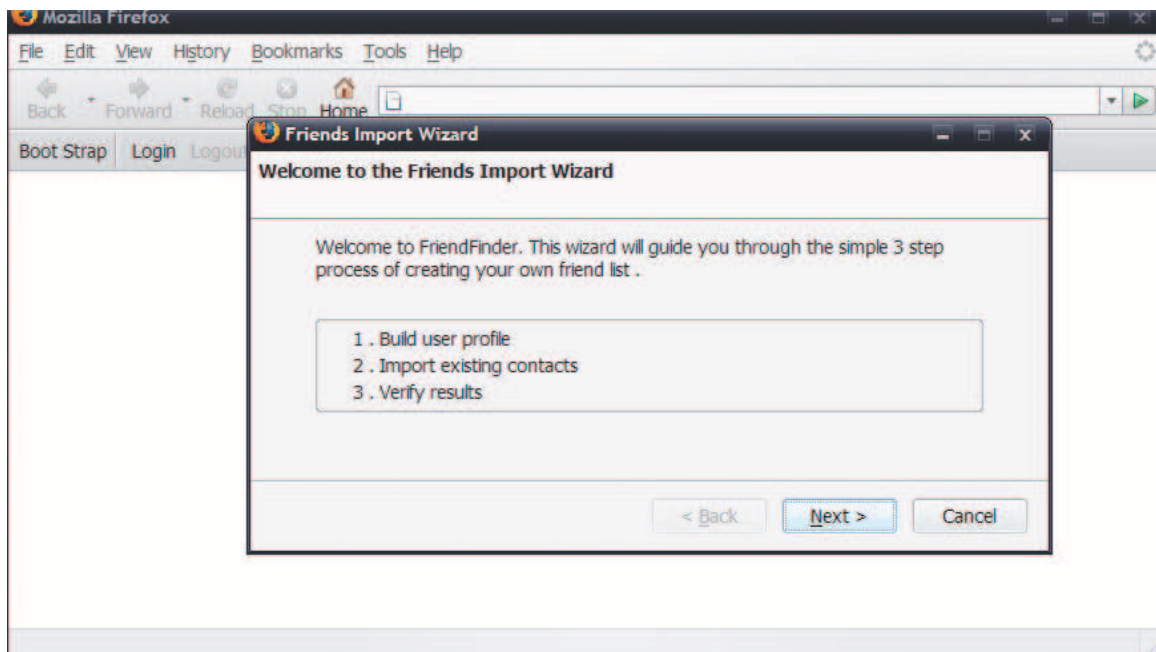


Figure 4.2: The bootstrapping interface

The user is provided with a list of pre-defined data sources from which his information can be extracted. Our system also supports adding a user-defined data source, provided the source allows third-party access of user data and that the data is in a format which can be understood by our system. Our system supports adding Gmail and Facebook as data sources by default, as shown in Figure 4.1.1.

