Abstract

CHATTERJEE, SUBHAYU SAROJ. Instant Messaging Interface and Transport for the MultiAgent Referral System (Under the direction of Munindar P. Singh)

Agent-based systems have been around for quite some time now. They have been extensively used in communication systems involving human interactions. The MultiAgent Referral System (MARS) helps automate the process of expertise location using referral chains. Previously, this system was implemented using email as the transport mechanism for the various referrals and queries generated by the agents. The asynchronous nature of email would prove restrictive in real-life scenario. This thesis develops an infrastructure using an Instant Messaging (IM) system that provides an user interface and transport mechanism for MARS. MARS has a distributed architecture and associates each user with an agent. This system is slightly different from a traditional IM system, which involves a client and a server, whereas in this case the messages from a user are routed through his agent to the server. Our specific approach exploits the open-source Jabber IM system, which enables us to integrate IM with MARS. In this manner, agent-to-agent communication is realized through IM and an IM-based user interface is provided to the users.

INSTANT MESSAGING INTERFACE AND TRANSPORT FOR THE MULTIAGENT REFERRAL SYSTEM

BY

SUBHAYU CHATTERJEE

A THESIS SUBMITTED TO THE GRADUATE FACULTY OF

NORTH CAROLINA STATE UNIVERSITY

IN PARTIAL FULFILLMENT OF THE

REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

RALEIGH

JANUARY 2003

APPROVED BY:

ROBERT ST. AMANT

PENG NING

MUNINDAR P. SINGH

CHAIR OF ADVISORY COMMITTEE

Biography

Subhayu Chatterjee was born on December 12th, 1977 in Mumbai, India. He lived in Mumbai till August, 2000. He got his Bachelor of Engineering degree in Computer Engineering from University of Mumbai in July, 1999. He worked for the next couple of months with NetAcross Online Solutions in Mumbai before deciding to get back to school for his MS. He was a Master's student in North Carolina State University in the Department of Computer Science from August, 2000 to November 2002. During his Masters, he also worked as a co-op with IBM since May, 2001.

Acknowledgement

I would like to express my gratitude and thank my advisor, Dr. Munindar P. Singh for showing me the importance of focused research and offering me valuable guidance. I would also like to thank Dr. Robert St.Amant and Dr. Peng Ning for serving on my thesis committee.

I would also like to thank Pinar, Ashok, Raghu, Amit, Paul, Mike and Zhengang for all the help and for those interesting discussions on various topics. It was a great experience to be a part of the DB Lab in school and work with these folks. Also, a special thanks to Dr. Bin Yu, who gave me some good advice on my thesis drafts and presentation.

I would like to thank Rohit, Rahul and Ravi for being such great and encouraging room-mates. I am grateful to Sarat, who always made me realise the value of my thesis. I would like to sincerely thank IBM and especially my manager, Mark Gould, for encouraging me and helping me out whenever I was in need of anything.

Finally, I wouldn't have been here without my family. I am greatly indebted to my parents who put in so much to get me here today. I am thankful to Sanjoy and Jaba-di for always being there for me and helping me out whenever things got rough. Also I am grateful to Minati, who has always been a great motivating friend to me.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Communication underlies how we relate with one another, how we exchange knowledge, and acquire information from others. There are many forms of communication involving humans as well as automated systems. The purpose of communicating with each other may be varied across different domains of life. The communications environment is changing its face towards an open market of information services where the vision is “information any time, at any place, in any form” [Magedanz et al., 1996]. The various types of communication may span from telephonic conversations to voice mail, from broadcasting of messages to face-to-face discussions, from letters to email to chat and instant messaging. The focus of this thesis is agent communication via Instant Messaging. This is demonstrated by using the existing MultiAgent Referral system (MARS). As today’s most complex computing environment, the Internet confronts researchers and application developers with entirely new challenges and computing paradigms. Agent technology has attracted a lot of interest and hopes for shaping up the future information society. In this modern age, electronic communication, which includes email and Instant Messaging, has become a pervasive part of business needs and home landscape.

1.1 Agent Communication

The Internet is broadly accepted as the technology of today, agents are expected to be the paradigm of tomorrow [Omicini et al., 2001]. Agent technology is an important new way to create complex software systems [Dignum and Greaves, 2000]. Agent programs are designed to autonomously collaborate with each other in order to satisfy both their internal goals and the shared external demands generated by virtue of their participation in agent societies. Inter-agent communication is usually defined by the use of an agent communication language (ACL). The ACLs exist in a logical layer above the transport layer. The transport layer is implemented using various protocols such as TCP/IP, HTTP or IIOP and is responsible for the communication at the level of data and message formats whereas ACLs manage the communication on the intent and social levels. The most widely implemented and used ACLs in the agent community are

- Knowledge Query and Manipulation Language (KQML) [Finin et al., 1994] and
- Foundation for Intelligent Physical Agents (FIPA) ACL.

ACLs are complex structures composed of different sublanguages that specify message content, ontology, and propositions. They are tailored to support the various kinds of collaboration, negotiation and information transfer required in multiagent interaction.

1.2 Social Networks

Social networks are a natural way to go about seeking information [Gladwell, 1999]. For our purpose, a social network is considered to be a group of people linked by some common professional activity or some affinity between them. Numerous studies have shown that

one of the most effective channels of dissemination of information and expertise within an organization is its informal network of collaborators, colleagues and friends [Wasserman and Faust, 1994]. Almost all real-world activities at some point involve interaction with others or are influenced by the presence and opinions of others in a social setting [Dourish, 1999]. Research has proved that social networks are as least as important as other indexing methodologies for certain kinds of problem solving. This is because locating experts in a community is critical for many problems and social networks can help in doing so. The MARS prototype uses this concept of social network to build a community and searches the social network for an expert on a topic through a chain of referrals from the requester to the expert.

1.3 MultiAgent Referral System

MARS was built to provide users a way of sharing knowledge among themselves and learn from each other. The main activities carried out by this system are knowledge management and expertise location. MARS builds neighbor models for the neighbors of a user. Intuitively, a user may list his friends as neighbors to be first contacted for various queries. The MARS architecture is entirely distributed and preserves the autonomy of its users [Mo, 2000].

In MARS, each user is assigned a software agent and this agent helps automate the process of expertise location by a series of referral chains. The present MARS architecture is implemented using an email system, which acts as the transport for data and messages between multiple clients who are each connected through a software agent. Using email as transport makes the system asynchronous and user dependent. The end user has to check his email account for emails with queries and answers, which could mean that the user

may check his mail after days without answering the query. In this scenario, MARS, which builds on a social network, may prove to be ineffective since the mode of communication is not instantaneous or synchronous.

1.4 Instant Messaging (IM) Systems

IM has become one of the most widely used consumer and business applications due to its ability to tie people together through dynamic and instant interaction. IM has proved to be an effective tool for informal communication which supports tasks for quick questions, scheduling meetings, and keeping in touch with friends and family. It has been found that instant messaging is most effective for short queries or questions. IM has ancient roots in Unix utilities, but it has found a wide audience in last couple of years via the commercial instant messaging products like AOL Messenger, MSN messenger, and Yahoo Messenger [Nardi et al., 2000].

1.5 Motivation

The motivation behind this work is to make MARS work with an IM-based protocol. Since the main goal of MARS is to locate expertise for a particular domain within a social community, it needs to interact with end users. Using IM highlights certain aspects of communication which are not a part of other types of communication. Recent empirical work has shown the importance of informal communication using IM for effective collaboration [Nardi et al., 2000]. The interactions or communications between the users support the prime objective of MARS i.e. joint problem solving, coordination, social bonding and social learning, all of which are essential for a complex collaboration. In this thesis we have

utilized a technology which is relatively new to this environment - IM, for effectively supporting referrals and informal communication. MARS models social networking, which depends on interactions between users that can be further enhanced and made instant by using instant messaging.

1.6 Thesis Organization

This thesis studies the process of developing an instant messaging framework which integrates with the existing MARS in order to build a social network and exchange information within the social network. The remainder of the thesis is organized as follows: Chapter 2 presents an overview and architecture design about MARS. Chapter 3 presents background information on IM systems and Jabber in particular. Chapter 4 deals with the proposed architecture design, assumptions, challenges and decisions. Chapter 5 explains the user's view and implementation details. Finally Chapter 6 offers some further research areas and concludes the thesis.

Chapter 2

Overview of Multiagent Referral System

MARS is a prototype implementation of a multiagent system that evolves by building and exploiting existing social networks. MARS helps build a circle of trust based on referrals from a trusted friend or a set of friends rather than on anonymous references. This system enables an entity involved in a particular social network to search for some information based on the expertise and sociability of the other entities involved in the same network.

According to [Yu et al., 2001]

- *Expertise is defined as the ability to produce correct domain answers* and
- *Sociability is defined as the ability to produce accurate referrals.*

The motive behind this approach is to make searching for specific information in a certain domain easier and more relevant. When one searches for certain information on the web, he often gets irrelevant information with no authority about the reliablility of the results.

A social network consists of entities with similar activities and having common areas of interests. Numerous studies have showed that one of the most effective channels of dissemination of information and expertise within an organization is its informal network of collaborators, colleagues and friends [Granovetter, 1973, Wasserman and Faust, 1994,

Kautz et al., 1997b]. MARS has been designed to support interactions between the various users that are representated by agents. Each user in the community is portrayed as an agent and information is searched within the social network with the assistance of these agents. The final goal here is to retrieve information from the social network using the basic knowledge management principles.

2.1 Agents

Agents are computer programs which attempt to perform some set of tasks autonomously for their users in a trustworthy and personalized fashion [Foner, 1997]. These programs can be programmed manually or can be automated to learn and adapt from situations or tasks that they are assigned to do based on the responses. In addition, an agent can perform more than one task. Agents have been around for quite some time but they are getting increasingly popular with more and more web based applications. In today's world we see a lot of agent based implementations for search engines and travel portals. Intelligent agents are software entities that carry out some set of operations on behalf of a user or another program with some degree of independence or autonomy, and in doing so, employ some knowledge or representation of the user's goals or desires.

2.1.1 Multiagent Systems

Multiagent systems are a network of agents which work in a collaborative fashion to achieve certain goals or complete some tasks. These agents may have been implemented with an initiative to get some task done. There are many applications which need to share data and communicate with each other over a network. In such an environment, multiagents are certainly desired to get some joint tasks done without much information about the other

applications. Multiagent systems have several advantages, though one of the largest problems with them is *how agents are supposed to find each other* [Foner, 1997]. The MARS implementation has a software agent that can automate the process of query-answer-referral and help the user learn from the experience and knowledge of others [Yu et al., 2001]. A multiagent system consists of individual agents which have the following properties

- Autonomy: Agents perform without the intervention of humans.
- Social Ability: Agents interact with other agents via some Agent Communication Language.
- Reactivity: Agents perceive their environment and respond based on changes in the environment.
- Proactiveness: Agents exhibit goal directed behavior by taking an intiative.

2.1.2 Communication between Agents

Communication plays an important role in multiagent learning systems. It may supply one agent with valuable information and thus avoid re-discovering the wheel. But setting a common language between various agents doesn't come easy as there are many problems involved in resolving language differences between agents. A lot of thought needs to be put into designing a system involving multiagents since the interfaces between agents have to be cleary defined to avoid problems related to interpretation of statements made by each agent.

