Use of Latent Semantic Indexing for Content Based Searching and Routing of Mobile Agents on P2P Network

A thesis submitted to Middlesex University
in partial fulfilment for the degree of Master of Philosophy

by M. Singh

School of Engineering and Information Sciences

Middlesex University

March 2010

Abstract

The peer-to-peer (P2P) system has a number of nodes that are connected to each other in an unstructured or a structured overlay network. One of the most important problems in a P2P system is locating of resources that are shared by various nodes. Techniques such as Flooding and Distributed Hash-Table (DHT) has been proposed to locate resources shared by various nodes. Flooding suffers from saturation as number of nodes increase, while DHT cannot handle multiple keys to define and search a resource. Various further research works including multi agent systems (MAS) have been pursued that take unstructured or structured networks as a backbone and hence inherently suffer from problems. We present the solution that is more efficient and effective for discovering shared resources on a network that is influenced by content shared by nodes. Our solution presents use of multiple agents that manage the shared information on a node and a mobile agent called Reconnaissance Agent (RA), that is responsible for querying various nodes. To reduce the search load on nodes that have unrelated content, an efficient migration route is proposed for RA, that is based on cosine similarity of content shared by nodes and user query. Results show reduction in search load and traffic due to communication, and increase in recall value for locating of resources defined by multiple keys using RA that are logically similar to user query. Furthermore, the results indicate that by use of our technique the relevance of search results is higher; that is obtained by minimal traffic generation/communication and hops made by RA.

Keywords: Resource Discovery, P2P, Reconnaissance Agent, Latent Semantic Indexing, Cosine Similarity.

Acknowledgements

I would like to acknowledge and extend my heartfelt gratitude to the following persons who have made the completion of this thesis possible:

My supervisors, Dr. Xiaochun Cheng and Dr. Roman Belavkin, for their vital assistance, encouragement, and support throughout the duration of the research project.

Late Emeritus Prof. Colin Tully for giving me the chance to pursue postgraduate studies. His initial encouraging words have been driving force for me throughout this work. May he rest in peace.

Mr. Sumeet Gautam for his assistance in collection of the topics for background chapter and testing.

Most especially to my family

And to **Akal Purakh Waheguru**, who made all things possible.

M. Singh

Middlesex University

Contents

List of Figures **vii**

List of Tables **ix**

1 Introduction **1**
1.1 Motivation & Background 1
1.2 Research Question 3
1.3 Aims and Objectives 3
1.4 Research Method 4
1.5 Contributions 4
1.6 Structure of Report 5

2 Literature Survey **6**
2.1 Indexing Architectures used by P2P Systems 6
2.1.1 Centralised Indexing
2.1.2 Decentralised Indexing - Unstructured Network
2.1.3 Distributed Indexing - Structured Network
2.2 Resource Discovery and Routing 12
2.2.1 Resource Discovery in Unstructured P2P Systems
2.2.2 Resource Discovery in Structured P2P Systems
2.2.3 Resource Discovery in Mobile Agent Systems
2.3 Critical Review 20
2.4 Agent Based System Development Frameworks 20
2.5 Qualitative Comparison of Mobile Agent Platforms 21
2.6 Summary of the Chapter 27

3 Design Features and Implementation **28**
3.1 The Proposed Multi-Agent System for Resource Discovery - Affinity 28
3.1.1 The Proposed Global System Architecture
3.1.2 Specification of Agents
3.2 The Proposed Mobile Agent Routing 32
3.2.1 Latent Semantic Indexing and Singular Value Decomposition (LSI-SVD) for Peer Clustering and Mobile Agent Routing
3.3 The Proposed Multi-Agent Collaboration for Resource Discovery 36
3.4 Implementation 39

3.4.1 Agents Communication Implementation
3.4.2 Feature Matrix - Frequency-Based Indexing
3.4.3 Implementation of Latent Semantic Indexing and Singular Value Decomposition
3.4.3.1 Index Maintenance
3.4.4 Similarity Function
3.4.4.1 Node Learning - Clustering
3.4.4.2 Node Searching and Ranking - Content Based Routing
3.4.4.3 Query Resolving
3.4.5 Mobile Agent - Reconnaissance Agent
3.5 Discussion 54
3.6 Summary of the Chapter 54

4 Experiments, Results, and Evaluation 56
4.1 Design of Experiments 56
4.1.1 Experiment Environment and Test Bed
4.2 Test 1 - Comparison to Flooding Technique 60
4.2.1 Experiment 1 - Response Time and Evaluation
4.2.2 Experiment 2 - Effectiveness of Search Technique and Evaluation
4.2.3 Observations
4.2.4 Critical Analysis
4.3 Test 2 - Comparison to Other Routing Techniques/Algorithms 65
4.3.1 Experiment 1 - Pair-Wise Document Similarity And Evaluation
4.3.2 Experiment 2 - Effectiveness of Search Technique And Evaluation
4.3.3 Experiment 3 - Effectiveness to Locate Resources and Evaluation
4.3.4 Experiment 4 - Degree of Relevance of Results and Evaluation
4.4 Discussion 71
4.5 Summary of the Chapter 72

5 Discussion, Conclusions, and Future Work 73
5.1 Discussions - Analysis of Other Research Works 73
5.2 Applications for Research Conducted 76
5.3 Conclusions 77
5.4 Future Work 78

References 80

A Similarity Measures and Weighting Functions 87

B Classes Realised - Affinity 88

C Program Listings - Affinity 89
C.1 Interface BootInf.java 89
C.2 Class Bootstrap.java 90
C.3 Class BootstrapServer.java 95
C.4 Class Extractor.java 97

C.5 Class ReportSim.java 99
C.6 Class Directory.java 101
C.7 Class MasterList.java 102
C.8 Class Repository.java 110
C.9 Class Node.java 112
C.10 Class InformationAgent.java 115
C.11 Class LocalAgent.java 130
C.12 Class LocalUI.java 146
C.13 Class MatchStore.java 148
C.14 Class InterfaceAgent.java 149
C.15 Class SearchGUI.java 154
C.16 Class ReconnaissanceAgent.java 160

List of Figures

2.1 A typical scenario of the centralised system. Source: Singh et al. (2009) . . . 7
2.2 Illustration of the flooding process. 8
2.3 Comparison of distributed indexing structures. (i) Gnutella-like local indexing. (ii) Global indexing. (iii) Hybrid indexing. (iv) Optimized hybrid indexing. a, b, and c are terms. X, Y, and Z are documents. Source: Tang & Dwarkadas (2004) 17
2.4 The SMART architecture Source: Wong et al. (2001) 22
2.5 The D'Agent architecture 23
2.6 The simplified version of Grasshopper Architecture. The basic services include MASIF and Core Services. MASIF includes agent creation, destruction, suspend, activate and location services and Core services include agent execution, transport, management, communication, security and naming. The enhanced services include APIs, GUIs and task control features. 24
2.7 The Aglets architecture Source: Schoeman & Cloete (2003) 25

3.1 Global architecture of the system 30
3.2 Peer clustering and overlay organisation achieved using *latent semantic indexing* 33
3.3 Interactions between multiple agents for resource discovery and realisation of an overlay network 37
3.4 The RA's interaction with InfA for issuing new peer GUID 39
3.5 Flow diagram for behaviour of the InfA and the LA upon arrival of the RA . 40
3.6 *ACLMessage* from the RA to the LA for search request. ACLMessage received by the LA from the RA using *MessageTemplate* and replying with *setContentObject* or in blocked state. 41
3.7 Creation of the RA in the method *onGUIEvent()* from class InterfaceAgent . 43
3.8 *ReceiveMessageRecon* class showing *blocked state* of when reply received is *null* and the *MessageTemplate* for receiving messages from the RA 43
3.9 Method *getKeywords()* for getting keywords and their frequencies and holding in data structure TreeMap 45
3.10 Realisation of Singular Value Decomposition from frequency based keyword-resource or keyword node matrix 46
3.11 Index Maintenance task performed recursively by Information Agent 47
3.12 Directory data structure used by *BootStrapServer* to pass clustered nodes result to *InformationAgent* 49

3.13 Realisation of node searching and ranking 51

4.1 Frequency distribution of response time analysis - Gnutella vs. Affinity 61
4.2 Query successful vs. unsuccessful - Affinity method 62
4.3 Query successful vs. unsuccessful - flooding method 62
4.4 Division of packets for Gnutella 63
4.5 Gnutella packets analysed using Wireshark 64
4.6 Precision and recall results comparing LSI to TF-IDF indexing model 65
4.7 Pair-wise document similarity TF-IDF Jaccard vs. LSI Cosine 66
4.8 Number of times a document appears for 30 queries Jaccard similarity vs. Cosine similarity 68
4.9 Number of documents found for 30 separate queries on corpus of documents . 69
4.10 Similarity score distribution TF-IDF Jaccard vs. LSI Cosine 70

B.1 Classes for resource discovery system - Affinity 88

List of Tables

2.1 A classification of P2P routing infrastructures in terms of network structures Source:Androutsellis-Theotokis & Spinellis (2004) 11
2.2 Summary of infrastructure for routing and resource discovery location Source: Androutsellis-Theotokis & Spinellis (2004) 12
2.3 Comparison of features of routing algorithms Source: Prakash (2006) 13
2.4 Qualitative Comparison Among Mobile Agent Platforms Source: Trillo et al. (2007) 25

4.1 Gnutella flooding peers test bed 58
4.2 MAS test bed 59
4.3 Keywords used for sharing resource on each node 59

A.1 Similarity measures 87
A.2 Local and global weighting functions 87

Chapter 1

Introduction

1.1 Motivation & Background

The volume of data published online per year is estimated to be of an order of approximately one terabyte and it is expected to grow exponentially. The solution offered to users is in form of a search engine, for instance Google. However, these solutions suffer from requirement of maintaining a large centralised database about online published information. In order to support the solution and also offer scalability, they require a large and highly costly hierarchical infrastructure. Moreover, any newly published information requires time for indexing and is often not indexed for weeks. Similarly, any information that has either been removed or ceases to exist also results in dead-links for users because of delayed indexing.

These reasons call for a requirement of a scalable infrastructure that is capable of indexing, routing and searching rich published content.

As opposed to centralised form for indexing offered by search engines, peer-to-peer (P2P) networks offer solution for resource discovery by making the task of hosting distributed. The P2P networks consist of a number of decentralised nodes sharing their resources on an overlay network. Here the resources mean services/files that are hosted on nodes of the network. P2P systems offer low-cost sharing of information and with high autonomy. P2P networks offer characteristics such as high availability, low cost and ease of deployment, data freshness and good scalability Yingwu Zhu (2005). Because of following features P2P networks become ideal choice as opposed to centralised solutions offered by search engines.

1. Autonomy: Autonomy of nodes allows them to join/leave at any time, control their data with respect to other nodes i.e. shared resources are published and indexed

immediately.

2. Query expressiveness: Key-lookup, key-word search

3. Efficiency: Efficient use of bandwidth, computing power and storage

However, the process of discovery of this shared information is not very efficient due to poor search performance and unavailability of heuristics Tran & Schonwalden (2008).

A classical client and server based centralised solution to a location of resource is offered by Napster Napster (2003); Aberer et al. (2004). In this approach, a client connects to central server - that is responsible for indexing resources and their location. Upon query about resource location from any other client, the central server issues the IP address of the client where resource is located. This solution cripples autonomy of a client due to centralised sever, as in case of server failure, clients cannot locate resources.

Another approach to resource location is offered by Gnutella, where the decentralised peers communicate to other peers when the resource location query is issued by user Chawathe et al. (2003); Forum (2002). This solution offer high degree of autonomy as peers can join or leave the overlay network without affecting rest of the network. When locating a resource, peer floods the user query on the overlay network usually with time-to-live constraint in order to query other peers about required resource. The inefficiency in this approach attributed to three facts:

1. The overlay network is created randomly as there is not structure associated with it

2. The queries for a resource location are forwarded "blindly" from one peer to another peer using technique called flooding due to which there is unnecessary quantity of message on the network

3. Saturation as number of nodes increase.

A more "rigid" approach is taken by a structured overlay that is based on hash functions supports key-based routing such that resource identifiers are mapped to the peer identifier address space and a resource request is routed to the nearest peer in the peer address space Ratnasamy et al. (2001); Rowstron & Druschel (2001); Stoica et al. (2001); Zhao et al. (2001). Although such systems are better than unstructured overlay from performance point of view as some heuristics are available for locating a resource (only where the search keys are known exactly), but they are not as effective for approximate keywords, or text based resource location Yingwu Zhu (2005); Tran & Schonwalden (2008).

1.2 Research Question

The author formulates the overall research question as following:

> Can the process of resource discovery be improved for P2P systems in order to increase search performance such that the higher number of relevant results can be achieved and keep the possibility of saturation of network low that is a resultant of routing on P2P network?

It is understood from literature that saturation can be decreased and hence improved, if informed search is performed that is resultant of availability of heuristics and that the search performance or recall can be increased if an efficient indexing technique and similarity functions are available. This results into breaking down of general research questions into:

1. Can global heuristics be distributed to nodes on the overlay network efficiently with constraint on communication overhead?
2. How can search performance or recall and routing be improved dynamically?
3. What type of characteristics and representation must the resource have in order to be indexed and further be used for representing the node?

1.3 Aims and Objectives

The main aim of this research is to design and implement a novel routing and searching technique based on Latent Semantic Indexing (LSI) and mobile agent technology in a P2P network created by collaboration of multiple agents.

The main objective is to design and implement a resource discovery system that uses mobile agent technology for discovering and selecting nodes and for routing the mobile agent through overlay network based on content of query with purpose of minimising response time, reducing possible delays, maximising network performance by reducing the possibility of saturation and maximising the recall by providing relevant results. Furthermore, it is endeavoured that this system will offer improvement over attributes of performance and scalability.

1.4 Research Method

The field of using mobile agents on P2P networks using LSI is fairly new and most previous attempts have been made using term-based matching techniques, flooding, Distributed Hash Table (DHT) on unstructured or structured networks. It can be concluded without a doubt that this field is growing rapidly and is not very well understood at this stage. The author concludes that the most suitable research method for this research project is experimental research where the evaluation of various experiments conducted will be compared both quantitatively and qualitatively to other related works in this field.