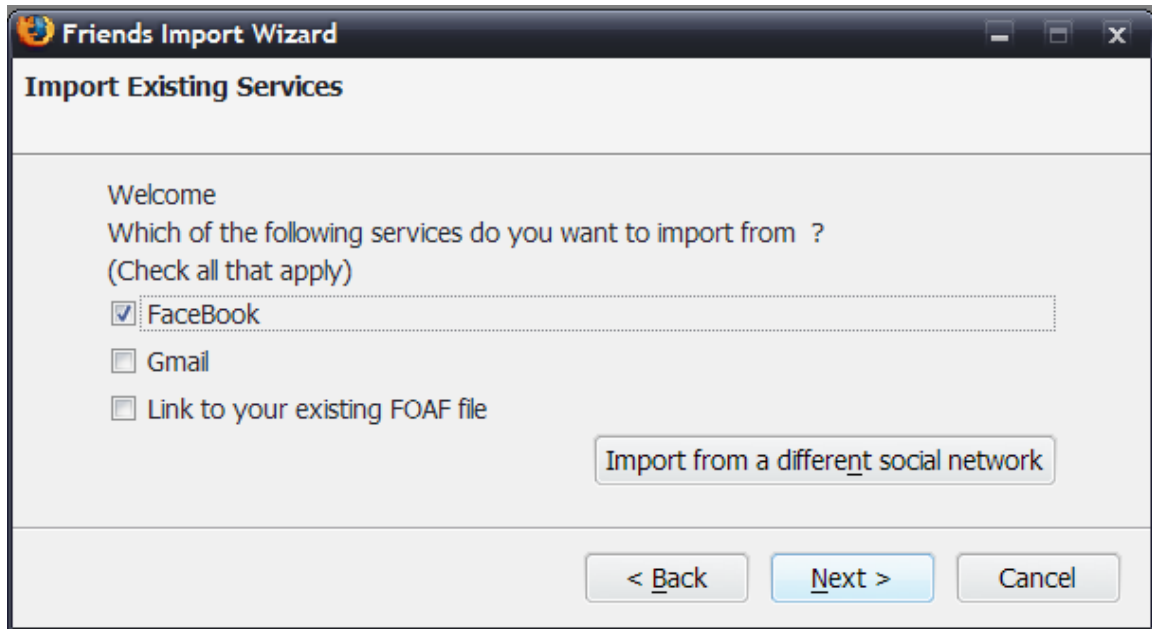


Figure 4.3: The import wizard

4.1.2 Performing searches

Once the profile is created and the user is logged in, he can then perform searches by typing the query in the search bar and pressing the search button. This is illustrated in Figure 4.1.2 which depicts the user performing a search about iPods. Pressing the “Intelligent Search” button would trigger either the Similarity Search algorithm or the Weak Tie Search algorithm.

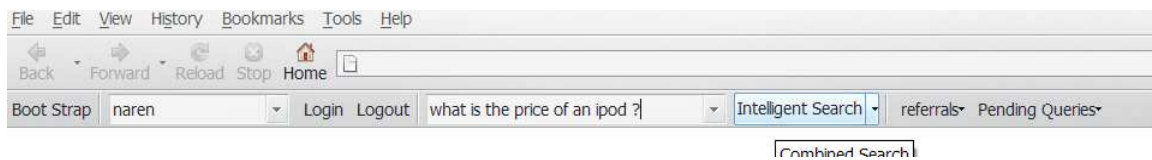


Figure 4.4: A search query

4.1.3 Issuing replies

If one of the user's friends happens to know about iPods, as determined by his agent, the query is presented to the user awaiting his response. Figure 4.1.3 shows the user's "Pending Queries" menu populated with the question about iPods. A menu entry contains the keywords associated with the query, and the Jabber ID of the query initiator.



Figure 4.5: The "Pending Queries" menu

Clicking on an item in the menu brings up a pop-up box displaying the actual question and a text field for the user to type in his answer. Once the query has been answered, it is removed from the "Pending Queries" menu and the reply is sent back to the initiator. Figure 4.1.3 depicts the user replying to the question about iPods.



Figure 4.6: Answering a question

The results are then propagated to the query initiator. Figure 4.1.3 illustrates the results returned for the question about iPods. For each search query, the results are displayed in the sidebar of the browser. The user can then choose to reward or penalize the other agent by choosing the "Accept Answers" and "Mark as Junk" buttons respectively, depending on his satisfaction with the answer.

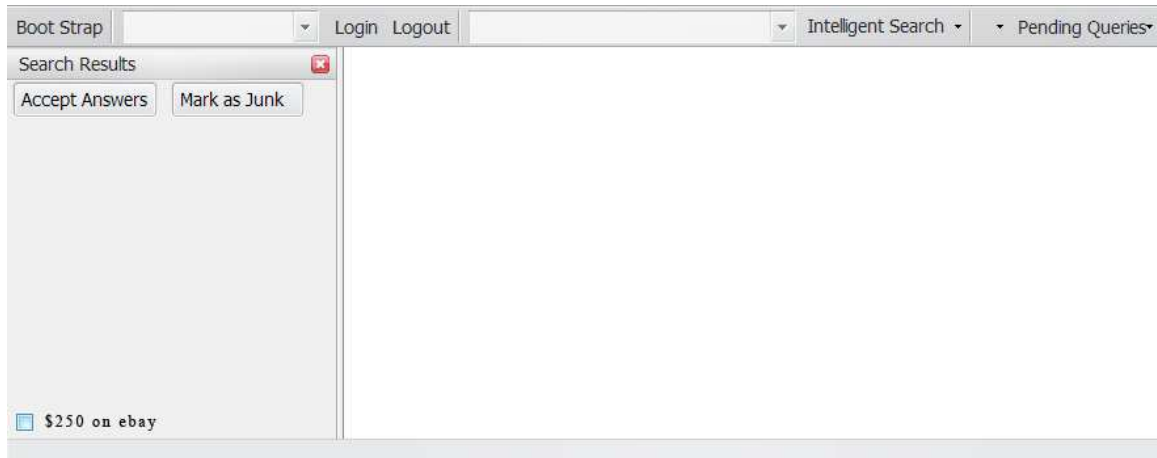


Figure 4.7: Replies for the query

4.1.4 Issuing referrals

If an agent does not find a close match between the query and the user's expertise, it issues a referral to one of the user's other friends. This referral is propagated back to the initiator. Figure 4.1.4 shows the referrals menu populated with the referrals obtained. A menu entry displays the Jabber ID of the provider of the referral and that of the target along with keywords present in the query.