2.2 Knowledge Management

Knowledge management can be defined as a systematic approach to create, capture, organize, access and use organizational knowledge and learning. Knowledge management can

be applied to a wide range of tasks involving information that needs to be shared within a social network. It finally leads to better decisions, extensive learning, better planning and more flexibility in terms of the range of information evolved from a search request.

MARS defines a set of domains based on the interests of users involved in the social network. Each entity in the social network has some kind of expertise in some domain. The objective here is to direct a request from a particular user of the social network to a person who has the expertise in that domain. Knowledge here is defined in terms of the technical know-how and objective answers to some queries provided by the other users involved in the social network. The information gathered gets evaluated by the end user to determine the relevance of the answer to the query. This way the users of the social network explicitly evaluate a user's expertise in a particular domain. In this prototype model of MARS, knowledge management approach pertains to every user since one has to

- Retrieve information from the social network
- Locate expertise in the social network
- Evaluate the results of the queries
- Update neighbor model based on results.

2.2.1 Information Retrieval

Searching the web for certain documents or related information can be frustating and unyielding. There is no authority to certify the validity of the results that are fetched from the searches. To evaluate results from multiple sources, there is no measure of relevance. Searching for information within a social network involving users with some expertise proves to be a better approach. MARS expects every user involved in the system to have some expertise in a particular domain.

2.2.2 Virtual Communities

Virtual Communities is a term used to refer to the various agents that are working collaboratively. Based on common tasks performed by these agents, they form a team or group to perform certain actions. The agents interact with each other to form a virtual community or a society.

In MARS, every user is representated by a software agent which maintains a local profile of the user it is representating, builds his neighbor models and handles interactions with the other users agents. All decisions made by the software agent are based on the user's profile and their neighbor model, which is initially bootstrapped with some neighbor id's and then updated through the agent learning process. The idea here is to make use of the hidden resources and explore virtual communities to gain knowledge in a particular domain.

2.3 Referral System

A Referral system is one that refers known entities in a particular social community based on some criteria. Referrals systems have always been compared with traditional recommender systems, which are based on anonymous recommendations. In real life, its not possible to make a big decision based on anonymous comments. This is where referral systems fill in the gap by extracting the benefits of a social community consisting of friends categorized by their knowledge. Many information gathering tasks are better handled by finding a referral to a human expert rather than by simply interacting with online information resources [Kautz et al., 1997a]. A personal referral in a social community helps users find information that may not have been made public. There are many forms of knowledge that cannot be searched on the web or that cannot be found using traditional

document search engines. Also a referral system helps in locating experts in particular domains within a social community, which in turn helps in finding information. The various advantages that a referral system has over a recommender system are as follows

- Referral Systems can be designed to provide trusted recommendations. The concept of a referral chain helps track an expert in a particular domain, in a social community.
- Referral systems complement traditional recommender systems. Information which is not disclosed in public can be extracted through such a system.
- Referral systems can access a larger community without fear of offense [Mo, 2000].

Referral Chains

The process of finding an expert who is both reliable and likely to respond to the user can be viewed as a search though the network of social relationships between individuals [Kautz et al., 1997a]. In MARS, an agent represents a user and also maintains its neighbor models and interactions with the other agents. When a query for some information is received, it checks for an expert in its neighbor models. If there is no expert, it checks for a user with a sociability value that is higher than the value set in the ABLE ruleset file defined in section 2.5.2. Once it finds some user, it forwards the query to that user. The other user, based on his expertise and neighbor models, may then decide to forward the query to some other user who it considers an expert in that domain. This way the query gets forwarded to many users which forms a chain of requests. This chain of requests can be termed as a referral chain. It is created to find some information in a social network and also determine the expert in that domain. If that expert user isn't present in the user's neighbor models, then he is added to the neighbors list.

The idea behind a referral chain is to locate expertise in a social community. Experts in a particular domain are not known to everyone. It is through these friends and neighbors that we reach the potential experts. Using referral systems to model social networks is an innovative approach.

2.4 MARS Architecture

This section briefly describes the existing MARS architecture. The MARS prototype impements a multiagent system that enables a user to search a social network for some information. The agent enables the process of querying and automates the referral process to get an answer from another user in the social network. The architecture of the system is a fully distributed system using email as the transport for messages. The main goal of the system is analogous to a general referral system's goal i.e. to produce accurate referrals so that better information retrieval is possible. Referral systems integrate on-line and person-to-person information access using software agents [Yu et al., 2001]. There are many implementations of Referral systems that are available e.g. ReferralWeb, ContactFinder etc. These systems do not provide accurate referrals and lack in other features [Mo, 2000]. MARS captures the right requirements and implements a system which can be applied to human structure in a social network. The architecture of the system is as shown in the following figure.

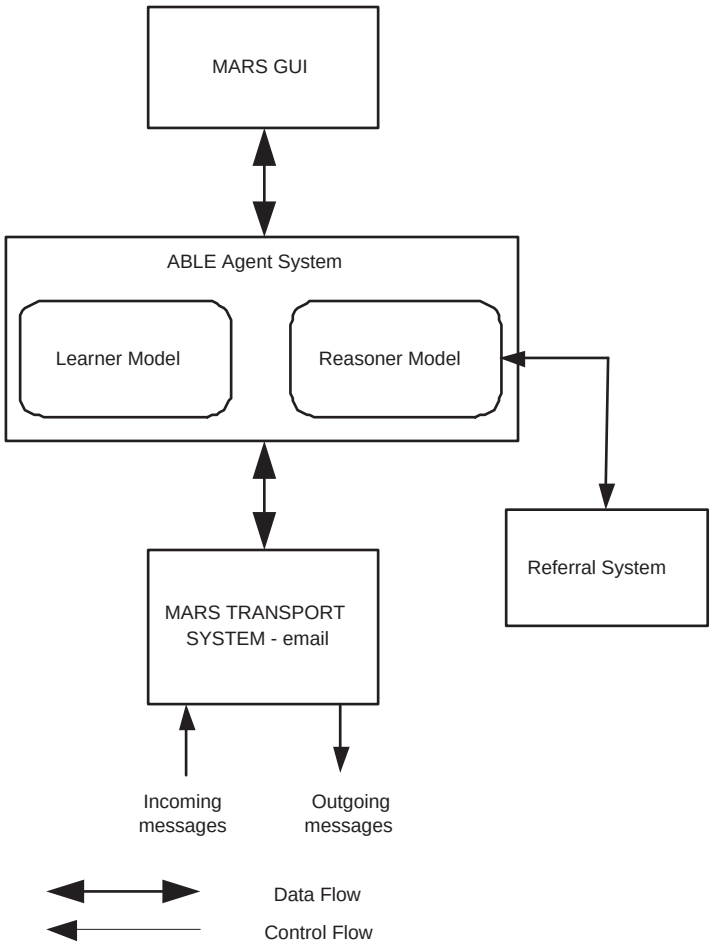

Figure 2.1: Original Architecture of MARS

The architecture is simple and is based on the fact that every agent represents a user in the social network. When a query reaches a user, his agent analyzes the query and decides if the query needs to be seen by the user. If the query is related to a domain in which the user doesn't have any expertise, it decides which users from its neighbor model need to be referred. The user who initiated the query then gets the referral list of user's whom it needs to forward the query to. The users on the referral list may send back a referral list or may return back an answer. An agent will show the query or answer the query only if

it is confident that it has knowledge in that domain to answer. The capability of a user to answer a query depends on his expertise in that domain and his confidence in referring a neighbor depends on the sociability or expertise level of that neighbor. The entire query-answer-referral flow as stated by [Yu et al., 2001] can be shown by 2.2

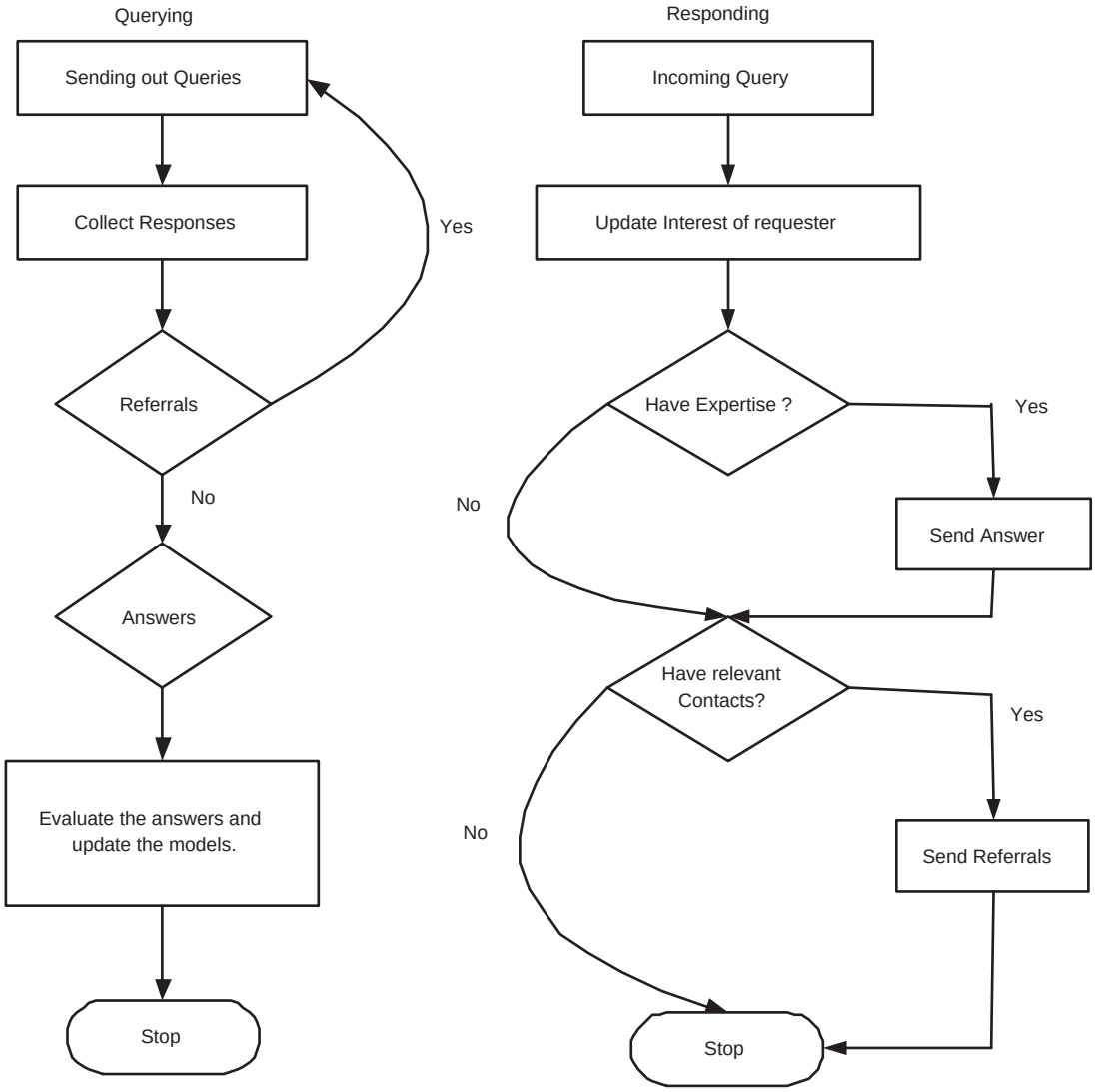

Figure 2.2: Querying and Responding flow diagrams

The MARS prototype has been implemented using Java. The email system which transports the messages between the users in a community has been implemented using JavaMail. The agent learning and reasoning is implemented using an agent building framework called ABLE (Agent Building and Learning Environment). The users of a social community need to register with the system in order to be a part of the system. The user decides to setup his expertise vector and sociability values initially. This process is called as bootstrapping, which essentially means that a user configures his initial expertise level based on which the neighbors will evaluate him. The system requires some prior knowledge of the users within the social community. This is due to the fact that when the system is initially deployed, there will be no relationship between the users and their neighbor models. This is contrary to a real world situation where a user is a part of a community since he knows some other users there or has been recommended by some user to join the community.