The author endeavours to conduct experiments in order to answer the research question and prove the postulated hypothesis that mobile agent can be used effectively for efficient resource discovery when powered by content-based routing to create network heuristics and discover the topology of overlay network for the purpose of maximising search performance, minimising response time, have higher inter-cluster links and higher degree of relevance of the obtained search results.

1.5 Contributions

The purpose of this research is to offer the multi-agent system (MAS) and the resource discovery based on content based routing of mobile agent that overcomes the disadvantages of structured overlay i.e. be able to locate resources even when the keys are unknown, approximate, or text based multiple keys and also offer the flexibility characteristic of autonomous unstructured overlay but by reducing number of message on the network and control or remove unnecessary flooding.

Through this research work, the author proposes the following:

- *Autonomous MAS System*: a flexible multi-agent based approach for dynamic organisation of P2P network that is based on the similarity of content shared by peers. The similarity of content between two or more peers is translated into similarity between peers or a cluster of peers sharing similar content.
- *Deterministic Content Driven Routing*: the resource location mechanism that uses semantic similarity between content shared by peers and search keywords to deterministically route a mobile agent called the reconnaissance agent (RA) to peers that host content that is similar to a user query.

- *Use LSI Based Indexing and Query Matching*: the use of LSI and cosine similarity by RA to find relevance of resource(s) hosted by peer as a best match for a user query (where the user query can be text based or an approximate query).

The author demonstrates that this method improves the resource discovery performance i.e. finding a relevant resource(s) with lower response time and hence reducing search load.

1.6 Structure of Report

The rest of the report is structured as follows.

Chapter 2 surveys the current literature and draws lessons to propose the capabilities that a resource discovery system should obtain. In doing so, chapter 2 also collates a large amount of research work relevant to field of study and also discusses the architecture and platforms for development of MAS.

Chapter 3 describes the design features of proposed MAS based resource discovery system, node clustering based on semantic similarity of content hosted by nodes and RA routing, and the multi-agent collaboration for resource discovery. Furthermore, it describes the implementation done using Java remote method invocation (RMI) and Java Agent Development Framework (JADE) Bellifemine et al. (2007).

Chapter 4 is dedicated for experimentation where the effectiveness of proposed resource discovery algorithm and resource locating algorithm is compared against flooding (Gnutella) in terms of response time and search load. Furthermore, proposed node clustering algorithm for routing the RA on the overlay network, messages on network and relevance of results obtained due to user invoked query is compared to contemporary research work done by other researchers in field of using mobile agents for resource discovery.

Chapter 5 is dedicated for discussions for assembling and comparing our concepts to other related works in the field of resource discovery and further provide list the conclusions and also presents the future work that can be conducted in this field.

Ending sections of report provide references, appendices, and program listings.

Chapter 2

Literature Survey

This chapter provides a detailed survey of current literature and draw lessons to propose the capabilities that a resource discovery system should obtain. It also collates a large amount of research work relevant to field of study and also discusses the architecture and platforms for development of MAS.

As described by Singh et al., there are diverse set of solutions that are available for resource discovery. These solutions are characterised through the routing strategy and resource searching strategy that is applied by them Karnstedt et al. (2004); Singh et al. (2009). The author have categorised and reviewed the resource searching techniques used by unstructured and structured P2P systems by initially discussing architectures. The author also presents most current search techniques that are being introduced to the resource discovery domain.

2.1 Indexing Architectures used by P2P Systems

2.1.1 Centralised Indexing

The first most popular P2P Network was Napster, which used Central Indexing Server for storing the locations of the resources Aberer et al. (2004). Using this network Napster client's in the network can communicate with the other Napster clients. In Napster a dedicated central server maintains an index of the files shared by the active peers on the network. Each peer in the network maintains a constant connection to one of the central server through which the query for file location is sent. When a central indexing server receives the query for a file location it cooperates to process the query and returns the corresponding matching file locations to the peer making the query. After the peer making

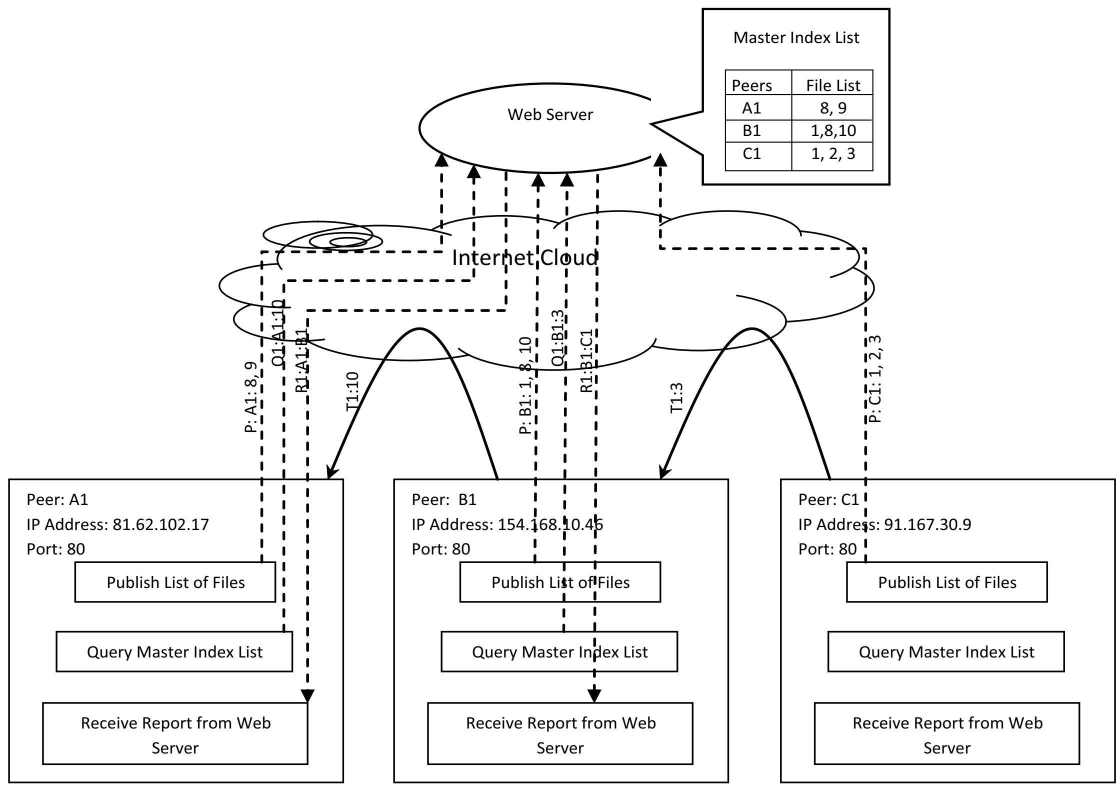

FIGURE 2.1: A typical scenario of the centralised system. Source: Singh et al. (2009)

query receives response from the indexing server about the list of locations of the resource, the peer can now make direct communication with the peers having the resources and initiate the transfer of the resource. Besides maintaining the list of resources in the network, the indexing server also keeps track of each peer that is active or monitors the state of the peer like keeping track of the information of the peer for instance the duration the peer has been active or the connection speed the peer is at Napster (2003). In Figure 2.1, the peer A1, peer B1 and peer C1 are sharing resources 8, 9; 1,8,10 and 1, 2, 3 respectively. The central server, “Napster.com” that keeps the index of all resources shared by the peers. The central server is queried by the peer A1 and peer B2 for the resource 10 and resource 3 respectively. The central server replies by providing the IP address of the resource providers to each of the peers. The direct connection is established between two peers for downloading of the resource.

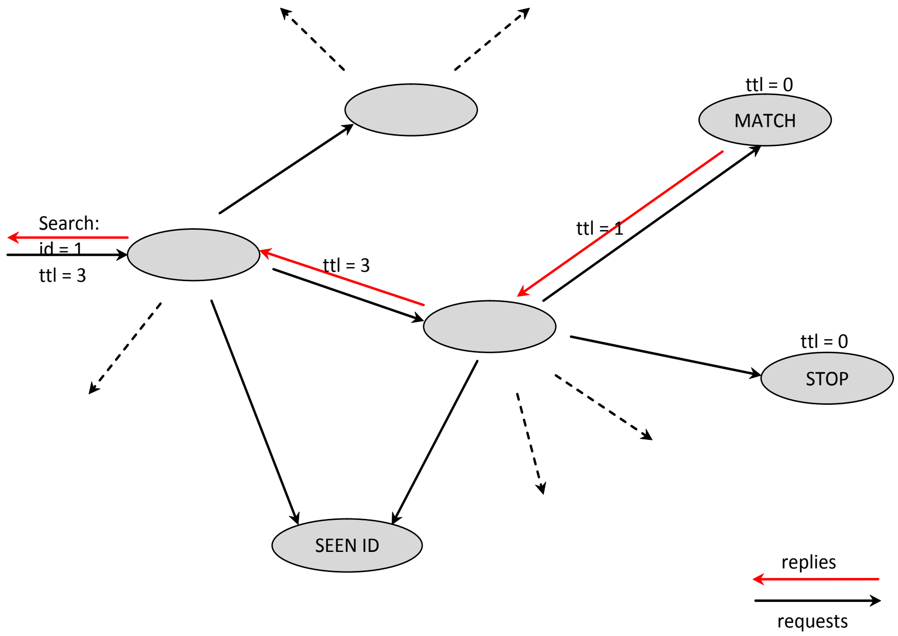

FIGURE 2.2: Illustration of the flooding process.

2.1.2 Decentralised Indexing - Unstructured Network

An unstructured overlay like GNUTELLA is organised into random graph topology where there is no specific topology that the overlay network follows and it uses flooding or random walks to discover resource in the network. This overlay is constructed easily when a node wants to join the network. During the resource discovery each node visited will evaluate the query locally on its data store. Before starting to exchange messages between the nodes, a Gnutella node connects itself to the network by connecting with another *well-known* node on the network. Once the connection is established, the addresses of one or more host will be supplied as the node joins the network. The listening node is advertised by Pong messages. When another node is located on the network TCP/IP connection is established and a handshake sequence is initiated. In Figure 2.2, it is observed that when the search begins from id=1, it is broadcasted to all the peers that are connected to the node with TTL=3. The TTL is decreased by 1 after every hop until TTL drops to zero. If the matching resource is found it is responding through the reverse path until it reaches to the originating node id=1. Details of Gnutella resource discovery protocol are discussed in Section 2.2.1.

2.1.3 Distributed Indexing - Structured Network

A structured overlay and DHT based systems like Chord, Pastry, Content Addressable Network (CAN), and Tapestry are the improvement on unstructured overlay to improve the performance of resource discovery Stoica et al. (2001); Rowstron & Druschel (2001); Ratnasamy et al. (2001); Zhao et al. (2004). It ensures that any node can efficiently route a search to some peer that has the desired file even in the rare availability Killmeyer (2006). The nodes in the network impose constraints on the topology as well as on the data placement to provide with efficient search mechanism and resource discovery. In all the DHT systems mentioned above files are associated with a key and each node in the network is responsible for storing list of resources hence having list of keys. The first and foremost operation in the DHT system is the look up for the key as lookup(key) which returns a location of the resource or the key and hence IP address.

Chord

Till date there are many load balancing approaches, Chord was the first to propose the concept of virtual servers and hence address the load balancing by having each node simulate a logarithmic number of virtual servers Zhu & Hu (2005). Using Chord, only $\log(N)$ messages are required to find the resource in the Chord Network where N being the number of active nodes in the network. Chord allows distributed nodes to agree on a single Chord node as a rendezvous point for a given key without any central coordination Project (2010). Chord algorithm does not particularly specify any means for storage of the resource; this is done by DHash which is built on top of Chord and also handles storage of data blocks on the active nodes reliably Project (2010). This is achieved using techniques like replication and erasure coding. The logical application interface for DHT based systems is defined as: $Key = put(data)$ and $Data = get(key)$ Project (2010).

Pastry

Pastry is completely decentralized, scalable and self organizing network which dynamically adapts to the addition or removal of nodes Guvnec & Urdaneta (2010). Each node in Pastry Network has unique and random identifier called NodeId in a circular 128-bit identifier space. With a message and a numeric 128-bit key, a node can route the message to a node with NodeId which is numerically close to the key within the live Pastry Network Rowstron & Druschel (2001). This results in first order balancing of the storage requirements and query among the nodes in the Pastry network and also does not require global co-ordination Rowstron & Druschel (2001).

Routing in Pastry for a given message it checks the following conditions Guvnec & Urdaneta (2010):

- If it falls within the NodeId's leafset then the message is directly forwarded to it.
- Else, the message is forwarded to a node that shares the most common prefix with the key using the routing table
- Else if the routing table is empty or the node is unreachable, then message is forwarded to node that is numerically close to the key.

If given N as number of live nodes in the overlay Pastry Network then expected number of forwarding steps $\mathcal{O}(\log N)$ and size of routing table for each node $\mathcal{O}(\log N)$ Rowstron & Druschel (2001).

CAN

CAN is also a distributed system which is DHT based that maps keys to values on big scale network like internet. As discussed above CANs basic idea is to build a hash table and the basic operations performed are insertion, lookup and deletion of the key, value pairs. In the CAN network each node stores a chunk (also called zone) of the total hash table. Moreover it stores smaller amount of information of adjacent zones Ratnasamy et al. (2001).

In CAN the network is formed in a tree like structure where each node is associated to one, at the parent level and to a group at a child level. When a query is made, it travels from the top most level going down through the network until the resource is discovered or until the last leaf is reached Guvnec & Urdaneta (2010). The architecture of the CAN is a virtual multi dimensional can be viewed as Cartesian coordinate space. CAN design centres around a virtual d-dimensional Cartesian coordinate space on a d-torus which is independent of the physical location and physical connectivity of the nodes Ratnasamy et al. (2001). The overall Cartesian coordinate space is dynamically partitioned among all the nodes such that each node belongs to one distinct zone with in the entire space Ratnasamy et al. (2001). To route a query, node maintains a routing table which holds the IP locations as well as the virtual coordinate zone of each of its neighbour. Using the co ordinates the message is routed towards destination.