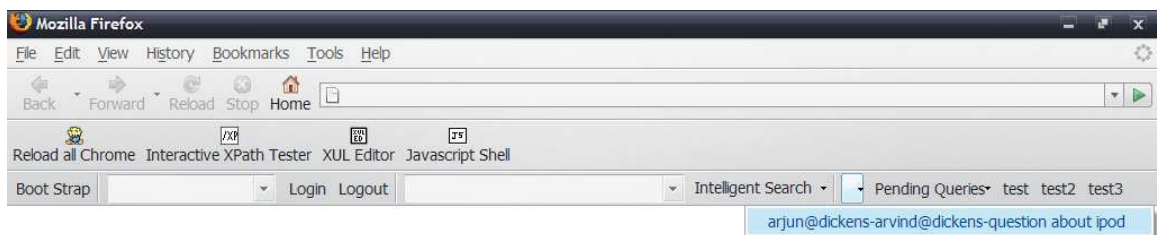


Figure 4.8: The referrals menu

The initiator can then choose to either follow-up the referral by selecting the corresponding menu item or discarding the referral.

Chapter 5

Conclusion

This thesis proposes a practical solution to the problem of finding experts in electronic communities. It has been shown that people prefer solutions given by friends to those provided by an arbitrary recommender system [Swearingen and Sinha, 2001]. Hence, we describe creation of a system which models the electronic community as a social network, based on information gathered from mining the user's email messages and importing information from existing social networking Web sites. We define algorithms for modeling trust and determining expertise based on information gathered from these data sources.

We define a framework which could be used to create agents that make use of the trust and expertise models to form a multiagent referral system. We recognize that for a referral system speed and computational cost are not the only concerns; ease of use and social concerns are also significant factors to be considered. Hence, the heuristics we define focus on practicality, ease of use, and social characteristics involved rather than any graph theoretical concerns.

We suggest a set of protocols which could be used to govern the decisions made by agents regarding making requests and providing replies. We show how following these protocols could lead the agent to identify an ideal referral chain from a user to an expert. We discuss the implementation details of such a system and show it in action.

5.1 Limitations

The limitations with our system are primarily because of tradeoffs made between developing a practical solution as opposed a theoretical solution which may not be suitable for implementation.

The performance of our system is largely dependant on the quality of information available at the external data source. If, for example, the user was in the habit of deleting his emails after reading them, we would not be able to infer information about his social network or build trust models.

Each agent in our system has information only about the user's friends and no knowledge of the underlying social network. This becomes an issue when assigning rewards and penalties to other agents for service provided, as none of the other users in the referral chain are likely to be in the user's Friend List and would hence be impervious to reward or penalties provided by the user.

Our system also assumes a central certification authority which grants public keys to each agent in the system. Having a centralized certification authority would adversely affect the loosely couple nature of our system, and the complexity involved in creating a decentralized public key distribution system would be prohibitive.

5.2 Future work

The system developed has yet to be tested with any significant number of users. We propose to do this in the next phase of our project. We have provided simple heuristics for propagating rewards and for setting priority values of the messages. These heuristics would have to be refined and tested with a larger sample size of users. Currently, bootstrapping is assumed to be a one-time process which builds a user profile based on his existing social network. However, the user's online network could evolve over time, we need to define a mechanism to detect these changes and suitably modify the models created by our system.

Bibliography

L. A. Adamic, R. M. Lukose, A. R. Puniyani, and B. A. Huberman. Search in power-law networks. *Physical Review E*, 64:046135, 2001.

Lada A. Adamic and Eytan Adar. How to search a social network, 2003.

Michael W. Berry, Susan T. Dumais, and Gavin W. O'Brien. Using linear algebra for intelligent information retrieval. *SIAM Rev.*, 37(4):573–595, 1995. ISSN 0036-1445. doi: <http://dx.doi.org/10.1137/1037127>.

Peter M. Buston and Stephen T. Emlen. Cognitive processes underlying human mate choice: The relationship between self-perception and mate preference in Western society. *PNAS*, 100(15):8805–8810, 2003. doi: 10.1073/pnas.1533220100.

K. Decker, K. Sycara, and M. Williamson. Middle-agents for the Internet. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, Nagoya, Japan, 1997.

Peter S. Dodds, Roby Muhamad, and Duncan J. Watts. An experimental study of search in global social networks. *Science*, 301(5634):827–829, August 2003.

Facebook. <http://www.facebook.com>.

Leonard N. Foner. Yenta: a multi-agent, referral-based matchmaking system. In *AGENTS '97: Proceedings of the first international conference on Autonomous agents*, pages 301–307. ACM Press, 1997. ISBN 0897918770. doi: 10.1145/267658.267732.