Based on the assumption that there exist users who have bootstrapped their expertise levels and initialized their sociabilities, we can start using the system. The neighbor models for the agents are initialized too. The initial queries are sent only to the immediate neighbors. If the neighbors cannot produce an answer for the query, they search their neighbor model to refer other users to the query initiator's agent. The expertise and sociability values are dynamically changed by an agent depending upon the quality of the answers generated by the neighbors. The agents dynamically update the neighbor models depending upon the referral and answers generated by its neighbors or other users in the social network. The table 2.1 explains the process of updating the expertise and sociability of a user.

The table sumarizes the approach to update the neighbor model based on responses. The agent updates the expertise and sociability of the agents in its neighbor models based on its success in obtaining a good answer or referral, leading to a good answer.

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 2.1: Example update scenario

2.5 MARS Components

MARS has several components that have been integrated together. The transport layer of the original MARS prototype has been implemented using Java Mail API. The agent learning and reasoning system has been built using the IBM ABLE toolkit. All these components work together to produce effective referrals and answers from the agents involved in the system. The MARS registration module and the GUI are the other components which have been implemented. The registration system takes care of setting up the initial expertise values of a user in certain domains.

2.5.1 MARS Mail Transport

MARS uses a emailing mechanism to transport messages between the various agents who are involved within the social community. Every user in the social community is registered with the email server and has an email account which is used for the MARS messages. The email transport system has been implemented using the Java Mail API. The Java Mail API, implemented as a Java platform standard extension provides a set of abstract classes that model a mail system [Mo, 2000]. The Java Mail API also includes the implementation of the core mail packages as well as implementations of the IMAP and SMTP service

providers. The MARS communication classes are implemented using the Javabeans Activation Framework (JAF) and POP3 provider along with the Java Mail API. The previously implemented version of MARS uses the email server on salevista.com with 30 email accounts as a testbed.

2.5.2 ABLE

The IBM Agent Building and Learning Environment (ABLE) is a toolkit for building intelligent agents that include both reasoning and learning [Mo, 2000]. ABLE beans are a set of function specific beans that are implemented to perform some actions. ABLE beans can be broadly categorized based on their functions into 3 sections.

- Data Beans
- Learning Beans
- Reasoning Beans

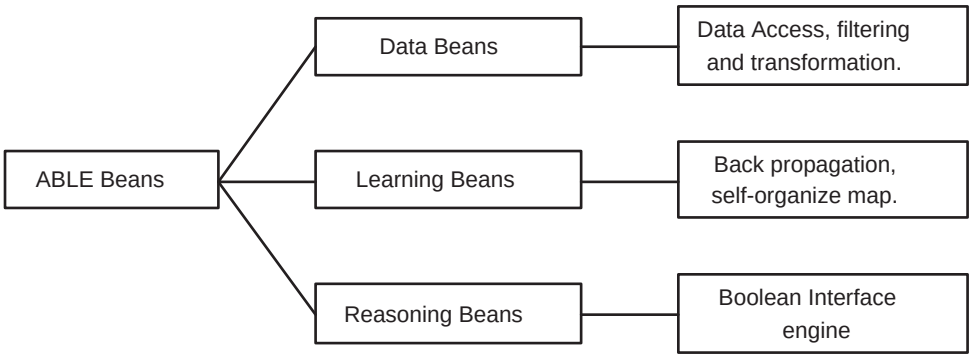

Figure 2.3: ABLE Beans structure

As shown in figure 2.3, the Data beans are responsible for handling the data, filtering and transforming the data to the required format. The Learning beans examine the data that has been provided to them and updates the parameters based on the changes that have occurred in the execution environment. The Reasoning beans handle the reasoning part

which is based on the rulesets that are defined prior to execution. The rulesets are defined as simple if-then rules which are then interpreted by the Reasoner to make decisions. ABLE has different inference engines which use the Agent Rule Language (ARL). This engine has been implemented in MARS to use a forward chaining algortihm to produce new facts.

2.5.3 Ontology

An ontology is defined as a specification of conceptualization [Gruber, 1993]. It can be regarded as a set of definitions of formal vocabulary. In MARS, the ontology is representated by a query domain. MARS analyzes the knowledge possessed by each user in the social community in terms of an expertise vector. The expertise maps the knowledge of the user to a particular domain. Based on these values, it determines the experts in a particular domain who are present in the social community. The original implementation stores all the information about an ontology in the file system.

2.5.4 Neighbor Model

A neighbor model is formed based on the neighbors present in a user's neighbors list. Neighbor model represents another user whom the original user is going to query based on his expertise and sociability values. When a user generates a query relevant to some ontology, his agent evaluates all the users in his neighbor models to figure out the users who have the required expertise in that domain or who have a high sociability value. The neighbor model needs to be updated by the users agent as and how it gets referrals and evaluates replies from other referred users. The neighbor models are not initially present. The neighbor models of a user have to be boostrapped initially to have some known users within the social community. Each neighbor model is representated by the following information

- DigitalID: An unique number assigned to each agent.
- Expertise: This value lies between 0 and 1 depending on the domain field. It is represented as a vector.
- Sociability: A scalar value representated by a number between 0 and 1 indicating a neighbors sociability level.

Chapter 3

Jabber Instant Messaging System

Jabber is an open source XML based protocol for instant messaging and presence detection. The characteristics that have propelled Jabber's use as an extensible messaging system make it an ideal choice for our implementation. This chapter provides a high-level overview of the architecture of Jabber, focusing especially on the design of the Jabber Instant messaging protocol and is based on discussions from various sources.

The idea of IM has been around for a long time, but recent advances have made it one of the most widely used communication tools. The innovation of recent IM systems is the packaging of the various IM systems into one managed messaging platform. Jabber takes this concept further and establishes an *universal messaging address* and the concept of *presence* that can be applied to the address for a simple communication. IM addresses are an extension of the well established email addresses that everyone has today. Figure 3.1 shows how we can have a universal id to manage different systems

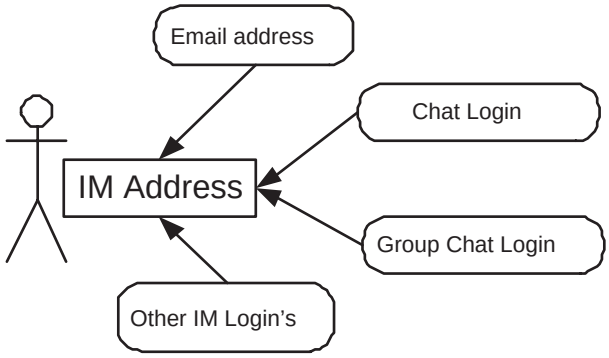

Figure 3.1: Universal Instant Messaging Address

3.1 Introduction to Jabber

The Jabber protocol and server architecture are built around the concept of switching XML document content between multiple locations. At its core, Jabber is an XML message switch supporting both external clients and internal or external services. Instant Messaging is the most popular service currently implemented on Jabber since it's a widely demanded and implemented application. Jabber has become popular and a widely adopted platform for IM applications because of its extensibility, or simply because of the ability to build on or around existing Jabber server installations.

The first application of Jabber technology, which is an IM and presence system, originated in and continues to be developed by the open-source community. The Jabber IM system is distinguished from existing IM services by several key features:

- XML foundation
- Distributed network
- Modular, extensible architecture
- Interoperability between proprietary IM protocols.
- Simple client implementation.

The adopters of Jabber IM have realized that Jabber delivers a platform for creating new clients and services, which go beyond the simple IM usages. In this thesis, we have used the Jabber protocol as the transport layer protocol for transferring messages from one agent to another. The main reason for choosing Jabber as a platform to deploy our application is that it provides us a central solution to converse with all the proprietary IM based software and it is also based on an open source XML based protocol which has been proposed as a RFC. The general architecture of the Jabber implementation on the Internet is shown in Figure 3.2

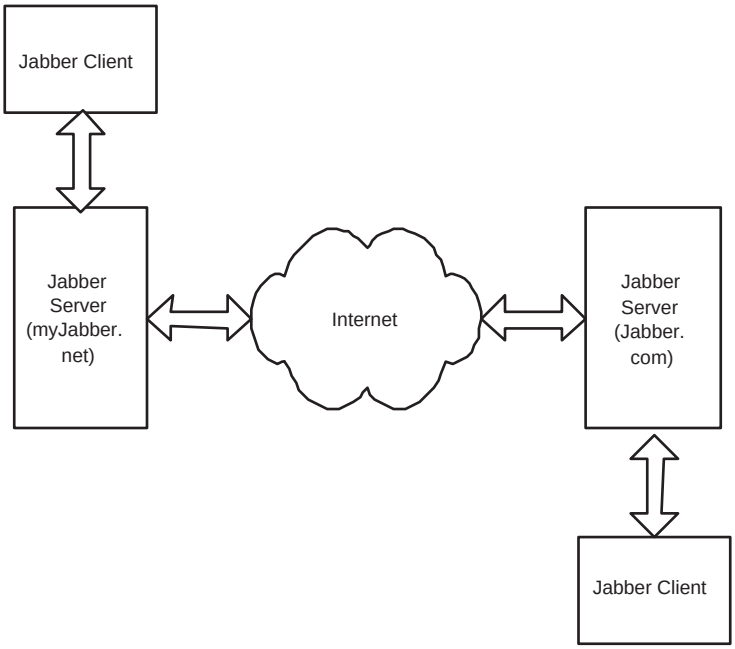

Figure 3.2: Jabber Main Architecture

3.2 Basic Architecture

The simplest form of the Jabber architecture represents a client-server architecture. Jabber clients only talk to the Jabber servers which route the messages to the various users linked

to them. The clients can talk to each other directly through a common protocol. This is not a part of the Jabber system specification though it provides support for this protocol using the out-of-band mode of communication to enable client-client communications. The Jabber system breaks down the IM support to various Jabber servers. Essentially the messages are passed from the sender's Jabber server to the receivers Jabber server and then finally to the receiver's client.

Jabber is client-server

The basic Jabber communication model follows the simple client-server architecture model. Jabber clients communicate only with the Jabber server in this case and also handle requests and information from the end user. The Jabber system has been designed so that the creation of simple clients is favored. Most of the processing and IM logic is carried out on the server side. The clients are assigned minimum responsibilities, which gives the Jabber client developers more flexibility to create clients based on their necessity. This enables us to integrate the jabber IM client with our MARS implementation to produce a customize IM client.