CAN construction take place in three steps:-

1. A joining node must find a node which is already on the CAN network
2. Using the CAN routing mechanism, it must find a node whose zone will be split
3. Lastly, the neighbours of split zone are informed.

Tapestry

Tapestry is another P2P structured overlay network which provides high performance, scalable as well as location independent routing of the messages. It uses adaptive algorithm with soft state to maintain fault tolerance with regards to changing node membership and network faults. Tapestry provides decentralized object location and routing (DOLR), the DOLR interface provides routing of messages to end points like nodes or object replicas Zhao et al. (2004). Each Tapestry node is assigned a unique id and more than one node can be hosted by a single physical host. Tapestry utilizes identifier space of 160 bit values with a 40 digit key. The efficiency of the Tapestry increases with the increase in the network size. Moreover to allow multiple applications every message contains an application specific identifier which helps the node to select a process or delivery of message to a specific port Zhao et al. (2004).

Table 2.1 shows the classification of P2P routing infrastructures in terms of their network structure, with typical examples. Table 2.2 summarises infrastructure for routing and resource discovery location.

	Centralisation		
	Hybrid	Partial	None
Unstructured	Napster	Kazaa, Edutella	Gnutella
Structured			Chord, CAN, Tapestry, Pastry

TABLE 2.1: A classification of P2P routing infrastructures in terms of network structures Source:Androutsellis-Theotokis & Spinellis (2004)

P2P Infrastructure	Description for Routing and Location
Flooding	Infrastructure that provides functionality for searching "blindly" on overlay networks.
Chord	A scalable peer-to-peer lookup service. Given a key it maps the key to a node.
CAN	Scalable content addressable network. A distributed infrastructure that provides hash-table functionality for mapping file names to their locations.
Pastry	Infrastructure for fault-tolerant wide-area location and routing.
Tapestry	Infrastructure for fault-tolerant wide area location and routing.

TABLE 2.2: Summary of infrastructure for routing and resource discovery location Source: Androutsellis-Theotokis & Spinellis (2004)

2.2 Resource Discovery and Routing

Table 2.3 compares various features of routing algorithms used in P2P systems.

2.2.1 Resource Discovery in Unstructured P2P Systems

In unstructured P2P systems for instance Gnutella, various nodes(peers) are organised into a random graph where the edges of the graph are the links between various nodes this constructing an overlay network Chawathe et al. (2003); Forum (2002). Flooding technique is used for routing a query through the overlay network. Upon query, the visited node compares the query against its shared resources and is then requested to forward the query to its neighbours. This system of resource discovery is highly robust and offers vast improvement on factor of scalability as compared to Napster or other centralised search systems but suffers from an expensive cost of saturation of overlay network due to large bandwidth consumption Chawathe et al. (2003); Forum (2002); Aberer et al. (2004); Napster (2003). Various techniques have been introduced to improve the efficiency of this system that includes random walks, informed searches, and node grouping Bawa et al. (2003); Zhu & Hu

Feature	Conventional Flooding	Random Walks	DHT	Range Query
Infrastructure	Performed on unstructured P2P networks	Performed on unstructured P2P networks	Performed on structured (DHT) P2P networks	Performed on structured (DHT) P2P networks
Scope	Works same with any network	Works best with multiple queries and peer clustering	Works same with all the DHTs	Works best when semantic proximity of keys is maintained
Search Complexity	On average search is done in k*N time (k = average degree of nodes, N = total number of nodes)	On average search is done in log time	On average search is done in log time	On average search is done in log time
Relevance and Results	Returns single result	Returns single result	Returns single result	Returns a set of results
Cost Associated with Resource Discovery	Very wasteful of resources, as every peer processes each query	Less taxing on resources	Not too taxing on resources	Min-max algorithm uses resources wisely
Response Time (Routing and Searching)	Is not very fast, as every peer processes each query	Result are reasonably fast	Routing is very fast	Shower algorithm is very fast

TABLE 2.3: Comparison of features of routing algorithms Source: Prakash (2006)

(2006); Lv et al. (2002); Crespo & Garcia-Molina (2002).

Random walks were introduced to improve the issue of saturation by introduction of techniques *time-to-live* (TTL) and checking Lv et al. (2002). Like flooding, random walks, is uninformed search technique where the query is randomly forwarded to nodes. As an answer to saturation of the overlay network, the total number of nodes to be visited is defined using TTL. Also, *checking* technique is used where before forwarding to next node, the query originator is "checked with". These techniques of controlled flooding refined resource searching mechanism but suffered from lack of results due to restrictions imposed by TTL.

To increase the effectiveness of search mechanism, informed searches were introduced that offered improvement in performance by using information on nodes and their resources Lopes & Botelho (2008). This information is collected as part of previous queries. Crespo et. al. introduced the technique *routing indices* (RI) for informed searches, where queries are routed to nodes that were more likely to provide a resource Crespo & Garcia-Molina (2002). In this technique uses distributed-index mechanism that maintains indices on each node. Given a query, the RI data structure returns a list of ranked nodes for forwarding a query. In informed searches, propagating a query to nodes where there is likeliness of discovering a resource help reduce the network load because of less flooding.

Other resource location techniques such as *SETS* and *ESS*, are based on a concept of grouping content to organise nodes Bawa et al. (2003); Zhu & Hu (2004). The search in SETS is based on topic-segmentation of overlay network. In other words, SETS partitions nodes into topic segments such as nodes with similar content belong to same segment Zhu & Hu (2006). SETS suffer from single point failure and hence has performance bottleneck Zhu & Hu (2006). ESS is based on information retrieval algorithms to perform resource discovery on Gnutella-like P2P systems. As in SETS, nodes with similar content are segmented into same semantic group Zhu & Hu (2004). The concept used by ESS is to place indexes of semantically close files into same nodes with high probability of exploiting information retrieval algorithms and locality sensitive hashing Zhu & Hu (2007).

A multiple keyword based searching technique called *local indexing* is used for locating resource using multiple keywords Tang & Dwarkadas (2004). As seen in Figure 2.3(i), the record of terms contained in each resource is stored on that particular node. Upon query, the search keywords are forwarded to each node using flooding technique, where they are compared for relevance. This technique is effective for getting better search results but suffers from classical saturation factor on overlay network.

2.2.2 Resource Discovery in Structured P2P Systems

Structured P2P systems have been proposed to provide a more scalable solution as compared to first generation unscalable unstructured P2P systems. In structured systems, a node is associated with keys and their values. When a query is presented it is changed into the search for the key. The hash table on the peer is used pass the query forward to other peer whose address is numerically closer to requested key. The examples of structured systems are Chord, and CAN Ratnasamy et al. (2001); Stoica et al. (2001). In hybrid systems for instance Pastry, the routing structure is comparatively more fluid as compared to Chord as the routing table can suggest the routing of the query to any node that is part of the defined subspace Talai et al. (2006); Rowstron & Druschel (2001).

Structured systems perform better than unstructured systems with respect to scalability, as DHT has many advantages, such as scalability, load balancing, logarithmic hop routing, fault tolerance, and self organising nature Singh et al. (2009). Although self-organising works as the advantage but as each peer must periodically update all its neighbours and hence results in increased traffic Mastroianni et al. (2005). When the nodes leave or join the network the updated index need to be redistributed and hence the tables need to be restructured. This is not the case in unstructured systems as node can leave or join the network without sending stabilisation message. Unstructured systems have provided many strategies for reducing traffic like dynamic querying, routing indices, and super-peers architectures Chawathe et al. (2003); Karnstedt et al. (2004). Structured systems have advantage over unstructured systems as these systems provide ability to route the queries in very small number of hops. DHT-based systems are known for exact-match lookups, given a query both Chord and Pastry resolve the queries in $\mathcal{O}(\log(n))$, while CAN requires $\mathcal{O}(n^{\frac{1}{d}})$ steps, where n is number of nodes and d is number of dimensions in CAN Stoica et al. (2001); Ratnasamy et al. (2001). As the peers and the resources are based on the hash function – key generated by the hash function is very specific Stoica et al. (2001); Ratnasamy et al. (2001). As the queries may not be exact, it may be difficult to find the resource in the structured network Mastroianni et al. (2005); Singh et al. (2009).

However, in *keyword-search* the queries do not have to be exact and can comprise of multiple-keywords. The information retrieved in such scenario consists of a set of resources that match the criteria given as a query. The proposed system that support keyword-search on top of DHT-based structured P2P system are categorised by their indexing technique viz. *global indexing* [Li et al. (2003); Reynolds & Vahdat (2003); Casey & Zhou (2009); Tang & Dwarkadas (2004)], and *hybrid indexing/optimised-hybrid indexing* [Zaharia & Keshav (2008); Tang & Dwarkadas (2004); Chen et al. (2008)].

In *global indexing* as seen in Figure 2.3(ii), the inverted list record is maintained on every node - information about nodes that contain a particular term. Upon query that contains multiple keywords, the query is routed to node containing that keyword. Then the inverted lists are intersected to find resource that contains the requested keywords. This largely reduces the number of nodes that need to be visited, however large amount of communication is introduced during intersecting phase. Moreover, communication cost grows with increase in length of inverted list Tang & Dwarkadas (2004); Zhu & Hu (2007).

In *hybrid indexing* as seen in Figure 2.3(iii), each node holds the complete inverted list of terms describing the resources on that node and also the inverted list of terms that are forwarding terms for resources shared on this node. Given a multiple keyword based query, the query is routed to node containing the search keywords. Then, this node performs a local search without connecting to other nodes about list of forwarding nodes by querying the inverted list of each found resource on this node. The efficiency of this type of indexing is higher than that of global indexing but suffers from increased cost of publishing term data Zhu & Hu (2006).

In *optimised hybrid indexing* (See Figure 2.3(iv)), the terms that describe a resource is published under resource's top terms (terms that are central to a resource) Tang & Dwarkadas (2004). Clearly, the search may be degraded because of limiting the publishing of keywords under resource's top terms Zhu & Hu (2006).

Another effective way for resource discovery process is to establish *semantic links* between the nodes that are based on node properties which are described by the resources shared by those nodes Sun et al. (2006); Kang et al. (2007); Crespo & Garcia-Molina (2004); Tang et al. (2003); Arabshian et al. (2009). In Kang et al., the semantics information is used for searching resources in a scalable manner Kang et al. (2007). A. Crespo suggests the semantic overlay network (SON) where the peers are organised based on logical similarity between the content Crespo & Garcia-Molina (2004). Semantic information can be used to create P2P networks that are more organised than unstructured overlay and are capable of handling multiple keys for finding resource on network unlike structured overlays. Locality

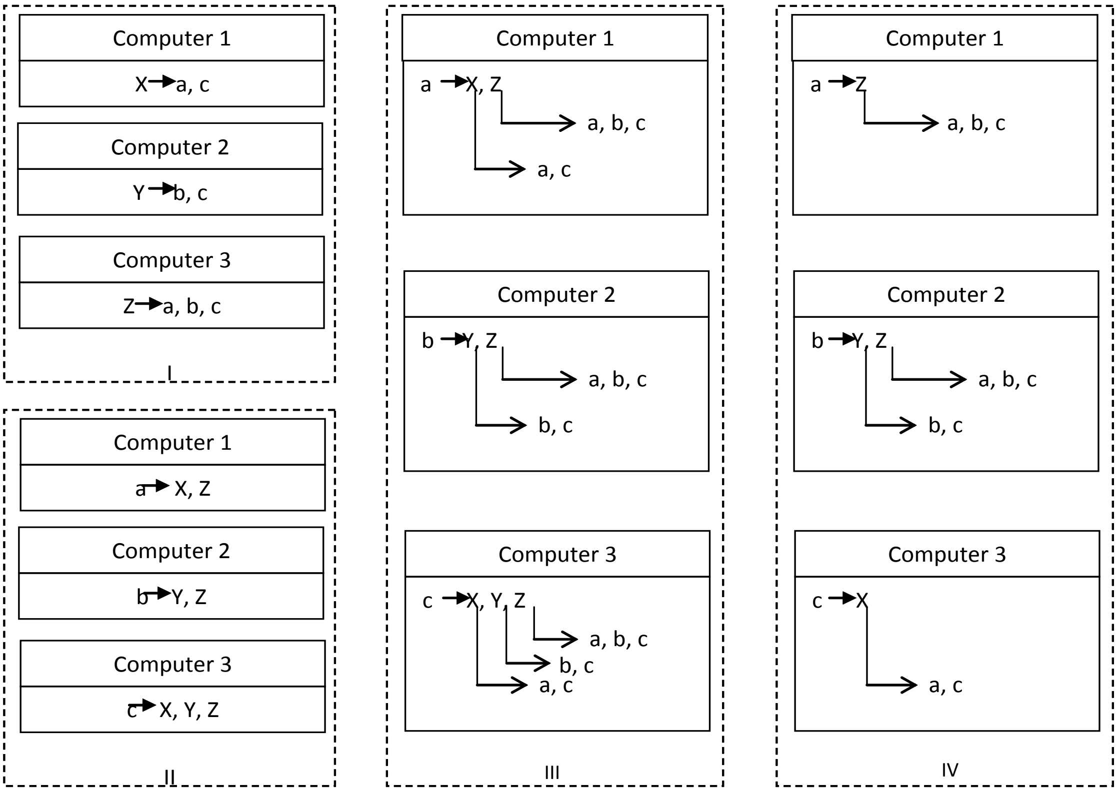

FIGURE 2.3: Comparison of distributed indexing structures. (i) Gnutella-like local indexing. (ii) Global indexing. (iii) Hybrid indexing. (iv) Optimized hybrid indexing. a, b, and c are terms. X, Y, and Z are documents. Source: Tang & Dwarkadas (2004)

awareness is another version where the peers are organised based on matching tags that are used to describe a resource Sun et al. (2006). pSearch introduces the concept of semantic overlay on top of a DHT based structured P2P system Tang et al. (2003). In this overlay, the resources are organised based on their semantic vectors (such as distance). pSearch proposed to integrate semantic storage and retrieval capabilities into CAN, where resource index is stored by using its vector representation as coordinates Zhu et al. (2003). GloServ uses a keyword-based search on a hierarchical hybrid P2P network to build semantic overlay between nodes that operate in the same domain Lopes & Botelho (2008); Arabshian et al. (2009). Even though this attempt at creating semantic links between nodes and resources may help improve the resource discovery, but no test results have been published yet by the authors.