Yair Frankel and Moti Yung. Distributed public key cryptosystems. In *PKC '98: Proceedings of the First International Workshop on Practice and Theory in Public Key Cryptography*, pages 1–13, London, UK, 1998. Springer-Verlag. ISBN 3-540-64693-0.

Yolanda Gil and Varun Ratnakar. Trusting information sources one citizen at a time. In *ISWC '02: Proceedings of the First International Semantic Web Conference on The Semantic Web*, pages 162–176, London, UK, 2002. Springer-Verlag. ISBN 3-540-43760-6.

Gmail. <http://www.gmail.com>.

J. Golbeck and J. Hendler. Reputation network analysis for email filtering, 2004.

Jennifer Golbeck and James Hendler. Inferring binary trust relationships in Web-based social networks. *ACM Trans. Inter. Tech.*, 6(4):497–529, 2006. ISSN 1533-5399. doi: <http://doi.acm.org/10.1145/1183463.1183470>.

Mark S. Granovetter. The strength of weak ties. *The American Journal of Sociology*, 78(6):1360–1380, may 1973. ISSN 0002-9602.

Thomas Hofmann. Unsupervised learning by probabilistic latent semantic analysis. *Mach. Learn.*, 42(1-2):177–196, 2001. ISSN 0885-6125.

John Horrigan Jeffrey Boase, Lee Rainie and Barry Wellman. The strength of Internet ties. Technical report, Pew Internet and American life project, 2006.

Sepandar D. Kamvar, Mario T. Schlosser, and Hector Garcia-Molina. The eigentrust algorithm for reputation management in P2P networks.

Henry Kautz, Bart Selman, and Mehul Shah. Referral web: combining social networks and collaborative filtering. *Commun. ACM*, 40(3):63–65, March 1997. ISSN 0001-0782. doi: [10.1145/245108.245123](http://doi.acm.org/10.1145/245108.245123).

Yahoo! Mail. <http://www.yahoomail.com>.

Stephen P. Marsh. *Formalising Trust as a Computational Concept*. PhD thesis, University of Stirling, Apr 1994.

R. Matthew, R. Agrawal, and P. Domingos. Trust management for the semantic Web, 2003.

Orkut. <http://www.orkut.com>.

Sztompka P. *Trust : A sociological theory*. Cambridge University Press, 1999.

M. F. Porter. An algorithm for suffix stripping. pages 313–316, 1997.

- RFC-3921. <http://www.xmpp.org/rfcs/rfc3921.html>.
- Gerard Salton and Chris Buckley. Term weighting approaches in automatic text retrieval. Technical report, Ithaca, NY, USA, 1987.
- J. Ben Schafer, Joseph A. Konstan, and John Riedi. Recommender systems in e-commerce. In *ACM Conference on Electronic Commerce*, pages 158–166, 1999.
- Michael F. Schwartz and David C. M. Wood. Discovering shared interests using graph analysis. *Commun. ACM*, 36(8):78–89, 1993. ISSN 0001-0782. doi: <http://doi.acm.org/10.1145/163381.163402>.
- Y!Q Search. <http://yq.search.yahoo.com>.
- FOAF Vocabulary specification. <http://xmlns.com/foaf/0.1/>.
- K. Swearingen and R. Sinha. Beyond algorithms: An hci perspective on recommender systems, 2001.
- Jeffrey Travers and Stanley Milgram. An experimental study of the Small World problem. *Sociometry*, 32(4):425–443, 1969.
- Stanley Wasserman, Katherine Faust, and Dawn Iacobucci. *Social Network Analysis : Methods and Applications (Structural Analysis in the Social Sciences)*. Cambridge University Press, November 1994. ISBN 0521387078.
- Bin Yu and Munindar P. Singh. Searching social networks. In *AAMAS '03: Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 65–72, New York, NY, USA, 2003. ACM Press. ISBN 1-58113-683-8. doi: <http://doi.acm.org/10.1145/860575.860587>.
- Jun Zhang and Mark S. Ackerman. Searching for expertise in social networks: a simulation of potential strategies. In *GROUP '05: Proceedings of the 2005 international ACM SIG-GROUP conference on Supporting group work*, pages 71–80, New York, NY, USA, 2005. ACM Press. ISBN 1-59593-223-2. doi: <http://doi.acm.org/10.1145/1099203.1099214>.
- Cai-Nicolas Ziegler and Georg Lausen. Spreading activation models for Trust propagation. In *Proceedings of the IEEE International Conference on e-Technology, e-Commerce, and e-Service*, Taipei, Taiwan, March 2004. IEEE Computer Society Press.