Distributed Jabber Servers

Jabber servers are arranged in a distributed fashion, much like the multiple email servers that are usually present. This is the case with most commercial Jabber server implementations. The Jabber servers are arranged in a domains defined by an internet domain name. As shown in Figure 3.3, we can see that there are two Jabber domains present and the users from one domain can message the users of the other domains. The hub-and-spoke distributed server architecture is quite common in messaging systems [Shigeoka, 2002].

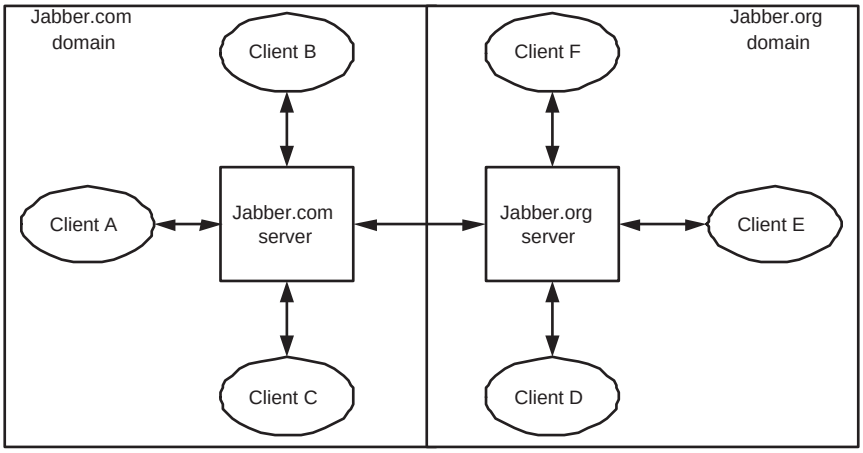

Figure 3.3: Distributed Jabber server with clients

3.3 Core Jabber Protocols

The Jabber protocols define a set of data structures and conventions for exchanging data to carry out IM related tasks [Shigeoka, 2002]. The jabber protocols are simple and flexible and they build upon the standard technologies like TCP/IP, Universal Resource Identifiers (URI) and XML. The core Jabber protocols are listed below with brief explanations

- Message

  The message protocol is perhaps one of the most widely used and most important protocols defined in the Jabber specifications. The majority of the packet traffic on the Jabber network can be classified as the message protocol packets. It is very easy to implement and its basic purpose is to deliver messages across the network. The message protocols have been designed to send human readable messages between users. The following example shows an implementation of the message protocol

  ```
  <message from='jack@jabber.com/work'
   to='mary@jabber.com/home'
  ```

```
 id='messageid2' type='normal'>
<subject>Subject of conversation</subject>
<body>This is the main message body</body>
</message>
```

- Presence

  Presence is another frequently used core Jabber protocol. This protocol governs the subscription, approval, and update of presence information in the Jabber system. Whenever a user logs into a Jabber server, the presence tag sends in the information about his availability to the server and in turn to all the users who have this user in their friend list. This presence tag is a simple way of managing presence efficiently in the Jabber system. The following example shows an implementation of the presence protocol

  ```
  <presence>
  <show>away</show>
  <status>Coffee Break</status>
  </presence>
  ```

- Info/Query

  The Info/Query (IQ) protocol is the last core Jabber protocol and serves as a generic request-response protocol. This protocol is meant to send information or request information from a particular source or the Jabber server. The general format of an IQ protocol contains an <iq> envelope that defines the type of the iq protocol and the desired list of recepients. The IQ protocol also allows us to use some extension protocols. These extension protocols can help us in Jabber account registration and authentication, querying the software version

of the client as well as the server. The following example shows an implementation of the info/query protocol

```
<iq type='set' to='mary@jabber.com'>
<query xmlns='jabber:iq:auth'>
<username>mary</username>
</query>
</iq>
```

3.4 Foundations of Jabber

Jabber was designed in large measure along the same lines as the most successful messaging system on the Internet i.e., email. Thus, Jabber communications are made possible by a distributed network of servers that use a common protocol, to which specialized clients connect to receive messages as well as to send messages to users of the same server or any other Jabber server that is connected to the Internet.

However, while email is a store-and-forward system, Jabber delivers messages close to real time because the Jabber server knows when a particular user is online. This knowledge of availability is called presence and is the key enabler of instant messaging. Jabber combines these standard IM characteristics with two additional features that make Jabber unique.

- The first is an open protocol, which enables interoperability among messaging systems.
- The second is a strong foundation in XML, which makes structured, intelligent messaging possible not only between human users but also between software applications [Adams, 2001].

Jabber's extensible Instant Messaging advantages come from some of the basic architectural features that have been provided. These features are discussed in details in the following sections.

- XML Based:

  All messages, which include the messaging content, presence details, and configuration information are delivered as XML fragments. The ease of development, the universal applicability of XML and the portability to other applications makes Jabber an ideal choice for IM extensions.

- Open API and simple Protocols:

  The Jabber XML based Messaging and Presence Protocol (XMPP) and the programming interfaces for the server are readily accessible. Jabber's completely open protocols enable system enablers to extend their API to meet their specific needs and languages. The protocols are open unlike other implementations, which makes a difference.

- External Component Services:

  Jabber's IM application can use external services for authentication services. Through the XML database Extension (XDB), Jabber makes available external data used to validate and establish privileges for IM users. User profile names, contact info, and personal information that users may want to share are stored in external databases or LDAP.

- Resource Identifiers:

  The Jabber ID or JID identifies each unique user and has an extension, which

is not found in traditional IM systems. Each Jabber ID includes a resource field that makes the specific instance of a user or service uniquely addressable. For example matt@jabber.com includes a resource for work, such as matt@jabber.com/work. These extensions in the Jabber namespace make it possible to determine if the IM corresponds to identify a buddy on a PC or cell phone.

- Presence detection:

  Availability information is a key component of Jabber IM and one that offers significant ability for extension. Presence is the information regarding a contact's readiness to receive information and the state of their system. Because Jabber messages include the capability to capture rich presence information and to notify presence changes, it can be extended to any IM application.

- Client/Server:

  Jabber uses a client-server architecture, not a client-to-client architecture as some instant messaging systems do. All Jabber messages and data from one client to another must go through the server. Any client is free to negotiate a direct connection to another client, but such connections are only for application-specific uses such as file transfers.

- Distributed Network:

  Jabber's network architecture is modeled after that of the e-mail system. Each user has their local server, which receives information for them, and the servers

transfer messages and presence information among themselves. There can exist any number of Jabber servers, which accept connections from clients as well as communicate with other Jabber servers. Each server functions independently of the other, and maintains its own user list. Any Jabber server can talk to any other Jabber server that is accessible via the Internet.

- Modular Server:

  The Jabber server plays two primary roles:

  - Listening for client connections and communicating directly with client applications.
  - Communicating with other Jabber servers.

  The Jabber open-source server is designed to be modular, with specific code packages that handle functionality such as user authentication and data storage (offline messages, rosters, user info and so on). As an example of such modularity, the exchange of messages and presence information between Jabber and any given non-Jabber messaging system is made possible by means of a separate “transport” that translate Jabber XML into the foreign protocol. Such transports are not part of the core server. Instead, they are server-side programs that can be added rather easily to the core server to provide enhanced functionality to the end user.

- Simple Clients:

  One of the design criteria for the Jabber system was that it must be capable of supporting simple clients. Indeed, the Jabber architecture imposes very few restrictions on clients. The only features which a client must support are:

- Communicate to the Jabber server via TCP sockets.
- Parse and interpret well-formed XML packets.
- Understand message data types.

The preference in Jabber is to move complexity from clients to the server. This makes it relatively easy to write clients as well as to update the functionality of the system. Jabber clients communicate with the server in XML through TCP sockets over port 5222, and do not normally communicate with each directly. In practice, many of the low-level functions of the client (e.g., parsing XML and understanding basic Jabber XML such as <message>, <presence>, and <iq>) are handled by Jabber client libraries, enabling client developers to focus on the user interface.

- XML Data Format:

  XML is an integral part of the Jabber architecture because it is of utmost importance that the architecture be fundamentally extensible and able to express almost any structured data. Specifically, Jabber utilizes XML streams for client-server and server-server communication. The XML stream is always initiated from the client to the server, and the lifetime of the XML stream is directly associated with the lifetime of that user's online session.

Chapter 4

Architecture of IM Based Agent Transport

The Internet is today much more than a mere distributed information repository, or a world-wide collection of network services [Omicini et al., 2001]. The Internet constitutes a global, distributed, open, heterogenous, decentralized and unpredictable computing environment. Applications on the Internet are built using various components which are varied in terms of design, architecture, and technology. Our goal is to design a system which makes use of MARS as the referral engine and uses an IM-based transport to communicate messages between the agents. The purpose of this thesis is to make social networking and knowledge management synchronous and involve instantaneous feedbacks. Based on previous research, we know that the concept of referral chains is effective in sharing knowledge within a social community. The concept of "presence" is what makes our approach unique and interesting.

Instant Messaging versus E-mail

We have replaced the existing email transport which uses the traditional store-and-forward messaging model by a IM based protocol. The motivation to replace the message transport system comes from the following reasons:

- Instant messages are delivered instantly. The concept of presence makes a great deal of difference compared to a emailing system.
- IM server's can also act as email server's by storing the message on the server, if the user isn't online and delivering the message once he comes online.
- Typical implementations give IM's a higher priority than e-mail by making the message pop up in a window as soon as it's received.
- Also IM protocols support some of the important features like 1-to-1 and 1-to-many transmission, subscription models and location independence. Although the scope of having a group chat or conference hasn't been evaluated for this thesis, it add's to the list of additional benefits that IM provides.

The previous implementation of MARS used email to transport messages. The message transport layer was implemented using the Java Mail API and users had accounts on a email server. In this thesis, we have redesigned the system involving the existing MARS referral engine to interface with a XML based transport layer, which is compliant with FIPA request interaction protocol. So, inorder to make MARS work with the IM protocol, we have replaced the transport layer of the existing system with an IM system. The Java mail classes have been replaced with the IM client API to wrap communication calls between the various users in a XML format. We built a IM-style GUI, which can be considered as a Jabber client implementation for communicating with MARS. This would enables us to integrate our system with a commercial IM in future.

The IM System has been implemented using the open source XML based Jabber protocol. The IM client has been designed to handle communications with the MARS framework. The IM client embeds the message in a specific format so that the agent can interpret the message and act based on the contents. The MARS API embeds the required message into a Jabber protocol format for the Jabber server to understand. To show compliance

with the FIPA interaction protocols, we have embedded the interaction stages within the payload of the message. The Jabber Server interprets the message which is representated in a specific XML format. The XML representation is a predefined set of definitions for the various actions performed by the Jabber server. The IM client communication with the MARS agent is compliant with the FIPA interaction protocols. The request/response messages are sent from the client to the agent based on the FIPA request interaction protocol specifications.

MARS was designed to build an intelligent social network which can direct queries to users who have the required expertise in that domain. Referral chains represent social networks, which are used to locate experts in a particular domain. In this system, each user is assigned a software agent.

The architecture of the entire system is shown in Figure 4.1. We have introduced a Jabber based IM framework for the communicating messages between the various users. This figure is a high level design of the system.