Both structured and unstructured systems heavily rely on stationary software modules. These modules keep track of all resource discoveries. They use the host computer resources and can potentially drain the local resources and may cause failure of host computer. Backbone of both approaches is P2P communication. P2P communication blurs the distinction between client and server computers. This can potentially saturate the network. Unstructured resource discovery has a linear connection between computers where each computer knows the ping computer. Failure of any computer in the chain results to loss of all down stream resources.

2.2.3 Resource Discovery in Mobile Agent Systems

As an alternative to stationary software modules, multi-agent systems offer following merits that make mobile agents in particular suitable for resource discovery in P2P systems Dunne (2001):

- Asynchronous: After a mobile agent is dispatched, there is no need for the creator peer to keep track of mobile agent. The thread can be completely released. Theoretically speaking, the creator peer does not even need to remain connected to a network. A mobile agent will perform the given tasks completely in parallel with the creator peer as a separate thread. After all of the tasks have been fulfilled, mobile agent will return to the creator peer (when it is connected to the network).

- Autonomous: Mobile agents can compute its itinerary as it progresses through the network. It is able to choose the next site according to conditions it has learnt about, and history of visited peers and current peer. Mobile agents may also visit peers that were unknown when it was originally dispatched, which in particular suits network based resource discovery.

- Compatibility: Agent based systems can be combined with successful features from other resource discovery systems.
- Bandwidth Consumption: The mobile agents for resource discovery require lesser bandwidth. As opposed to the multiple interactions between peers, mobile agent packs these interactions and sends them as discrete piece of traffic. Also mobile agents are much smaller in size and grow dynamically as they accommodate more data. In structured or unstructured systems, the communication is synchronous which is not the case with mobile agent which can encapsulate its state and carry on the execution on the different node asynchronously Bellifemine et al. (2007).

Dasgupta et al and Kambayashi et al introduced multi-agent systems (MAS) for resource discovery Dasgupta (2003); Kambayashi & Harada (2009). Both systems are inspired from ant communities for development of their P2P system. They use *Anthill* MAS that emulates the resource coordination behaviour as observed in ants Babaoglu et al. (2002); Babaoglu & Jelasity (2008); Yang et al. (2007). In this MAS P2P system resources are known as nests and user request to locate resources is carried out by ants. Upon query, the ants visit various nests on overlay network. Ants restrict from communicating to each other but leave information about the service they are implementing in the resource manager found at each nest site. The behaviour has analogy to pheromones that has advantage of allowing network to self-organise over a period of time Lopes & Botelho (2008). Ants greatly improve upon the flooding issue raised in unstructured P2P systems as only one ant visits the nest at one time. The next nest chosen for ant to visit is either deterministic or random, which means that search performance may be slow. This is observed in [Dasgupta (2003) and Kambayashi & Harada (2009)], where overlay network becomes more "knowledgeable" over a period of time. To improve upon this disadvantage, Kambayashi et al build their P2P system on top of structured P2P system called Chord Stoica et al. (2001); Kambayashi & Harada (2009). Mobile agents (ants) in their system may use <key, value> map to find resource in cases when deterministic path cannot be calculated. Kambayashi et al also use indexing (TF-IDF) to calculate logical distance between two nests based on correlation between keywords shared between nodes. The correlation is calculated using primitive form of Jaccard similarity.

2.3 Critical Review

For the research work, the author understands from the literature survey that semantic links between the nodes is useful for resource location and for node coordination - to be used for deterministic routing of the query which is also one of objectives of this research work. The author further understands that MAS and mobile agents offer nodes a greater degree of autonomy as they can migrate to new nodes based on information provided by visited nodes and hence offer relevant results to user. It is further understood that search load can be reduced by reducing number of messages or number of hops made by mobile agent during migrations from one node to another. The author aims to exploit heterogeneity of resources hosted by nodes on overlay network to locate resources in minimum number of hops i.e. drive/route the mobile agent on overlay network based on the content hosted by nodes.

2.4 Agent Based System Development Frameworks

This section provides review of the different mobile agent platforms and justifies the choice of the mobile agent platform - JADE Schoeman & Cloete (2003); Trillo et al. (2007):

Mobile Agent System Interoperability Facility (MASIF) standard OMG organization defined a standard named as Mobile Agent Framework (MAF) (later on changed to MASIF), which is aimed at promoting the interoperability of JAVA based mobile agent systems developed by different vendors Zhong & Liu (2003). MASIF presents a set of definitions and interoperable interfaces for mobile agent systems. The *MAFAgentSystem* interface and the *MAFFinder* interface are the two primary ones which are designed towards the following interoperability concerns Schoeman & Cloete (2003):

1. Management of agent, including creation, suspension, resumption and termination;
2. Commonly accepted mobility infrastructure that enabling the communications between different mobile agent systems and the transport of mobile agents;
3. A standardised syntax and semantics for naming services; and
4. A standardised location syntax for finding agents.

MASIF also excludes the following important architectural components in its standardisation attempts Schoeman & Cloete (2003):

(1) It only addresses interoperability between agent systems written in Java, thus brings the obstacle of the interoperability between non-Java based systems and MASIF compliant systems;

(2) It does not address local agent operations such as agent interpretation and execution;

(3) Some conventional issues of inter-agent communication are excluded Milojicic et al. (1998).

Foundation for Intelligent Physical Agent (FIPA) is the standards organisation for agents and multi-agent systems who promotes agent-based technology and the interoperability of its standards with other technologies Vieira (2001). A collection of specifications have been provided, which are intended to promote the interoperation of heterogeneous agents and the services they represents. However these specifications are focussed on agent communication languages, agent management, message transport and the support for the use of ontologies in general.

2.5 Qualitative Comparison of Mobile Agent Platforms

The following are the most popular mobile agent platforms Schoeman & Cloete (2003):

JADE Specification of FIPA are implemented by Java Agent Development Framework (JADE) that provides Application Programming Interfaces (API) for Java based implementation of multi-agent systems Bellifemine et al. (2007). The agent platform can be distributed on multiple hosts. Each platform only hosts one application and hence only one Java Virtual Machine (JVM). JVM can allow several agents to execute concurrently on the same host. The *Agent* interface is the primary interface that concerns is implemented for all types of agents. JADE implements the complete Agent Management specifications suggested by FIPA including services such as Agent Management System (AMS), Directory Facilitator (DF), Message Transport Service (MTS), and Agent Communication Channel (ACC). In addition JADE has implemented Agent Communication stack, ranging from FIPA-ACL for message structure and FIPA-SL for message content and other FIPA interaction and transport protocols Bellifemine et al. (2007).

The main drawback is that currently inter-platform mobility service is being developed and not available to researchers. Also, there are no proxies and agent searches the current location of its target by querying the AMS.

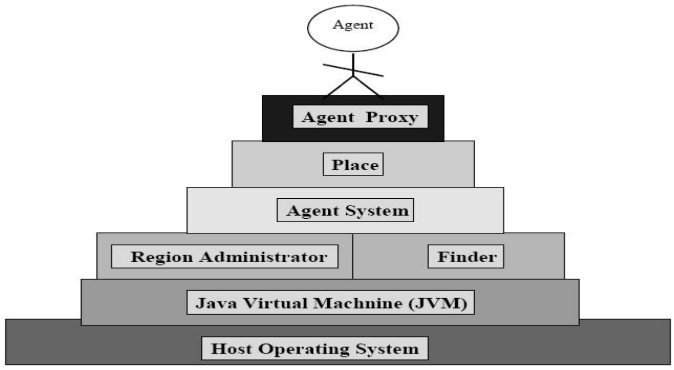

FIGURE 2.4: The SMART architecture Source: Wong et al. (2001)

Voyager is a commercial mobile agent platform supporting dynamic aggregation feature. The basic idea behind Voyager and dynamic aggregation feature is to reuse existing Java classes and make objects of such classes mobile by means of incorporating those objects as its attachments (known as facets) and move from one site to another hence moving those objects with itself. The objects will retain their internal state upon moving from one host to another Wong et al. (2001). The main focus is on the management of remote communications of traditional Common Object Request Broker Architecture (CORBA) and RMI protocols. It also offers dynamic generation of CORBA proxies and mobile agents. Agents communicate via RMI using proxies.

The main drawback of Voyager is that it is commercial product and is not freely available.

Scalable Mobile and Reliable Technology (SMART) SMART Wong et al. (2001) is a MASIF specification compliant client-server based mobile agent platform. As Figure 2.4 shows, there are four main components in smart architecture Wong et al. (2001): Region administrator, which uses a finder model to provide naming services to the region administrator and also to the agent system; Agent system, enables mobile agents to create, migrate and destroy themselves; Place, forms the execution environment; and Agent proxy, provides the mobile agent API for applications written in SMART.

The main disadvantage of SMART is that it does not support agent communication as described in MASIF standard. Also, it does not provide good security mechanisms.

D'Agents (Robert S. Gray) is a general purpose mobile agent system which was developed to support distributed information retrieval and to support for strong mobility and

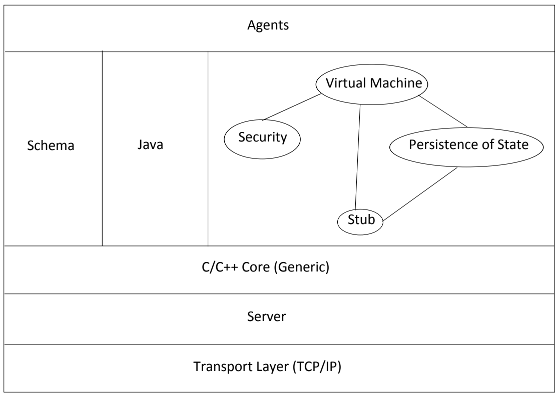

FIGURE 2.5: The D'Agent architecture

multi-agent languages. Using D'Agent, several information-retrieval applications, ranging from searching three-dimensional drawings of mechanical parts for a needed part to supporting the operational needs of a platoon of soldiers have been implemented. The architecture of D'Agent is shown in Figure 2.5. TCP/IP is used to provide transport mechanism. Server layer is a multi-threaded process and runs multiple mobile agents as threads inside a single process. The Generic C/C++ core layer holds shared C++ libraries used by agent threads. The upper layer provides the execution environment for Java, Tcl, or Scheme. The agents themselves are defined on the top layer Schoeman & Cloete (2003).

The disadvantage is that for deployment using Java platform the virtual machine (VM) needs to be extended instead of agent server that resides on top of VM.

Grasshopper is an OMG MASIF and FIPA-conformant agent platform, which consists of a Distributed Agent Environment (DAE) and a Distributed Processing Environment, as Figure 2.6 shows. A host in the distributed agent environment include an agency that has access to the services including execution, transport, management, communication, security, naming mechanism, adapter interfaces for external hardware/software, task control functions, and application-specific GUIs Schoeman & Cloete (2003). The distributed processing environment is composed of following components: Regions, facilitates the management of the distributed components (agencies, places, and agents) in the Grasshopper environment; Places, provides a logical grouping of functionality inside an agency; Agencies, as well as their places can be associated with a specific region by registering them within the accompanying region registry; and Different types of agents – mobile agents and stationary agents. Mobile agents move from one platform to another, whereas stationary agents reside

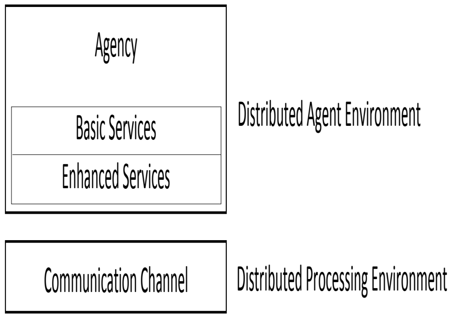

FIGURE 2.6: The simplified version of Grasshopper Architecture. The basic services include MASIF and Core Services. MASIF includes agent creation, destruction, suspend, activate and location services and Core services include agent execution, transport, management, communication, security and naming. The enhanced services include APIs, GUIs and task control features.

on one platform permanently (Grasshopper Mobile Agent Platform).

The main disadvantage of Grasshopper is that it is not available anymore and new versions will not appear in the future. The region server could become a bottleneck, as it must update every proxy right before using it Trillo et al. (2007).

Aglets (Aglet) is a well known Java based mobile agent platform, which contains libraries for developing mobile agent based applications. This platform follows MASIF specification Trillo et al. (2007). Aglets are built around single-thread model for agents and a communication infrastructure based on message passing. Both synchronous and a synchronous messages are supported Trillo et al. (2007). Agents in Aglets use proxies as abstraction to refer to remote agents for sending messages that is similar to stubs in Remote Method Invocation (RMI). As Figure 2.7 shows, Aglets' architecture consists of two layers: Runtime layer, consists of a core framework and sub-components to provide services such as serialization/de-serialization, class loading and transfer, reference management and garbage collection, persistence management, maintenance of byte code, and protecting hosts and agents from malicious entities; and Communication layer, defines the methods for creating and transferring agents, and tracking and managing agents in an agent-system-and-protocol-independent way Schoeman & Cloete (2003).

The drawback of this platform is that the proxies it provides are not dynamic proxies and hence cannot be used when the agent has migrated which means in case of using it again, the proxy has to updated manually. Single thread model is also an issue as in case of synchronous messages being sent by one agent to other agent at the same time can result

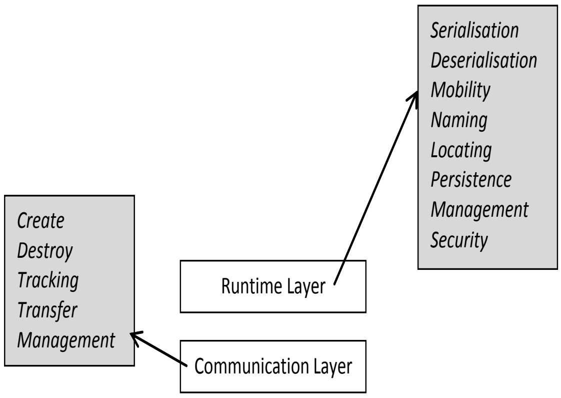

FIGURE 2.7: The Aglets architecture Source: Schoeman & Cloete (2003)

in a deadlock.