4.1 Architecture Description

The Figure 4.1 shows the various components involved in this system. The architecture shows the distributed nature of this system. The idea is to find the experts in every domain so that a query pertaining to that domain can be sent to those experts.

The basic components involved in this architecture are

- Jabber Server
- Jabber Client
- MARS Agent
- Jabber User Directory (JUD)

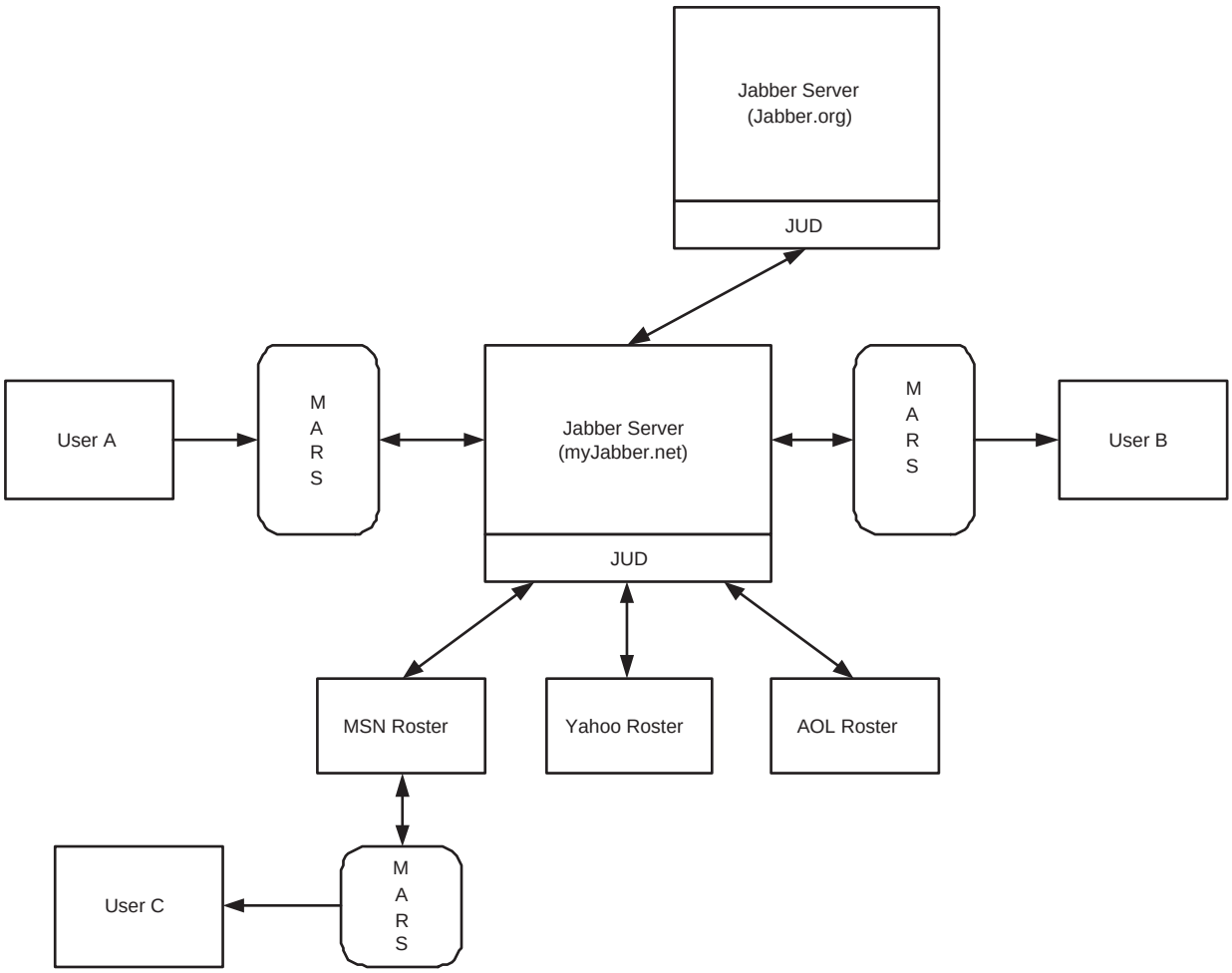

Figure 4.1: Architecture involving Jabber and MARS

The Jabber Server is the main component where most of the activity takes place. The server stores all user accounts. The directory of users and their login and passwords are stored in a LDAP directory or a database. This component is generally called the Jabber User Directory (JUD). Whenever a user logs into the system, his userid and password are authenticated with the stored records in the directory or the database. Most of the innovations or new components are added as plugins into the server module. The Jabber server also handles the most important task of routing the messages. The messages are

in a pre-defined XML format which the server understands and then based on the tags in the message, it forwards the messages. The Jabber server is designed to accept and handle incoming client connections, parse XML streams over that connection, and react to Jabber commands sent over the XML stream. The general modules of a Jabber server are as follows:

- A session Pool
- An XML Parser
- Packet Handlers
- Plugins for other IM providers.

The session pool helps the server maintain active sessions for the users who are logged in. The XML parser is used to interpret the XML packets by parsing them. The packet handlers implement a multi-threaded packet queue which takes care of the various packets arriving from various users. Also the plugins help the Jabber server communicate with other commercial IM systems like AOL, MSN and Yahoo. These are additional packages deployed on a Jabber server which helps in interoperability with other commercial IM systems.

The Jabber client has been designed to send messages to the Jabber server. Messaging is the heart and soul of every IM system [Shigeoka, 2002]. The client establishes a connection with the server and then communicates with the server by sending packets to and receiving packets from it. The MARS agent communicates with the referral engine which determines the neighbor list for every user. It also decides the users to whom the message has to be sent at runtime based on the query and domain. It determines the expertise and sociability of the neighbors and then sends them the message accordingly. The Jabber User Directory stores the list of users that are registered with that Jabber Server. It contains enough data to authenticate a user and establish a session with that user.

The Jabber server is generally designed as shown in Figure 4.2. The basic server operation starts when a client connects to the Jabber server. The server creates a session object for each connection that it assigns. The XML parser generates packets which contain the parsed XML message. These packets are then pushed into a packet queue. The server then processes each packet in this queue and accordingly generates a response. The response is then packaged into an XML outgoing stream and associated with the correct session object. The session pool, XML parser and packet queue work together to support processing of Jabber packets.

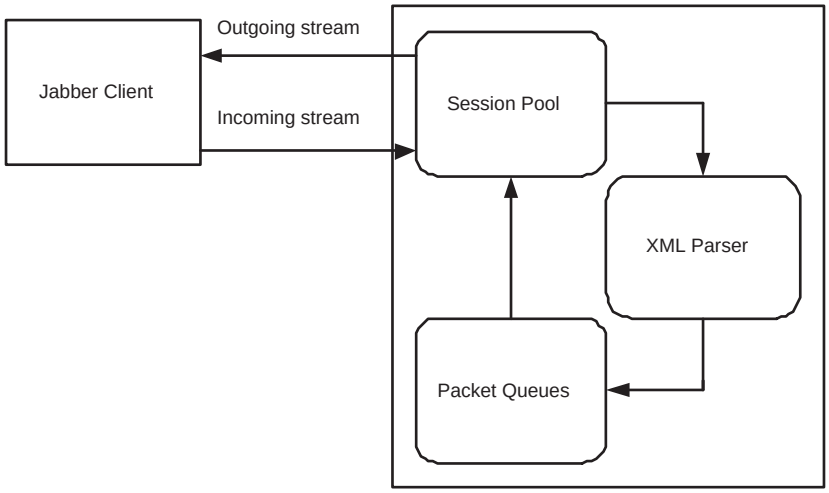

Figure 4.2: Organization of Jabber Server

4.2 Design Assumptions

The main purpose of using IM is to make information exchange easier and momentaneous. There exist many commercial IM clients which are extremly popular with the existing audience. This introduces a second requirement for us i.e., to integrate the existing IM implementations with our design since users may not want to install different IM clients

for every application. The Jabber IM system is an ideal choice for us since it provides an interface for adding plugins to other popular IM clients. This combines the friend's list from all the IM accounts into a single roster. Existing users who are registered with the MARS system and want to use this system will have to use our IM client. The system is generic enough to pass on the query to any user on the roster list. The end user could be a MSN user or a Yahoo user, but he needs to use the same client. This is because, only our IM client code interacts with the MARS engine. The MARS system handles the neighbor model and also the referral list for the various users. Once the message reaches the Jabber server, it will parse the message and when it finds the end user to be a MSN or a Yahoo user, the plugin will translate the above message in a format that the MSN or Yahoo messenger understands.

4.3 Jabber Instant Messaging Client

The IM-based solution is quite similar to the traditional client-server architecture. The IM client connects to the Jabber server and establishes a session with the server. The IM client acts as a user interface for the end user to send his messages across the network to another user. The IM client usually starts by opening a socket connection with the server. In our design, we have a IM client which loads with the default settings. Only when the user is authenticated, the other features of the client load. Our goal is to design a simple client which is easy to use and serves the purpose of integrating a referral engine with an IM client. The GUI implementation for MARS using Jabber is made user-friendly and with enough features for the application to work with MARS. The IM client supports the basic protocols involved in Jabber. These are as follows

- Messaging - Sending and receiving Jabber instant messages.

- Presence - Sending and receiving presence information.
- Roster Management - Subscribing and maintaining your online presence status.
- Authentication - Logging in to a Jabber Server.

Other features like registering with the Jabber server have been combined with the intial implementation of MARS registration. The idea is to combine the process of registeration with MARS as well as creating an account with the Jabber server through which the messages will be routed. The IM GUI has been redesigned to integrate with MARS and also enable IM-based communication. The IM client's basic responsibilities involve the following tasks

- Connect to Jabber Server
- Send an opening <stream:stream> tag to intiate the message transfer.
- Send <message> packets.
- Receive and display <message> packets.
- Send a closing </stream:stream> tag.
- Disconnect from the Jabber server.

4.4 Authentication and Roster Information

Every user needs to register and have an account with the MARS system. Once the user starts the application, he has to log into the system which effectively logs him with a Jabber server. The Jabber server authentication is an extension of the IQ protocol. This protocol allows clients to prove their identity to the server. Based on the authentication the client gets a list of users that are listed as friends in the roster, which are then listed in the IM client GUI. There is no standard authentication protocol set for the Jabber server. The Jabber server authentication is directly tied to a user account. Users who do not have an

account with the Jabber Server can register with Jabber to get an account.

The Jabber authentication protocol is defined in the jabber:iq:auth namespace. It consists of two phases. The first phase is called the probe phase in which the user determines the authentication methods supported on the server. This phase returns authentication tokens required for the next phase. The second phase is to attempt authentication by sending an authentication query containing proper authentication credentials. Most Jabber servers have a set type of authentication and do not support multiple types of authentication. In the probe phase, the authentication method specifies a get query along with the user account name.

4.5 User Presence Detection

IM is differentiated from email by its instant delivery and the ability for users to project and detect each other's online presence [Shigeoka, 2002]. Presence enables a user to know if another user, who is in his friends list is online so that he can send him a message. Based on an user's presence status, one decides to involve him in a group chat or other online IM based collaborative activity. Jabber protocol provides a mechanism by which presence information about an user can be propogated to all the users.

The basic presence protocol is used in two primary contexts:

- Presence update: Informs people of your current presence state.
- Presence subscription management: Allows people to subscribe to another user's presence.

In both the above cases, the Jabber server acts as an arbitrator between the presence information generator and presence recepients. The presence update protocol uses a simple one way message. A client sends the presence update packet to a server. The server forwards

the presence packet to the various user's who are on the sending user's presence subscription list. The presence packet uses a <presence> element with standard Jabber attributes like to, from and type.

4.6 Message Formats

The message needs to be in a specific format so that the MARS API can parse and evaluate the message. Based on the requirements of the message format, the IM system builds a message and sends it to the MARS system which then evaluates and responds based on the message. A typical MARS message can be represented by the following format according to [Mo, 2000]

originID	originName	qid	msgType	domain	toID	toName	content

Table 4.1: Typical MARS message

where

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, referral, answer

domain: The domain field related to the query

toID: Destination agent's ID

toName: Name of the destination agent

content: Message content along with FIPA interaction protocol stage payload

Referral Scenarios

The following message formats represent the various referrals scenarios that can exist in this system

Case 1: Send out a query

myID	myName	qid	QUERY	domain	receiver's ID	receiver's name	content

Table 4.2: Initial Query

Case 2: Receive a query and then answer it back

sender's ID	sender's Name	qid	QUERY	domain	myID	myName	content

Table 4.3: Received Query

Since i have the expertise in that domain, my agent display's the message on my screen. If i decide to reply, the answer will be sent in the following format

myID	myName	qid	ANSWER	domain	sender's ID	sender's name	content

Table 4.4: Answered Query

Case 3:Receive a query, refer it to neighbors

If the user gets a query in a certain domain and he doesn't have expertise in that domain, he decides to refer some friend from his neighbor's list. In this case a referral message will be sent back to the originator with the friend's ID in the query.

refID	refName	qid	REFERRAL	domain	friend's ID list	name list

Table 4.5: Typical Referral

Case 4: Receive a referral, then send the queries to all the users who have been referred.

The referral message which will be received is in the following format

refID	refName	qid	REFERRAL	domain	friend's ID list	friend's name list

Table 4.6: Referral message

The originator's agent then automatically sends out the query to those who were referred. For each query message

myID,refID	myName, refName	qid	REFQUERY	domain	recv's ID	recv's name

Table 4.7: Refquery message

Case 5: Receive a query, ignore it

Either the user and all his friends from his friend list do not have expertise or the user intentionally doesn't want to answer the query.

Case 6: Receive a answer back If a user receives a answer back from a user who had the expertise in that domain, the message is in the following format

myID	myName	qid	ANSWER	domain	answerer's ID	answerer's name

Table 4.8: Answer a Query

4.7 FIPA Interaction Protocols

One of the main goals of our implementation is to make our application compliant with the FIPA interaction protocols. The intent behind this to develop a FIPA based interface which enables our application to be replaced by any other agent based application which is FIPA compliant. Our implementation is compliant with the FIPA Request Interaction Protocol and FIPA Query Interaction Protocol. The FIPA interaction protocols specify the pattern of the communication between the request/query initiator and the other participants. The following figures show the FIPA Query Interaction Protocol and FIPA Request Interaction Protocol as implemented by our application.

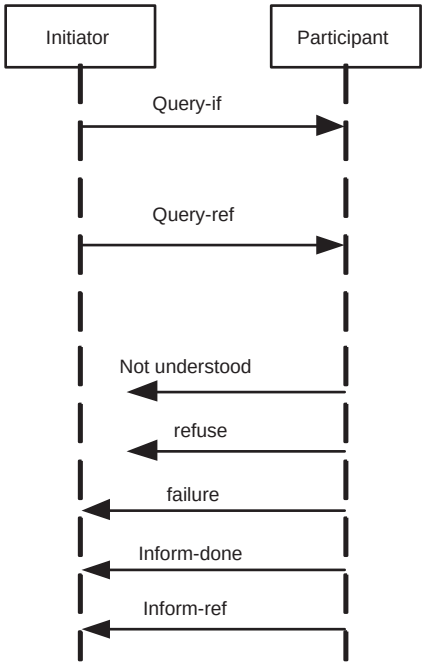

Figure 4.3: FIPA Interaction Protocols

4.8 Integration with MARS

The existing MARS system needs to be integrated with our IM-based message transport layer. Other than the communication layer, the system needs to have a common user registration process in order to form virtual communities.

4.8.1 Registration

Every user in the social community who wants to be a part of this system has to be registered with our system. The registration process involves setting up an id in the Jabber Server. This process also involves some tasks pertaining to the expertise vector mapping for the domains or ontologies. The domains are pre-defined based on users interests. When the user desires to be a part of the social network to share information, he has to register with our software. The Jabber client needs to be installed prior to using the registration module. The client has a button which takes the user to the registration module. The registration module has certain fields like username, password etc. which need to be filled up. Along with these credentials the user needs to enter his expertise in some domains. This can be regarded as a bootstrapping process in which each user can set his expertise for some of the domains. These values need to be between 0.0 to 1.0. These values indicate the level of expertise an user has in each domain. Once the user fills up these values, the system registers him with the Jabber server. The Jabber server has a JUD which enables a user to have an account. This is the primary ID by which every user will communicate with. Initially the ID's are assigned as mars01@jabber.com format for our test runs, which indicates that he was the first user who registered with the Jabber based MARS prototype and the Jabber server used in this case is jabber.com.

4.8.2 Message Flows

The communication between the various users are framed as messages. These messages need to be sent to the users who have expertise in certain domains. The message is sent only to those users who are listed in that user's neighbor models. The MARS agent figures out the list of user's in their neighbor models and checks their expertise and sociability values. Based on these values, the message is sent to the appropriate Jabber server which then redirects the message the appropriate user. The Jabber server then parses the XML stream and sends the message accordingly to that user. Once the message arrives at the other user's end, the MARS agent representating that user analyzes the message to determine if that user has the necessary expertise in that domain. If that user has the expertise in that domain, it shows the message to the end user. Incase the user doesn't have the necessary expertise, the MARS agent builds a referral list of users who may have the expertise or have a good sociability level based on the user's present in his neighbor list. This referral list is then sent back to the originator, whose agent then analyzes the message and sends a refquery message to all the users who have been referred by the other user. These users are also added to the originator's neighbor list with some default values.

Figure 4.4 shows a simple scenario in which a message is sent from a user A to user B. User A sends a query related to some domain. The MARS agent which represents user A decides to send the message to user B since user B either has the expertise in that domain or has a high sociability level.

4.9 Adapters and Plugins

The adapters and plugins are additional components which can be installed on the Jabber Server. Commercial Jabber server's have implemented plugins for MSN, AOL and Yahoo

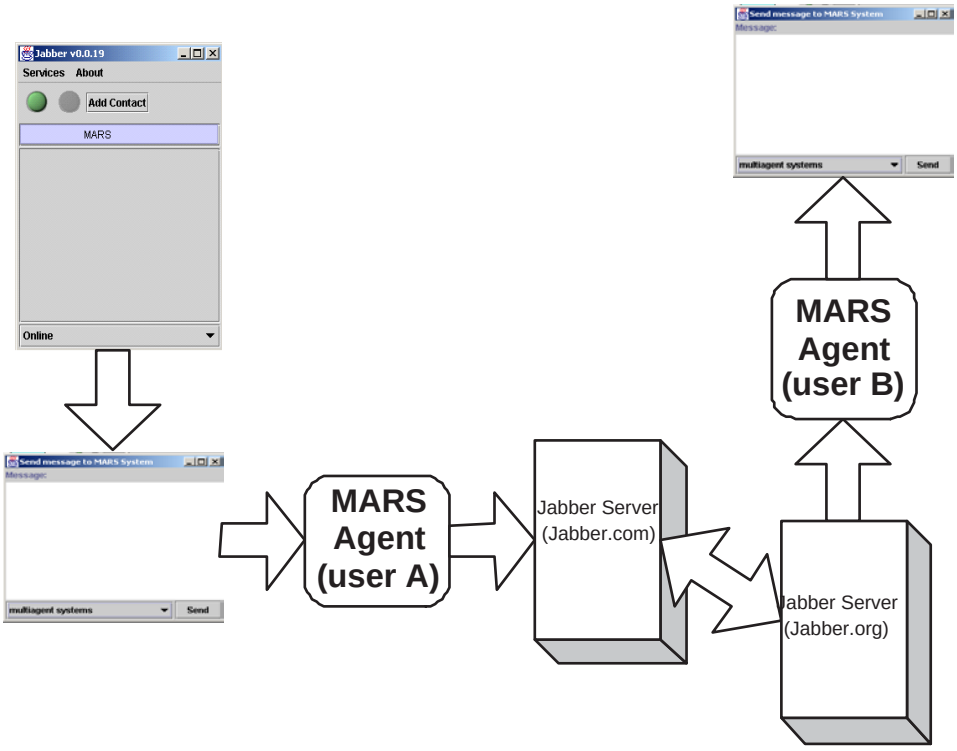

Figure 4.4: Message flows from User A to User B

transports. These plugins translate the Jabber XML based representations to their proprietary format. So when a Jabber user wants to send a message to a MSN user, the Jabber server handles the communication via the server plugin for MSN transport. There are other plugins available for other tasks like sending raw XML tags. There are various adapters implemented for login and authentication of Jabber users on the jabber server. Authenticating login and passwords with entries in a LDAP server is one of the common adapters which almost most of the Jabber servers provide. There are other adapters which provide logging facility for the Jabber server. This enables logging of every error or exception that is thrown during the process of communicating the message from one user to another.

Chapter 5

Implementation of MARS using IM Transport

In this thesis, we have demonstrated a working version of MARS using the IM protocol. Originally MARS was implemented by using email as the message transport mechanism. This implementation serves as the basis for our proof of concept which states that the MARS prototype can be embedded into an IM-based implementation. The fact that we chose Jabber as our IM platform gives us the benefit to integrate our solution with any of the existing commercial IM applications. Also, we were successful in complying with FIPA based interaction protocols such as the Query Interaction Protocol and the Request Interaction Protocol which enables our application to interact with other FIPA compliant systems.

The application has been built using Java 1.3 SDK and Jabber XML schema definitions. The original application had implemented the agent reasoning and learning system using the IBM ABLE framework. We used the existing MARS API to make calls to the reasoning system and build the neighbor model for each user. We introduced a new set of API to handle communication with an IM based transport. The ABLE reasoning engine needs to be setup using a ruleset file which is specified using the ABLE Rule Language (ARL). Information about the users and the referral lists are all stored in the file system. These files

need to be accessed and updated based on referral scenarios.

MARS implementation involves several programming toolkits and utilities [Mo, 2000]. The learning and reasoning engines use IBM Research Center's ABLE 1.4a software along with SUN's JavaMail API and POP3 implementations. The API involving e-mail infrastructure has been removed in this implementation since we replaced the transport layer by an IM protocol.

5.1 Implementation Environment

The implementation environment describes the default environment to run our application. We have developed and tested this application on Windows 2000 environment as well as a Linux based environment. The ideal soution would involve installing and configuring a Jabber server of our own which would also include the plugins for all the other IM systems like MSN, Yahoo and AOL. We decided to use a commercial Jabber server i.e. Jabber.com since we wanted to test whether our application in a real world setting.