Table 2.4 summarises the main features of mobile agent platforms.

Table 2.4: Qualitative Comparison Among Mobile Agent Platforms Source: Trillo et al. (2007)

Model	Behaviours	Events	Procedural	Procedural	Procedural
Elements	Containers Main containers Platform Agents DF, AMS, MTS	Contexts Agents (aglets) Tahiti	Servers Agents	Regions Agents Places	Regions Agents Places
Proxies	No	Yes	Yes	Yes	Yes
Dynamic proxies	No	No	Yes (forwarding)	Yes	Yes
Synchronous communication	No	Yes (deadlocks)	Yes	Yes	Yes

Feature	JADE	Aglets	Voyager	Grasshopper	SMART
Available to download	Open Source	IBM Public Licence	Not Free	Open Source	Open Source
Asynchronous communication	Yes	Yes	Yes	Yes	Yes
Messages	Yes (FIPA Standard)	Yes	No	Yes (FIPA)	Yes
Remote calls	No	No	Yes	Yes	Yes
Callbacks after migration	No	No	Yes	No	No
Call/messages by name	Yes (Agent Identifier)	No	No	No	No
Migration by name	Yes (AMS)	No	No	No	No
GUI tools	Yes	Limited	No	No	No
Level of activity	Very High	Very Low	Medium	None	None
Security mechanism	Yes	Basic	Yes (security managers)	Basic	No
Some other features	Ontology Support, FIPA Compliant	Itinerary Setup	Multicast Publish and Subscribe	MASIF FIPA	MASIF

JADE was one of the first FIPA-compliant platforms developed. JADE offers an agent runtime system and a predefined programmable agent model and of a set of management and testing tools that are missing features in other platforms. It simplifies the development of applications that requires negotiation and coordination that is one of the highlights of MAS system developed as outcome of this project. With use of ACL and mailboxes for each agent, developers steer clear of remote method invocations where the remote references require updating upon migration - a facility that is required for mobile agent. Not that it is required within the scope of this research work but due to JADE's compliance with FIPA specification end-to-end interoperability between agents of different platforms is possible. JADE's API is independent from underlying network and Java version and is standard across Java Enterprise Edition (J2EE) and Java Mobile Edition (J2ME) that allows reusability of application code. Also, JADE has ontology support where this work can be extended for future work.

2.6 Summary of the Chapter

In this chapter, the author has collated and researched the information in the field of resource discovery on unstructured, structured, and MAS systems. Also, the author has categorised the resource discovery techniques used in various types of overlay networks. This researched information provided insights into resource discovery systems and clearly characterised the properties that such systems should be attributed with. Based on these insights, in Section 2.3, the author proposed the characteristics that a successful resource discovery system should have for achieving maximum search result relevance with minimum search load and messages on the overlay network. In the penultimate section, the author discussed various mobile agent platforms - their features and drawbacks and also justification for use of JADE platform for development of MAS system.

In next chapter, design features and implementation of the proposed system are provided. The details about implementation of proposed design features are presented in program listings section.

Chapter 3

Design Features and Implementation

The chapter details about system architecture and design features that implement the proposed characteristics of a resource discovery system using mobile agent. The author has conceived multi-agent resource discovery system using mobile agent called Affinity that

1. Captures the features of clustering of peers based on semantics of content shared,
2. Handles multiple keys to locate a resource by use of LSI similarity, and
3. Finally reduce the bandwidth consumption by providing mobile agent with ability to negotiate with peers regarding finding next site for migration and matching resource hosted by peer to user query under given constraints from user.

The features have been divided into sections and each section of this chapter discusses that feature and its realisation. The final section provides snippet of mobile agent communication - agent communication language, based on FIPA standards implemented using JADE and also implementation details of proposed features using JADE and Java. Detailed information about code based implementation and its deployment are found in program listings section of the thesis.

3.1 The Proposed Multi-Agent System for Resource Discovery - Affinity

3.1.1 The Proposed Global System Architecture

The architecture for the conceived system is illustrated in Figure 3.1. As articulated in the figure, the system has four layers - interface layer, reconnaissance layer, directory and resource layer and visiting agents layer. Each layer contains agents dedicated to perform certain task (detail specifications of agents are provided in Section 3.1.2).

The purpose of each layer is as follows:

- *Interface Layer*: This layer contains the interface agent that is used by the client to interrogate the system. The goal is to capture the requirements or needs of the user and respond back to them appropriately. User's interaction with system is through interface agent that helps in realisation of the given task. The request from user i.e. the search query facilitates the function of reconnaissance layer. The additional function of transformation of the submitted user request into a feature vector is also realised in layer.

- *Reconnaissance Layer*: This layer contains the reconnaissance agent that is created as a result of submitted query in the interface layer. The function of this layer is to temporarily contain the new created mobile agent while it communicates to stationary agents in directory and resource layer for node address where it can migrate to in order to realise the submitted query.

- *Directory and Resource Layer*: The function of this layer is to receive requests from reconnaissance agent, process them and return the results. This layer holds two stationary agent - local agent and information agent and is responsible for managing the data associated to shared resources on the node and multiple sources of node addresses that are semantically similar to content shared on this node. The task of determining appropriate node address and hence deterministic route to the node that hosts resource similar to given query is completed in this layer. The management of directory of shared resources on this node that are transformed into feature matrix after indexing is the function of local agent and the management of list of peers that are semantically similar to content of this node is done by information agent. The functionality to achieve autonomy is also achieved on this layer where information agent communicates to bootstrap server about its status every $300,000ms$.

- *Visiting Agent Layer*: The function of this layer is to provide platform for the migrated reconnaissance agent that is visiting a particular node. This layer is a class that is capable to provide functionality of sending messages to and receiving messages from directory and resource layer of the visited node. This layer also provides additional functionality of query matching by collaborating with directory and resource layer for

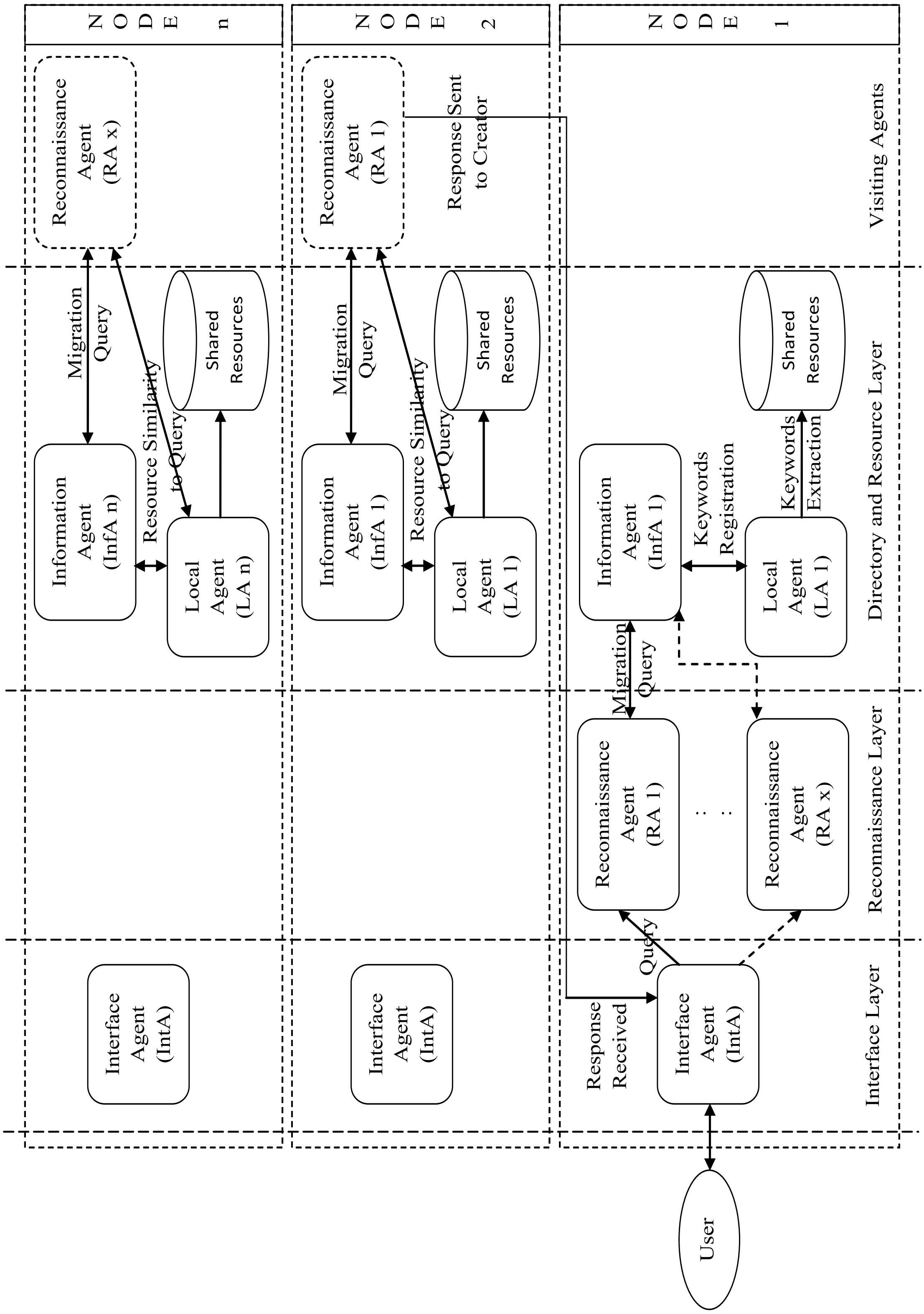

FIGURE 3.1: Global architecture of the system

realising the task of finding resource(s) hosted on this node that is semantically similar to submitted query.

3.1.2 Specification of Agents

The proposed system - Affinity is hybrid system based on the semantic overlay network, unstructured P2P system, and MAS. All peers share their resources that are maintained by the set of collaborating agents on each peer. The collaborating agents on each peer are

1. *Interface Agent (IntA)*,
2. *Local Agent (LA)*,
3. *Information Agent (InfA)*, and
4. *Reconnaissance Agent (RA)*.

The purpose of each is as follows:

- *Interface Agent*: IntA is a static agent that provides user interaction to the system. The user interacts with IntA using the GUI interface that a.) shows search query, b.) informs search results, and c.) inform active RA(s).
- *Local Agent*: LA is another static agent that holds information i.e. keys for defining local resources and the corresponding location of resource on the peer. In addition, it has tasks to serve InfA for keywords request and RA for keyword similarity. Local indexing of shared resources is maintained by LA.
- *Information Agent*: InfA holds information about peers that are semantically similar to this peer i.e. the indexing results propagated by bootstrap server are maintained by this agent. It holds a data structure that contains all peer's GUID, similarity value and keywords that it is sharing. InfA is responsible for computing routes for RA upon request of migration query. InfA also communicates to LA to request a list of keywords that a peer is sharing that it in turn is submitted to bootstrap server for registration and finding peers that belong to same cluster.
- *Reconnaissance Agent*: RA is a mobile agent for resource discovery; that is created by the IntA upon user's search request. RA migrates to new peers by requesting node

address from InfA. RA's task is to migrate to peers and to investigate LA that is responsible for hosting resources (hence keywords) about their possible similarity to user's query and report it to IntA.

In addition to the proposed multi-agent system, the overlay network organisation of this P2P system is improved by InfA registration to bootstrap server. A *bootstrap* server maintains a list of peers that are currently in the system. Upon *registration/joining*, the bootstrap server replies with list of peers that are semantically similar to this peer.

As detailed in Section 3.2.1, the cosine similarity between peers is actually keyword similarity of hosted resources of those peers. The result of this similarity is cluster effect as illustrated in Figure 3.2. Although, the sparsity of keyword matrix on the bootstrap server is large but still it is overlooked by potential advantage, that each peer is now organised in overlay network (i.e. it only knows the address of neighbours in a cluster). The Globally Unique Identifier (GUID) of neighbours in cluster are used to prepare a hash table that is maintained by InfA. When a neighbour *disconnects* from overlay network, it informs bootstrap and its neighbours to remove its GUID from matrix and hash tables respectively. Upon *creation* of RA, it communicates to InfA to provide it with itinerary (next site) for migration. InfA uses the hash table provided it by bootstrap server to issue a peer GUID that host resources/keywords that are close to requested user query.

3.2 The Proposed Mobile Agent Routing

Peer clustering is based on the conceptual content of resources shared by peers. The objective is to organise an overlay network in such a way that when given a query, small number of peers are selected based on "higher" chance of query hit. The benefit of this strategy is two-fold. First in context of peer clustering - the peers to which RA migrates to will have many matches, so that the query is answered faster, and second in context of RA routing - the peers with "lesser" chance of getting a query hit will be steered clear by the migrating RA, thus avoiding wasting resources on that query (and allowing other queries to be processed faster). Peer Clustering and Agent Routing are accomplished using LSI. The following Section 3.2.1 explains the state vector and singular vector decomposition (SVD) based semantics for peers and keywords hosted.

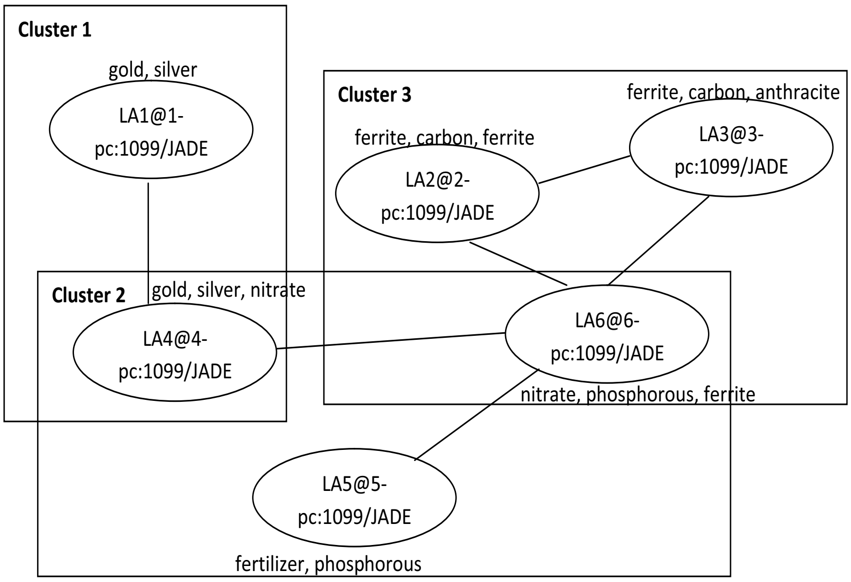

FIGURE 3.2: Peer clustering and overlay organisation achieved using *latent semantic indexing*

3.2.1 Latent Semantic Indexing and Singular Value Decomposition (LSI-SVD) for Peer Clustering and Mobile Agent Routing

Latent semantic indexing (LSI) is a variant of a vector space model, where low rank approximation to the vector representation of the corpus is computed Gao & Zhang (2005). LSI considers that latent structures may exist in documents that may not be visible and may very well be hidden due to variability in word choice Gao & Zhang (2005). Singular value decomposition (SVD) of the corpus is calculated to estimate the structure of lexicon usage across the documents.

The nodes may be represented by number of keywords (lexicons) that it shares. Hence, a set of nodes can be represented by a matrix called *keyword-peer* matrix A. The elements of the *keyword-peer* matrix represent the frequency of each keyword f on a particular node. Let N be the number of peers in a P2P network, and K be number of distinct keywords (lexicons). It should be noted that N can be resources when observed from RA-LA point of view, but generically the author assumes it as number of peers. The feature matrix called *keyword-peer matrix* is constructed as $A\epsilon\left[a_{ij}\right]_{KxN}$ where a_{ij} =frequency of the keyword i on node j. $A_{ij} = 0$ if the peer j does not contain the keyword i. Not all keywords appear on all peers and hence matrix A is generally it is a sparse matrix. Now, matrix

A denotes <peer, keyword> pairs in the network, which is the knowledge of correlations between peers and keywords. To properly characterise latent semantics and correlations between peers in LSI, that matrix A is factored into product of three matrices using SVD Golub & Loan (1996).

$$A = USV^T \tag{3.1}$$

$U^TU = I_K$ and $V^TV = I_N$, I_K and I_N are identity matrices of order K and N respectively. Matrix S is a diagonal matrix with elements $diag[\alpha_1, \alpha_2, \alpha_3, ...\alpha_{min\{K,N\}}]$, $\alpha_i > 0$ for $1 < i \leq d$, and $\alpha_j = 0$ for $j > d$, where d is the dimensionally reduced matrix. SVD is a low rank approximation of matrix A Golub & Loan (1996). SVD is used to find the singular vectors corresponding to k largest singular values which dominate the original matrix. Peers and keywords can be characterised by linear combination of singular values i.e. a k-dimensional point in the feature space spanned by k singular vectors Liu et al. (2004). Deerwester et al. (1990) shows the small dimensions are enough to express latent semantic i.e. $k \ll min\{K, M\}$. The resulting singular vector and singular value matrices are used to may keyword-based vectors for peers and queries into a subspace in which semantic relationships from the *keyword-peer* matrix are preserved while keyword usage variations are suppressed Hasan & Matsumoto (1999). The reduced dimension decomposed matrix as a new pseudo-keyword-peer matrix is given by

$$A_k = U_k S_k V_k^T \tag{3.2}$$

where columns of U_k contains the eigenvectors of the $A_k A_k^T$ matrix or first k columns of matrix U and the rows of V_k^T are the eigenvectors of the $A_k^T A_k$ matrix or first k rows of matrix V^T. S_k is a diagonal matrix that has its diagonal elements with special kind of values of the original matrix Deerwester et al. (1990); Golub & Loan (1996). These are termed the singular values of A_k that has first k largest singular values.

In SVD representation of original vector space, $A_k^T A_k$ is a $N * N$ symmetric matrix for inner products between peer vectors, where each peer is represented by a vector of keyword weights. This matrix can be used for cluster analysis for collection of peers. Each column of matrix, $A_k^T A_k$ is a set of inner products between peer vectors in corresponding column of the matrix A, and every peer in the collection. The cosine similarity measure of peers i and j can be computed as follows:

$$sim(i,j) = \frac{\langle i,j \rangle}{|i||j|} \tag{3.3}$$

For information retrieval in K-dimensional space query Q is treated as another set of keywords and hence query Q becomes $q = Q^T U_K S_K^{-1}$that is compared to the peer represented by $p = p^T U_k S_K^{-1}$. These equations present the coordinates of the vectors in the K-dimensional space and query-peer cosine similarity is given by

$$sim(q,p) = \frac{\langle q,p \rangle}{|q||p|} \tag{3.4}$$

All peers share the keywords that inform about the hosted resources. The similarity between the keywords shared by various peers forms a cluster of peers that are similar to each other and thus forming a cluster and in turn an organised overlay network. This also increases the efficiency of discovering a resource as number of hops that RA has to take to find a resource are decreased. In order to get list of peers, another parameter - *minimum support* is passed by user. The significance of this parameter is to give user a level of control over list of known peers by forming a "canopy" on known peers. The value of minimum support ranges between -1.0 to $+1.0$ where -1.0 explains ambiguity - list of all peers registered i.e. ignoring the similarity results, and $+1.0$ explains certainty - list of all peers that are exactly similar to this peer i.e. only peers that are sharing same keywords with same frequency.

RA's routing is directly affected by the minimum support value passed by user during acquisition of peer list i.e. lesser the value of minimum support, larger set of peer list and that means RA has larger number of ambiguous peers to choose from or vice versa. However, another value of *minimum support* for resource discovery and this time it means the similarity of query passed by user to the keywords shared by various peers in peer list allows RA to find the peer where it will migrate to.

The exact value of minimum support has not been established but through experimentation it is realised the initial value for peer registration can be $+0.1$ or higher and for resource location $+0.5$ and higher can provide suitable results.

The unstructured network is created at random where to locate/search for particular resources, the message has to be forwarded to number of times. If this is limited by N hops, where N is the number of nodes within the query message's reach, then query routing complexity on an unstructured P2P network is of the order of N, or $\mathcal{O}(N)$. On structured networks, or the MAS that have underlying overlay network based on structured overlay the query routing complexity is typically $\mathcal{O}(\log(N))$, where N is the number of network nodes This is because the size of routing table increases according to power of two hence each step cuts the distance to target resource by half thus resulting in a lookup complexity of $\mathcal{O}(\log(N))$ Doval & O'Mahony (2003). In the proposed case, suppose N is the number of nodes and m is the *minimum support* of a node that ranges as $0.0 \leq m \leq 1.0$, and N_D is the maximum number of nodes that are semantically close where $N_D \ll N$, then the complexity of query routing is given as $\mathcal{O}(N_D)$, when $m = 0.0$ and $\mathcal{O}(N_D^{-\log(m)})$, when $0.0 < m \leq 1.0$. As *minimum support* increases the number of nodes required to be visited by migrating RA decrease logarithmically, and when *minimum support* is 0.0, it means the RA has to visit all nodes N_D in this particular domain. It is seen that the query routing complexity for resource location is much more effective is our system as compared to structured and unstructured system because of informed migrations performed by the RA. The results are later justified in experimentation in Chapter 4.

3.3 The Proposed Multi-Agent Collaboration for Resource Discovery

The system starts by starting up a bootstrap server. The LA locates all the resources that are shared by the peer and preparing the keyword list that defines the resource. The InfA requests the LA to inform it about the keyword list that in turn is used by the InfA to register the peer on bootstrap server. This behaviour is a cyclic behaviour of InfA that is scheduled every $300,000ms$. Upon registration, based on *minimum support* value, the InfA receives the peer list containing list of peers, their similarity value and keywords shared by those peers. User's request for resource location to the IntA is attributed by list of keywords that form a query, *minimum support* value for acceptable results, and number of hops that the RA can make. The detailed interactions between the collaborating agents are shown in Figure 3.3.

The resource discovery is carried out using following algorithm:

1. When query is passed by user to the IntA, the IntA in turn creates the RA for that specific query.

2. The RA is informed about query, minimum support, and number of hops by IntA.

3. The RA requests the InfA for peer name in order to create route for migration. The

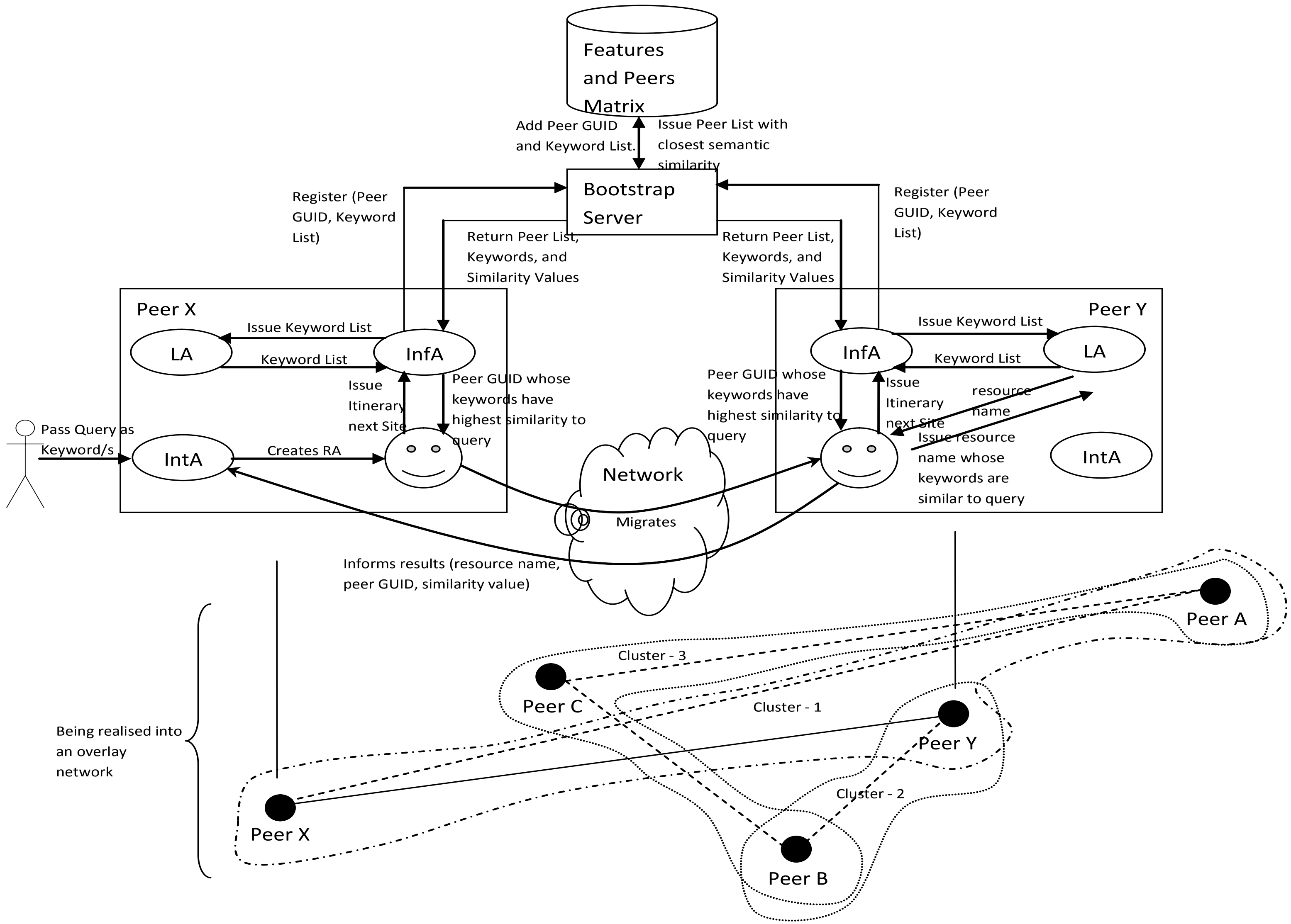

FIGURE 3.3: Interactions between multiple agents for resource discovery and realisation of an overlay network

peer name is informed by the InfA to the RA by looking up the peer name in hash table based on the semantic similarity of the query and the keywords shared by that peer.

4. The RA requests the agent management service (AMS) to find the container/platform where the selected peer is located.
5. The RA migrates to that peer and increments the number of hops by one.
6. The RA requests the LA of this peer to inform it about the resource name whose keywords are semantically similar to the query and higher than minimum support given by user. The LA provides the resource name to the RA.
7. The RA informs the IntA about located resource i.e. GUID of the peer where resource is located, resource name, cosine similarity value.
8. If number of hops made by the RA is less than maximum number of hops allowed by user then go to Step 9 otherwise, go to Step 10.
9. The RA requests the InfA of this peer for a new peer name where it should migrate to (hops to previously visited peers and to creator peer are not allowed). Go to Step 3.
10. As, number of hops made by the RA are equal to maximum number of hops allowed, the RA terminates itself.