The application has been built on an open source Java IDE environment called Eclipse from eclipse.org. We are running the Eclipse 2.01 version and used it for the code development as well as running the application using the Eclipse launcher which sets the java execution environment for us which includes setting up the classpath dynamically. The application also generates XML strings which have to be compliant to the Jabber XML schema since we use a commercial Jabber server. The Jabber server interprets the instructions which are embedded in the XML strings. There are some specific information that are passed as a part of the message payload and need to be evaluated by the MARS referral engine. The FIPA message indicates that the communication between the various agents are compliant with the interaction protocols.

5.2 Mars and Jabber Integration

The email mechanism has been replaced by the IM transport since it provides us with a synchronous messaging model which is essential for effective social networks. The referral engine used by the present application is a part of the existing MARS system. The interface to the MARS referral engine had to be changed to make the IM based transport operate with it. A new set of API has been provided for this purpose. The integration of the new transport mechanism using IM based messaging with MARS can be seen in the code flow UML representation in Figure 5.8.

5.3 Code Flow

The complete code is packaged in two main structures, one of which contains the entire MARS code base and the other which contains the code base for the XML based message transport layer and GUI interface. The MARS code is packaged into a jar file and the location of the jar file is put in the classpath. The entire code is structured into the following packages

- package mars
- package mars.engine
- package mars.engine.referral
- package Jabber

The basic code flow within the system can be representated by figure 5.8.

5.4 GUI designs

We have used Java Swing API for developing the GUI for this application. The previous version of MARS also has a Java Swing based GUI which makes it easy for us to integrate both the systems. Our application uses the existing registration system that the MARS implementation provides us. Also the evaluation component is linked through the previous MARS implementation. This component helps evaluates the answers of the query from the various users.

The GUI designs which pertain to the IM based MARS system are as follows. The figure 5.1 shows the main console for MARS. There are certain users who are present in the user's roster or friend's list and there is a static entry for the MARS agent.

Figure 5.1: MARS-Jabber UI

Figure 5.2 shows the login console for a MARS user.

Figure 5.2: MARS-Jabber Login screen

Figure 5.3 shows the message query box which has a drop down containing the list of ontologies.

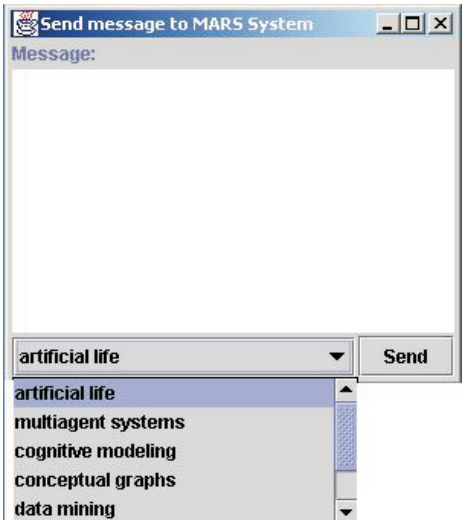

Figure 5.3: MARS-Jabber Message screen

Figure 5.4 shows a message that is received from some user. The message has the complete string with the message embedded in it. It also provides some textarea to reply back to the message.

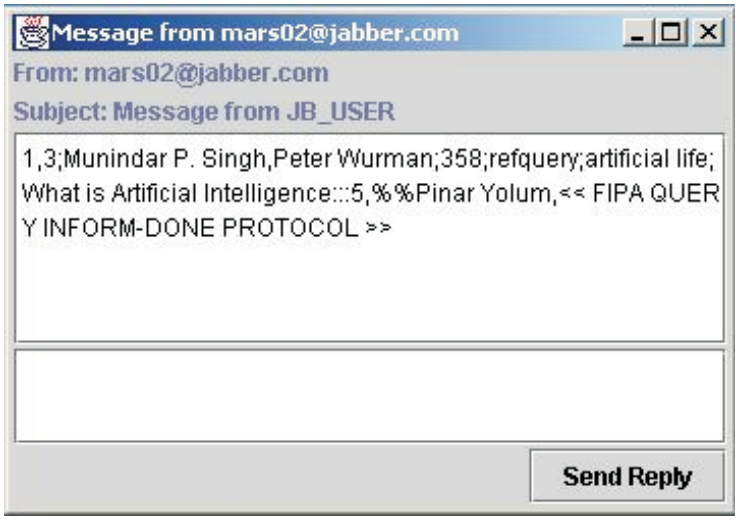

Figure 5.4: Message received by IM client

5.5 FIPA query interaction protocol compatibility

A simple representation of the FIPA compliant queries, referral's and answers are shown in the following figures. The numbers in the figures indicate the agents who are a part of the social network. These are some sample scenarios that we simulated in the lab. These are a small subset of the number of test cases that we have executed. These figures have been shown with a small fraction of the content payload that they carry along with the message that needs to be communicated

Figure 5.5 shows a scenario where Agent 1 initiates a query. Agent 9 is in his neighbor model and it satisfies the sociability condition. So the query is sent to Agent 9, who then figures out that it does not have the required expertise in that domain and sends a referral for Agent 3 to Agent 1. Agent 1 then add's Agent 3 to his neighbor model and sends the query to Agent 3. Along with the queries, the various FIPA interaction protocol stages are also indicated.

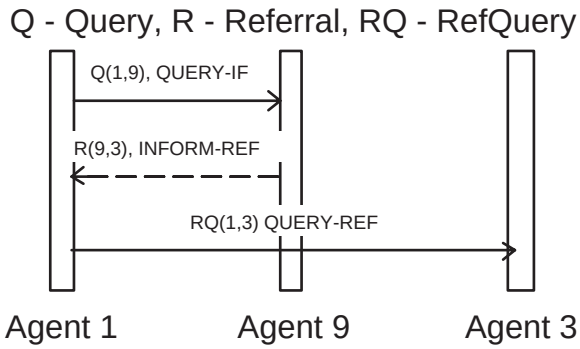

Figure 5.5: Query Testcase

Figure 5.6 shows a scenario in which Agent 1 has Agent 9 and Agent 3 in its neighbor list. When it sends a query to both the neighbors, Agent 9 does not answer back. Agent 3 finds that it does not have the expertise and it refers Agent 5 to Agent 1. Agent 5 is added to Agent 1's neighbor models and the query is sent to Agent 5.

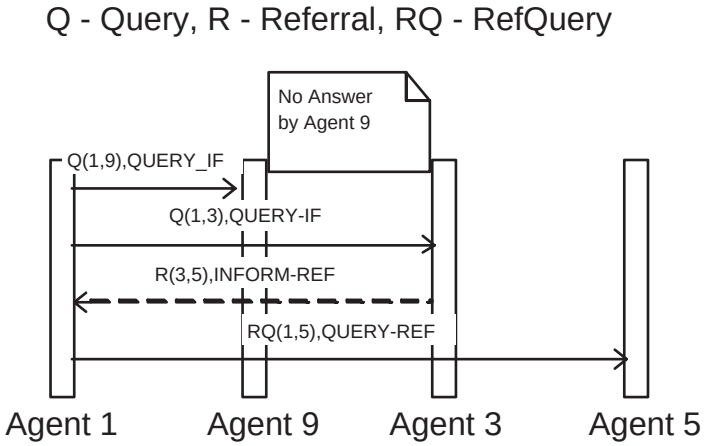

Figure 5.6: Query-Referral Testcase

Figure 5.7 shows a complex referral scenario in which Agent 1 sends a query to Agent 9, who refers Agent 3, who in turn refers Agent 5. Finally Agent 5 decided to accept the query since it has the expertise. The various FIPA interaction protocol stages are indicated. Once Agent 5 answers back the query, Agent 1 evaluates the answer.

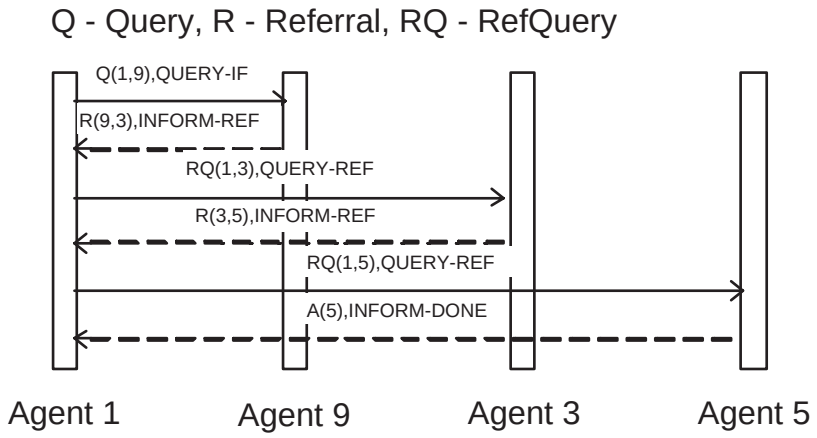

Figure 5.7: Query-Referral-Answer Testcase

5.6 Testing Considerations

We tested our application with some limited environment settings in the lab. We have some pre-defined test cases, based on which we evaluate if our application is giving us the desired results. This application is a proof of concept to show that IM based protocol can be integrated with the original MARS application.

We setup a distributed environment for executing the test cases. The test simulations involved small runs and predictable results. We installed the entire code base in three machines and made multiple agents run on each machine. The test cases involved simple queries as well queries which involved multiple referrals. Based on this setup we were able to prove that our application works fine under the given circumstances. This application is effective and can be used if it is applied to an existing social community which have a set of users with an existing neighbor model.