Step 3 is shown in detail in Figure 3.4. The RA requests the InfA for GUID of the peer that it should migrate to; to find the resource. The InfA refers to the directory and calculates cosine similarity value based on degree of match between the query and list of keywords available. The highest similarity value is used to determine the peer GUID by looking up in the directory. Finally, the GUID of selected peer is informed to the RA. The RA now uses the GUID to find the container name from AMS where the corresponding peer GUID resides. The GUID of agents is generated based on container identifier and type of agent. This mechanism is better than "blind" or flooding technique as in this case the RA migrates with certain knowledge i.e. where and why to migrate to a certain peer as opposed to flooding the overlay network with communication messages or with multiple clones of the RA. Essentially, it improves the routing of the RA. The behaviour of the RA has been defined by *beforeMove* and *afterMove* methods. afterMove method is invoked just

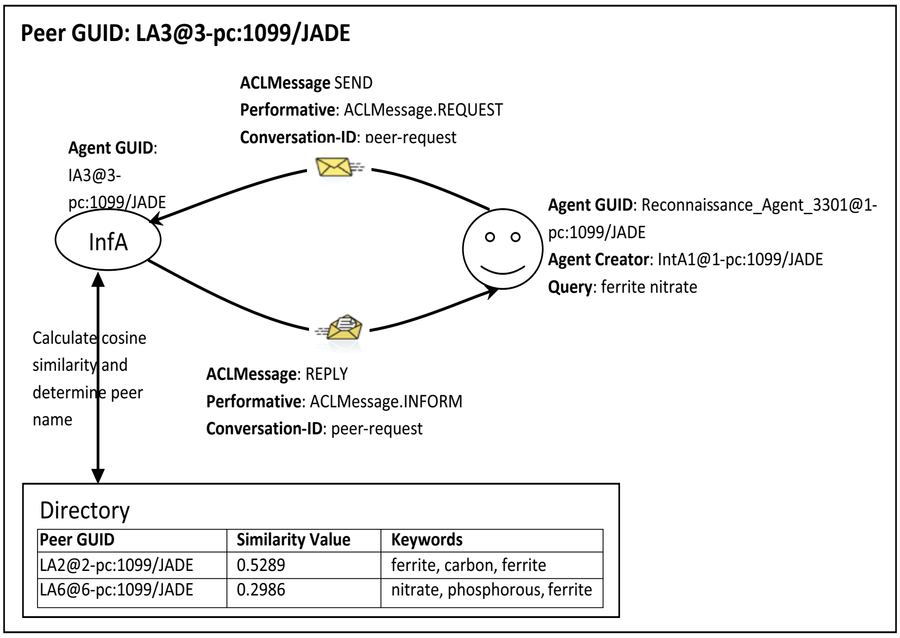

FIGURE 3.4: The RA's interaction with InfA for issuing new peer GUID

after migration to increment the number of hops made by the RA followed by checking the termination conditions.

Figure 3.5 presents the flow diagram for behaviour of the InfA and the LA when the RA arrives at a certain peer. Shown in the flow diagram are behaviours of three agents the RA, the LA, and the InfA. In addition to behaviours of agents, *blockedState* of the RA and *blockingReceive* of the RA is observed. These methods are invoked based on the ACLMessages in the mailbox of each agent. Essentially, as long as the RA has not received any message that matches the *MessageTemplate* (as seen in Figure 3.6), the RA is in blocked state.

3.4 Implementation

This section presents in detail various functions that have been implemented for realising the features viz. - feature matrix - indexing, clustering (nodes learning about other nodes), ranking and selection (nodes ranked and selected for routing of mobile agent), similarity, and behaviour of agents. The author has presented algorithm or pseudocode and its code based realisation details. Detailed implementation details can be found in Appendix C for

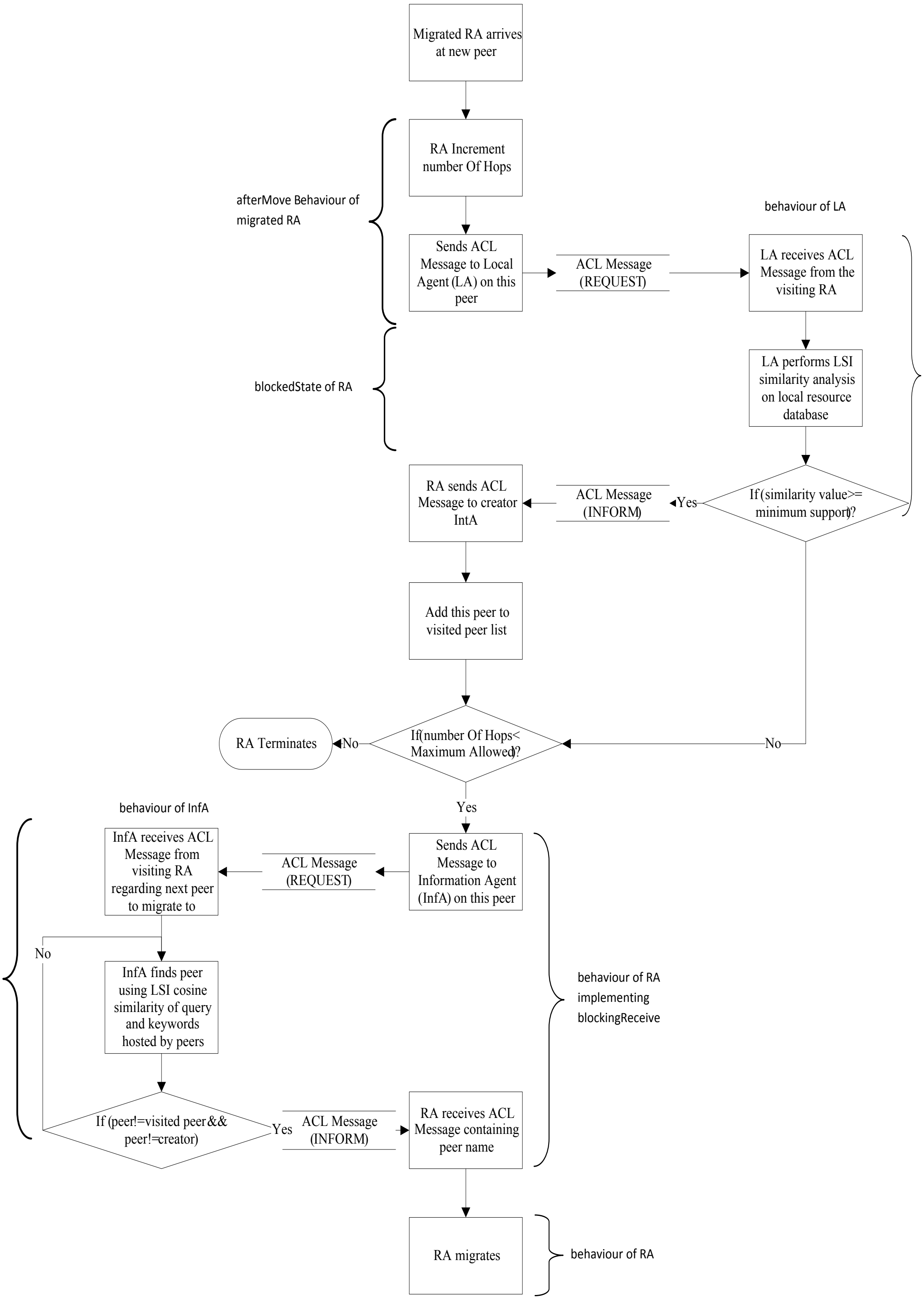

FIGURE 3.5: Flow diagram for behaviour of the InfA and the LA upon arrival of the RA

```
ACLMessage request=new ACLMessage(ACLMessage.REQUEST);
request.addReceiver(new AID(destLAName,AID.ISGUID));
request.setConversationId("search-request");
request.setReplyWith("request"+System.currentTimeMillis());
request.setContent(cont);
send(request);

private class ServeIncomingMessage extends Behaviour
{
        private MessageTemplate mt =
        MessageTemplate.and(MessageTemplate.MatchConversationId("search-
        request"),MessageTemplate.MatchPerformative(ACLMessage.REQUEST));
        public void action()
        {
                try
                {
                        ACLMessage request = receive(mt);
                        ACLMessage reply = request.createReply();
                        if(chosen!=null)
                        {
                                reply.setPerformative(ACLMessage.INFORM;
                                reply.setContentObject(matchStore);
                        }
                                myAgent.send(reply);
                        }else
                        {
                                System.out.println("No message yet");
                                block();
                        }
                }catch(Exception e)
                {
                        e.printStackTrace();
                }
        }
}//end class
```

FIGURE 3.6: *ACLMessage* from the RA to the LA for search request. ACLMessage received by the LA from the RA using *MessageTemplate* and replying with *setContentObject* or in blocked state.

Program Listing or media disc.

The author has used Java remote method invocation (RMI) and Java Agent Development Framework (JADE) (Bellifemine et al. (2007)) to implement the multi agent resource discovery using mobile agent system. Jade's agent management environment is used for creating multiple containers emulating distributed environment where peers are active. Java Remote Interface has been used for defining and implementing the bootstrap server.

3.4.1 Agents Communication Implementation

All the agents are developed using FIPA complaint agent framework - JADE. Instead of using RMI or socket based communication between various agents including the mobile agent (RA), agent communication language (ACL) has been used for communication particularly using performative (REQUEST, INFORM, and CLP (Call_For_Proposal)). In addition as the RA is a mobile agent, it is further required to register FIPA standard FIPA_SL0 (slCodec) content language.

Agents do not invoke methods on other agents and communicate through ACLMessages. Hence, to handle messages from various agents and/or various kinds of messages the author has implemented the use of *MessageTemplate*. A receive method takes a message template as a parameter and only returns messages matching that template. This is an important feature that is implemented for successful multi-agent communication system. Figure 3.6 shows the snippet. The behaviour implemented by the LA includes case:

1. Where the RA communicates to the LA to locate resource name whose keywords are semantically similar to user's query

2. Where the LA informs the RA about selected peer GUID using ACL.

The multi agent system has been designed to receive search requests from the users through the IntA. IntA class has a graphical user interface associated with it that takes input parameters - keywords for the query (search terms for a resource). The *minimum support* and the time to live (*number of hops*) parameters have been defaulted in the experimental setup to be 0.00 and 3 hops respectively. Upon invoking the search, the RA is created by the IntA in the method *onGUIEvent()* as shown in Figure 3.7 that has an identifier - (GUID) and the minimum support and number of hops as the parameters.

In addition to creation of the RA, the IntA is also responsible for displaying results sent by the RA throughout its life cycle. As mentioned before in Section 3.4, the FIPA specification implemented by JADE does not allow agents to communicate to each other using method invocation or more specifically in this case remote method invocation, the IntA hence, offers functionality for receiving messages from the RA through ACL implemented in the inner class *ReceiveMessageRecon*. This inner class extends the *CyclicBehaviour*, that creates instance of *MessageTemplate* for only receiving messages sent by instances of the RA created by this instance of IntA using *MatchCoversationId("results")* and *MatchPerformative(ACLMessage.INFORM)*. See Figure 3.8 for details of *MessageTemplate* for receiving message from the RA. The IntA also has an inner class *ReceiveTerminationRecon* that extends *SimpleBehaviour* for receiving termination message from the RA when RA has reached end of its life cycle or if a matching resource has been discovered.

The RA is responsible for discovering the resource on other nodes by migrating to those nodes. The node that is most likely to host the resource is provided by the InfA that holds directory of nodes for routing the RA on the overlay network. Again, the communi-

```
protected void onGuiEvent(GuiEvent ev)
{
command=ev.getType();
....
        if(command==NEW_RECON_AGENT)
        {
            jade.wrapper.AgentController a = null;
     try
            {
                Object[] args=new Object[5];
                args[0]=getAID();
                System.out.println(args[0]);
                args[1]=gui.getQuery();//query
                args[2]="0.0";//minimum support
                args[3]=(Object)name;
                args[4]="2";//number of hops
                String name_of_Agent="Reconnaissance_Agent_"+(count++);
                a=home.createNewAgent(name_of_Agent,ReconnaissanceAgent.class.getName(),args);
                a.start();
                agents.add(name_of_Agent);
                gui.activeAgents(agents);
            }catch(Exception ee)
            {
                System.out.println("Problem while creating new agent "+ee);
            }
            return;
        }
}
```

FIGURE 3.7: Creation of the RA in the method *onGUIEvent()* from class InterfaceAgent

```
  //inner class
private class ReceiveMessageRecon extends CyclicBehaviour
    {
        MatchStore matchStore=null;
        MessageTemplate mt =
MessageTemplate.and(MessageTemplate.MatchConversationId("results"),MessageTemplate.MatchPerfor
mative(ACLMessage.INFORM));
        public void action()
        {
            try
            {
                ACLMessage reply = receive(mt);
                if(reply!=null)
                {
                    matchStore=(MatchStore)reply.getContentObject();
                    gui.setResult(matchStore);
                }else
                {
                    block();
                }
            }catch(Exception e)
            {
                e.printStackTrace();
            }
        }
    }
```

FIGURE 3.8: *ReceiveMessageRecon* class showing *blocked state* of when reply received is *null* and the *MessageTemplate* for receiving messages from the RA

cation between the InfA and the RA is using ACL and the *MessageTemplate* uses *MatchPerformative(ACLMessage.REQUEST)*. See InformationAgent.java code in Appendix - C.

Upon migration, the RA communicates to the LA as shown in Figure 3.6 for a matching resource. All the results obtained are communicated back to the IntA through the *MessageTemplate* described above.

3.4.2 Feature Matrix - Frequency-Based Indexing

Feature matrix is created for shared resources hosted by nodes. A modified form of feature matrix called feature vector is used to present node based on the content shared. The process of indexing has two sub tasks. The first subtask is the assignment of tokens for a resource and the second subtask is the assignment of weights to the tokens. The weight is numeric value that is directly proportional to the importance of the token in a resource. The weights are of type integers. These integers present the count of number of unique tokens in a resource. The text for a resource is split into tokens where tokens are only content keywords (adjectives, adverbs, nouns and verbs). The content keywords form index. The representation for a node called feature vector is created by using the indexes for entire collection of resources on that node. Number of feature vectors when collated on bootstrap server form the master feature matrix for entire collection of participating nodes. Each node also has keyword-resource feature matrix that is created by recording frequency of keywords for each resource. The process of locating keywords is given in following pseudocode:

1. Receive the text to be parsed.
2. Build a custom stopword list based on the type of text.
3. Generate a list of tokens from the text of given resource.
4. Initialise a list of content words and loop through the list of tokens.
    (a) Skip the token if it does not begin with a valid character.
    (b) Skip tokens that are less than 3 characters long.
    (c) Skip tokens that are found in the stopword list.
    (d) Add the token to the list of content words.
5. Return the list of content words.

```
public void getKeywords()
{
        int index=0;
        String keywordSet="";
        ArrayList keywords1 = message.getKeywords(); //Data Structure for holding keywords
        for(int i=0;i<keywords1.size();i++)
        {
            keywordSet+=(String)keywords1.get(i)+" "; //Concatenate Keywords
        }
        //Tokenise keywordSet based on Regular Expression
        StringTokenizer token = new StringTokenizer(keywordSet);
        //Get number of rows
        size=token.countTokens();
        //Create Array based on number of keywords found
        makeTKArray(size);
        //Loop and count keywords
        while(token.hasMoreTokens())
        {
            tk[index]=token.nextToken();
            findTokenFrequency(tk[index]);
        }
        //add keywords and their frequency into TreeMap
        database.addKeywords(map);
}
```

FIGURE 3.9: Method *getKeywords()* for getting keywords and their frequencies and holding in data structure TreeMap

For finding frequency of keywords found using the above psuedocode, method *getKeyword()* in private class *FrequencyFinder* is invoked. The method stores all keywords in *TreeMap* data structure as shown in snippet Figure 3.9. *TreeMap* guarantees that the map will be in ascending key order, where keys are distinct keywords and values are the frequency of each key. The list of tokens/keywords in step 3 is stored in *ArrayList* data structure. For loop is used for getting frequency of each token and storing the counting as a value in *TreeMap*. This forms a feature vector for each resource and the collection of feature vectors for all resources on a node form a *keyword-resource* matrix. This functionality is achieved by concatenation of all feature vectors to form a sparse matrix called *masterKeywordMatrix* in private class *LocalDatabase*. The *masterKeywordMatrix* is a two-dimensional array of type *double*. Each node is represented by concatenated list of keywords and their frequency that is globally maintained by in class *MasterList*.