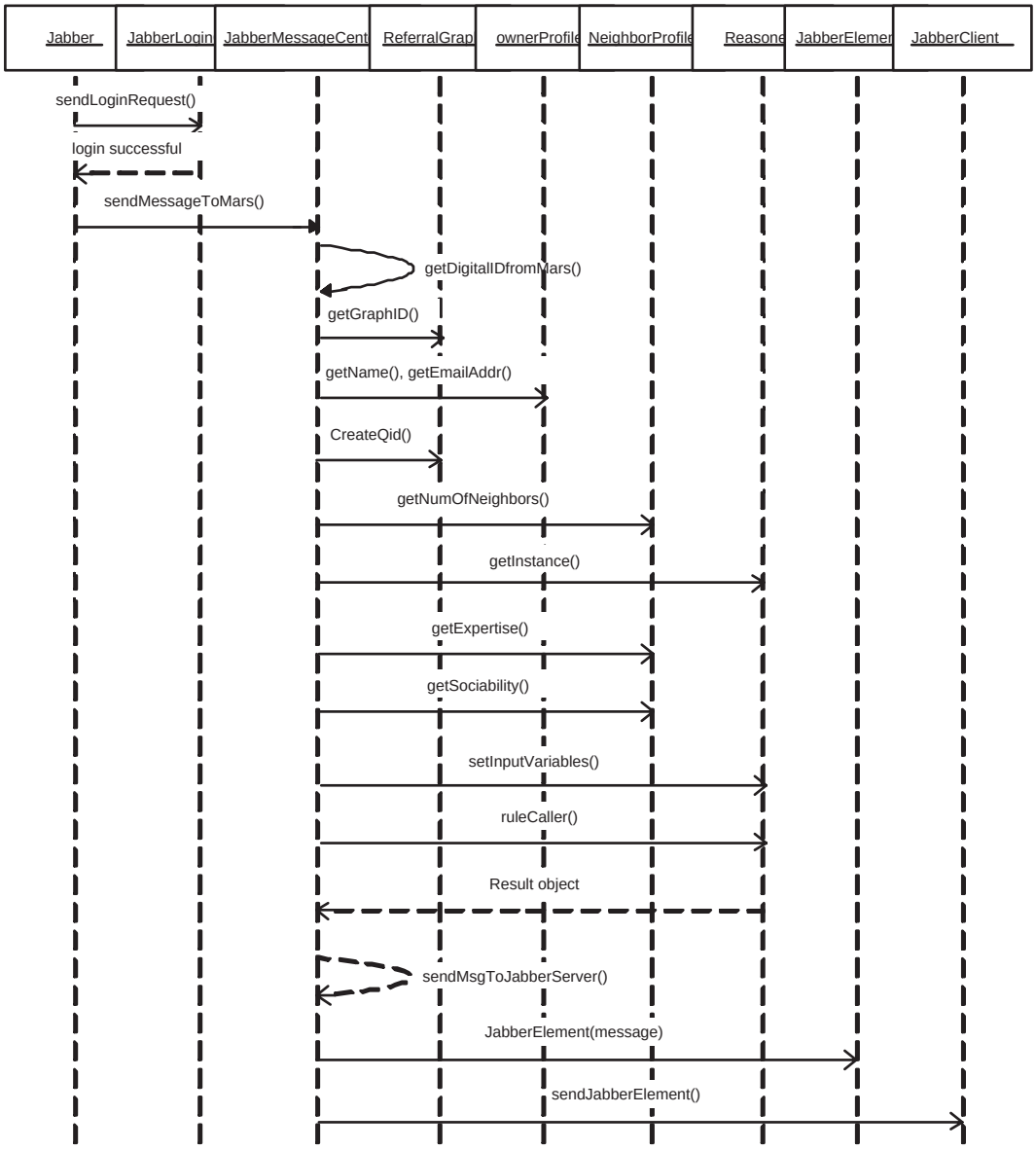

Figure 5.8: Code flow between the various modules

Chapter 6

Discussion

This thesis presents a new approach towards building a system that supports social networking. In this thesis we have shown how we can use an IM architecture to support a multiagent based referral system. Since we are designing a system which involves social networking between various users of the system, it needs to handle a lot of communication involved in a social community. IM seems to be an ideal choice in such an environment. IM based referral system lets the users of the system communicate with each other so that they can send queries and referrals. Chat programs and IM services are very popular among internet users, but basic issues with the interfaces and data structures of most forms of chat limit their utility for use in formal interactions and decision-making tasks [Smith et al., 2000]. In this thesis we have successfully modeled our application involving multi-agents which make complex decisions using the IM protocols.

The functionality of the MARS using IM as a transport has been tested with a small scale deployment on a home-based network with ten users sending queries and referrals simultaneously. The various functionalities provided by the agent based system were simulated and a set of unique test runs were excecuted. With the wide scaled proliferation of IM-based protocols today, agent based referral networks using the IM protocol would

play an important role in reducing communication chains and identifying the users who are knowledgeable in their respective domains. The availability of a global communication infrastructure and its ubiquitous accessibility, suggests implementing solutions in a truly distributed manner which gives more deployment scenarios for such a heterogeneous architecture.

6.1 Related Work

MARS is a social networking application that can be classified as a special kind of a recommender system. Typical recommender systems can be classified into the following categories

- Content based filtering: Content based filtering systems use information retrieval schemas to fetch documents. This kind of a system is used to locate articles are documents. It recommends documents based on the similarity with the documents which a particular user liked in the past. Text documents are recommended based on a comparison between their content and a users profile. Examples of such systems are InfoFinder [Krulwich and Burkey, 1996] and NewsWeeder [Lang, 1995]. The concept of message transport does not apply to such a system.
- Matchmaker systems: In a centralized matchmaker system, a central server maintains information about user interests and the users connect to the server to discover whether they have a match. Examples of such an implementation are Webhound [Lashkari et al., 1994] and Firefly [Shardanand and Maes, 1995]. Compared to these approaches, MARS is a totally distributed system. It does not force to cluster the agents together with similar interests, although virtual

communities will be formed via agents interactions [Mo, 2000].

- Other Referral systems: There are other referral systems which have been in existence for quite some time. Examples of those include ReferralWeb [Kautz et al., 1997b] is a centralized referral system, in which co-occurence of names in close proximity on web pages is used to suggest direct person-to-person relationship. Then there is one known as ContactFinder [Krulwich and Burkey, 1996], which is a agent that reads messages posted on bulletin boards, extracting topic areas and using a set heuristic. Compared with MARS, these referral systems have limited functionality and can be used only in certain scenarios [Yu et al., 2001].

6.1.1 Intelligent IM Agents for Collaborative Learning

A lot of work is been done in introducing intelligent agents using IM in the fields of collaborative learning. Since the concept of IM has become so popular in workplaces, there is an initiative to effectively use this infrastructure for project learning and information sharing. Research shows that this technology can be effectively applied for an educational venture, thus helping people work in groups irrespective of their locations and schedules. Examples include BuddySpace [Eisenstadt and Dzbor, 2002].

6.1.2 Finding friends using XML and RDF

A project called Friend-of-a-Friend (FOAF) evaluates the social networking aspect. This work suggests to have some kind of representation for social communities which have proliferated through the use of internet. The FOAF vocabulary gives a basic expression for community membership describing people and their basic properties.

6.1.3 Interactive agents

Interactive agents like Activebuddy interact with users over the messaging services like IM and web chat. These agents act as automated query agents and respond to queries based on data which is stored in some data repository. Another example of such a system is ELLEgirlBuddy, which offers fashion tips to people.

6.2 Limitations with our current design

As with every software architecture, our system also has certain limitations and restrictions on the executing environment. Most of these restrictions come into picture because of the underlying protocols used by our implementation. Since we are dependent on IM and Jabber protocols, we are bound by their limitations. Some of these limitations can be overcome by additional design changes and implementation tweaks which are listed in this thesis as further directions.

The limitations and restrictions can be listed as follows:

1. Since we are using IM based protocol to model a social networking environment, one may get a lot of messages popping up on his screen from unknown users. This could lead to bad referrals or no response from a user.

2. This application operates in a typical client-server environment where the IM clients send a message to the Jabber server, which then sends it accordingly to the final user. In this case, the Jabber server can prove to be a single point of failure to jeopardize the entire application since all the messages are routed through the Jabber server. Scalability and performance of the server are also key concerns for this application to work.

6.3 Future Directions

This thesis presents an IM-based approach for implementing the message transport mechanism underlying the agent communications layer. It discusses the implementation and benefits of having a IM based message transport protocol which is compliant with the FIPA request and query interaction protocols.

Several interesting enhancements are possible. One of them is to create a XML schema for storing the user information and his neighbor model. This would make the data underlying the MARS model much clearer. Also it would make searching for sociability and expertise values for specific ontologies easier. Since we use a XML based message transport mechanism, it would fit well into the present model. We would like to link the roster entries of each user with the neighbor model so that we can query them directly whenever we have a query pertaining to a domain in which they have the desired expertise. Also we would like to exploit the possibility of integrating this architecture with existing IM systems such as MSN or AOL, since IM systems are getting popular in a corporate environments. This would enable us to evaluate our implementation with a wider audience.

Bibliography

D. J. Adams. *Programming Jabber*. O'Reilly, 2001.

Frank Dignum and Mark Greaves. Issues in agent communication. In *Issues in Agent Communication*, volume 1916 of *Lecture Notes in Computer Science*. Springer, 2000. ISBN 3-540-41144-5.

P. Dourish. Where footprints lead: tracking down other roles of social navigation. In *Social Navigation of Information Space*, London, 15–32 1999.

Marc Eisenstadt and Martin Dzbor. Buddyspace: Enhanced presence management for collaborative learning, working, gaming and beyond, March 2002.

T. Finin, R. Fritzson, D. McKay, and R. McEntire. KQML as an Agent Communication Language. In N. Adam, B. Bhargava, and Y. Yesha, editors, *Proceedings of the 3rd International Conference on Information and Knowledge Management (CIKM'94)*, pages 456–463, Gaithersburg, MD, USA, 1994. ACM Press. URL `citeseer.nj.nec.com/finin94kqml.html`.

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

Malcolm Gladwell. Six degrees of lois weisberg. *New Yorker*, pages 52–53, January 1999.

Mark Granovetter. The strength of weak ties. *American Journal of Sociology*, 78:1360–1380, 1973.

Thomas R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition 5*, (2):199–220, 1993.

Henry Kautz, Bart Selman, and Mehul Shah. The hidden Web. *AI Magazine*, 18(2):27–36, 1997a.

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

B. Krulwich and C. Burkey. Learning user information interests through the extraction of semantically significant phrases. In *Working Notes of the 1996 AAAI Spring Symposium on Machine Learning in Information Access*, March 1996.

Ken Lang. NewsWeeder: learning to filter netnews. In *Proceedings of the 12th International Conference on Machine Learning*, pages 331–339. Morgan Kaufmann publishers Inc.: San Mateo, CA, USA, 1995. URL `citeseer.nj.nec.com/lang95newsweeder.html`.

Y. Lashkari, M. Metral, and P. Maes. Collaborative Interface Agents. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, volume 1. AAAI Press, Seattle, WA, 1994. URL `citeseer.nj.nec.com/lashkari94collaborative.html`.

T. Magedanz, K. Rothermel, and S. Krause. Intelligent agents: An emerging technology for next generation telecommunications? In *INFOCOM'96*, San Francisco, CA, USA, 24–28 1996. URL `citeseer.nj.nec.com/magedanz96intelligent.html`.

Wentao Mo. A referral based recommender system for e-commerce. Master's thesis, North Carolina State University, August 2000.

Bonnie A. Nardi, Steve Whittaker, and Erin Bradner. Interaction and outeraction: instant messaging in action. In *Computer Supported Cooperative Work*, pages 79–88, 2000.

Andrea Omicini, Franco Zambonelli, Matthias Klusch, and Robert Tolksdorf. Preface: Coordination of Internet agents. In Andrea Omicini, Franco Zambonelli, Matthias Klusch, and Robert Tolksdorf, editors, *Coordination of Internet Agents: Models, Technologies, and Applications*, pages IX–XVI. Springer-Verlag, March 2001. ISBN 3-540-41613-7.

Upendra Shardanand and Patti Maes. Social information filtering: Algorithms for automating "word of mouth". In *Proceedings of ACM CHI'95 Conference on Human Factors in Computing Systems*, volume 1, pages 210–217, 1995. URL `citeseer.nj.nec.com/shardanand95social.html`.

Ian Shigeoka. *Instant Messaging in Java*. Manning, Greenwich, 2002.

Marc Smith, Jonathan J. Cadiz, and Byron Burkhalter. Conversation trees and threaded chats. In *Computer Supported Cooperative Work*, pages 97–105, 2000. URL `citeseer.nj.nec.com/smith00conversation.html`.

Stanley Wasserman and Katherine Faust. *Social Network Analysis*. Cambridge University Press, New York, 1994.

Bin Yu, Mahadevan Venkatraman, and Munindar P. Singh. An adaptive social network for information access: Theoretical and experimental results. *Applied Artificial Intelligence*, 2001. To appear.