3.4.3 Implementation of Latent Semantic Indexing and Singular Value Decomposition

Frequency-based indexing method cannot utilise any global relationships with the resource collection Konchady (2006). LSI indexing method based on the SVD transforms the *keyword-resource* matrix such that major intrinsic associative patterns in the collection are revealed.

Algorithm 3.1 Algorithm for Latent Semantic Indexing of keyword-resource or keyword-node matrix

$Input\ keyword - resource\ or\ keyword - node\ matrix\ A(i,j)$

$A = (A(i,j))\ where\ i = 1,t\ ,\ j = 1,r\ (t * r)\ matrix\ of\ keywords\ and\ resources$

$Perform\ SVD:\ A = USV^T$

$Set\ all\ but\ the\ k\ highest\ singular\ values\ to\ 0$

$Compute\ A_k = U_k S_k V_k^T\ by\ retaining\ the\ largest\ k\ singular\ values$

$Output A_k\ Latent\ Semantic\ Index$

```
/**
 * SVD calculation
 */
   public void calculateSVD(double[][] matrix)
   {
       Matrix mat = new Matrix(matrix);
       SingularValueDecomposition svd = mat.svd();
       U = svd.getU();// Left Eigen Vectors
       S = svd.getS(); //Singular Values
       S_inverse = S.inverse();
       V = svd.getV(); //Right Eigen Vectors
       V_transpose= V.transpose();
   }
```

FIGURE 3.10: Realisation of Singular Value Decomposition from frequency based keyword-resource or keyword node matrix

LSI does not depend on individual keywords to locate a resource, but rather uses concept to find relevant resource. The main purpose of transforming the projection of resource from vector space to LSI space is to locate groupings of resources and use a similar representation for the group (hence a cluster). The algorithm for performing LSI on a group of resources is given as follows (See Algorithm 3.1):

The implementation in the *calculateSVD(double[][] matrix)* method of *MasterList* class the data structure called *Matrix* provided in JAMA API to create a clone of double[][] array and then computes the decomposition of the matrix by invoking method *svd()* (See snippet in Figure 3.10). The return type of this method is *SingularValueDecomposition* that is further used to invoke accessor methods *getU()*, *getS()* and *getV()* for getting left eigen vector (U), singular orthogonal matrix (S) and right eigen vector (V) respectively. The output matrices are then subjected to dimensionality reduction based on top k sigma value in singular matrix (S). The number of sigma values, k, is the floor of the square root of number of resource. A new keyword-resource matrix is generated using the truncated k dimensions.

```
private class KeywordRequestor extends TickerBehaviour
{
       private KeywordRequestor(Agent a)
       {
           super(a,300000); //Timer of 300,000ms
       }

       ...

       //Behaviour Implemented upon expiry of Timer
       public void onTick()
       {
           //Send Message
           ACLMessage request=new ACLMessage(ACLMessage.REQUEST);//Request Message
           request.addReceiver(new AID(nameLA,AID.ISGUID));//Receiver Local Agent
           request.setConversationId("keywords-request");//ID keywords-request
           request.setReplyWith("request"+System.currentTimeMillis());//Update Time
           myAgent.send(request);//Post Message
           ....
           callNodeRegistry();//Update Bootstrap Server
           }
           else
           {
              block();//Blocked State
           }
        }
}//end inner class
```

FIGURE 3.11: Index Maintenance task performed recursively by Information Agent

3.4.3.1 Index Maintenance

As the resource collection is dynamic and the nodes are autonomous, new resources and node are added and existing resources and nodes are modified or deleted. The index built from SVD of a keyword-resource or keyword-node matrix is a snapshot of the document collection at some earlier time. The changes made to the collection after the SVD computation, are not reflected in the index. For effective routing, clustering, the index of the bootstrap server must reflect the most recent state of the resource or node collection. Nodes are represented by the content hosted by them and the mobile agent is routed based on most updated state of index. To compensate for these changes, the information agent recursively (after $300,000ms$) updates the index by supplying bootstrap server with most recent state of a node. This behaviour is implemented in the inner class - *KeywordRequestor*, that extends *TickerBehaviour* that invokes method *onTick()* recursively after expiry of time passed as parameter in constructor - shown in Figure 3.11.

3.4.4 Similarity Function

The cosine measure is the ratio of sum of the products of common keywords to the products of the lengths of the two vectors. It measures the degree of overlap and uses the presence of keywords to compute similarity. As described in Section 3.2, the author has proposed the use of cosine similarity function for clustering nodes, searching nodes based conceptual similarity between node and query and also for matching the query resource hosted by a node. In following sections, the author presents realisation of these functionalities.

3.4.4.1 Node Learning - Clustering

For the purpose of clustering nodes that are conceptually similar, node represented by keywords is transformed into node vector of k dimensional space on bootstrap server. This transformation is required for comparing node vector to existing other node vectors for calculating cosine similarity. The similarity value of nodes that is less than *minimum_ support* constraint provided by user is returned to *InformationAgent*. The realisation of this functionality is provided in *MasterList* class. The return type is serialised object called *Directory* that contains the NodeId, similarity value and shared keywords. The *Directory* data structure forms a local repository and cluster of conceptually similar nodes. The pseudocode for locating nodes belonging to same cluster is as follows:

1. Initialise local node vector based on concatenation of keywords and their weights.
2. Submit local node vector to BootstrapServer.
3. Transform local node vector into k dimensional space.
4. Run a loop until convergence.
   (a) Calculate cosine similarity $sim(l_k, n_k)$ between the transformed vector and other available node vectors
   (b) If $(sim(l_k, n_k) > minimum\ support)$ then
      i. Node belongs to the clustered.
      ii. Add node GUID to *Directory* data structure.
      iii. Add node's similarity value to *Directory* data structure.
      iv. Add node's keywords to *Directory* data structure.

(c) Else if $(sim(l_k, n_k) < minimum\ support)$ then

i. Node does not belong to cluster, reject node.

5. Terminate when number of nodes converges.

The snippet in Figure 3.12 shows the serialised *Directory* data structure that holds the result of clustered nodes. This object is passed by *BootstrapServer* to *InformationAgent* in order to facilitate the functionality of plotting route upon query for *ReconnaissanceAgent* through overlay network.

```
/**
 * Data Structure for holding the directory peer - keyword matrix used by
 * Information Agent and Bootstrap
 *
 */
public class Directory implements Serializable
{
    //Hold Similarity Value
    double similarityValue;
    //Hold Keyword Frequency Weights
    Matrix keyWeights;
    //Hold Keywords
    ArrayList keywords;
}
```

FIGURE 3.12: Directory data structure used by *BootStrapServer* to pass clustered nodes result to *InformationAgent*

3.4.4.2 Node Searching and Ranking - Content Based Routing

For guided search on an overlay network and hence to reduce saturation, the mobile agent is required to have some heuristics about nodes on the overlay network. As described and implemented in Section 3.4.2, all nodes are represented by the concatenated set of keywords and their respective weights. In order to guide *ReconnaissanceAgent* towards the node that host resource that is conceptually similar to the query passed by user, cosine similarity is measured between the query keywords and the list nodes available to node. Based on *minimum_support*, the selected nodes are sorted and ranked such the node with highest similarity value is ranked as 1. The *ReconnaissanceAgent* is issued with GUID of this selected node that is further used by *ReconnaissanceAgent* to request AMS for container, where the selected node exists. Once the container address is available, the *ReconnaissanceAgent* migrates to this selected node for facilitating query resolving task. The pseudocode for selection and ranking of node is as follows:

1. Assuming that *Directory* containing list of nodes - their similarity values, keywords and GUIDs is available to node.
2. For each node in *Directory*
   (a) Measure cosine similarity between node and issued query
   (b) Add node GUID and the similarity value to *HashMap*
3. Sort elements in *HashMap* based on similarity value - to get node with highest similarity value as rank 1
4. While node is not selected
   (a) If node GUID does not exists in visited nodes array then
      i. Select node GUID
      ii. Inform *ReconnaissanceAgent* about GUID of selected node
      iii. Change state to node selected
   (b) Else
      i. Increment index of visited node array.
5. Migrate *ReconnaissanceAgent*

The pseudocode is implemented using a private class *NodeRequestor* that extends *CyclicBehaviour*. Upon receiving an *ACLMessage.Request* from *ReconnaissanceAgent*, the method checks if the *Directory* is not empty or the list of *clusterNeighbours* exist. All the GUIDs referred to as *keysIPS* are recalled to create a new matrix with their weights including the keywords suggested by user in query. Cosine similarity is calculated and the results are stored in *simR* matrix data structure. *simR* is checked to be valid against user provided *minimum_support* parameter before the chosen node is submitted to *ReconnaissanceAgent* agent. (See snippet in Figure 3.13)

3.4.4.3 Query Resolving

In order to resolve a query - it is represented in k dimensional space like a new resource. The set of query keywords are projected on the existing keywords vector and weighted by the k dimensions. The result of computation is a query vector that can be compared with

```
if(clusterNeighbours!=null)
{
    //GET NODE GUIDs
    Set keysIPS = clusterNeighbours.keySet();
    //LOOP TO FORM WEIGHTED NODE_KEYWORD MATRIX
    for(int j=0;j<keysArray.length;j++)
    {
        ...
        double[][] weightsMatrix = weights.getArray();
        //UPDATE WEIGHTS
        for(int k=0;k<weights.getRowDimension();k++)
        {
            if(weightsMatrix[k][0]==0)
            {
            }else if(weightsMatrix[k][0]>=1)
            {
                for(int u=0;u<weightsMatrix[k][0];u++)
                {
                     updated.add(a.get(k));
                }
            }
        }

       ...
    }
    ...
//SIMILARITY RESULTS
Matrix simR=database.getSimMatrix();
double mins = Double.parseDouble(minSup);
...
//CHECK SIMILARITY VALUE AGAINST MINIMUM SUPPORT
for(int q=0;q<simRArray.length;q++)
{
    simVal=simRArray[q][0];
    if(simVal>mins && simVal>temp)
    {
        ...
    }
}
//CHOOSE NODE
ArrayList clientAgents=database.getClientList();
String chosen = (String)clientAgents.get(indexer);
System.out.println("THE CHOSEN ONE IS "+chosen);
//CREATE REPLY TO RECONNAISSANCE AGENT
ACLMessage reply = messagerec.createReply();
if(keywords!=null)
{
    reply.setPerformative(ACLMessage.INFORM);
    reply.setContent(chosen);
}
myAgent.send(reply);
```

FIGURE 3.13: Realisation of node searching and ranking

Algorithm 3.2 Algorithm for transforming query into k dimensional query vector, calculating similarity and ranking resources

$Input$

$A_k = U_k S_k V_k^T$ Locally Shared Resources on Node

$Query:\ Q$

$Perform\, Q^T$

$Perform\, S^{-1}$

$Compute\, Q_k = Q^T U_k S^{-1}$Transformed Query Vector in k dimensional space

Perform Cosine Similarity Test and Ranking

$For\, i = 1\, to\, n$

$Compute\ sim(Q_k, R_i) = \frac{\langle Q_k, R_i \rangle}{|Q_k||R_i|}$

$Perform\, ranking$

$Next$

$Sort\, rank\, based\, on\, minimum\, support$

$Output$

$Return\ sim(Q_k, R_i),\ rank$

other resource vectors in the same k dimensional space. Details of transforming the query to a query vector are provided in Section 3.2.1. Query resolving is performed by *LocalAgent* upon request from *ReconnaissanceAgent*, when mobile agent visits a node. The algorithm for transforming the query into a query vector in k dimensional space is shown in Figure 3.2.

The implementation of algorithm (Algorithm 3.2) is realised through the method *calculateSim()* of private class *LocalDatabase*, used for computing cosine similarity between the query vector and the keyword-resource matrix. The method returns the matrix data structure that contains cosine similarities values for all local resources on a node as compared to query vector. The method computes the numerator that is the sum of product of common keywords. The denominator is computed by products of length of each vector. The ratio is stored in a matrix *simM* and returned to mobile agent RA.

3.4.5 Mobile Agent - Reconnaissance Agent

ReconnaissanceAgent is a mobile agent that is responsible for discovering resources on the overlay network. It is also responsible for migrating from node to node while comparing the search query against hosted resources. Important methods that have been implemented to realise the responsibilities include:

1. *takeDown()*: This method is overridden and implements *doDelete* method for terminating ReconnaissanceAgent.

2. *afterMove()*: This method is overridden and is responsible for finding local LocalAgent and compare the query against the catalog it is keeping. This method is responsible for the following tasks. a.) if any of the results are greater than minimum support, it is responsible for sending *ACLMessage* to its creator (InterfaceAgent) informing about the discovery - name of file and name of LocalAgent hosting it. b.) checks, if it has made number of hops less than maximum number of hops allowed. If the number of hops are less than maximum allowed then it should communicate to local InformationAgent on this node and get the next migration address and container else it kills itself.

3. *commForJump()*: This method implements the steps that required to be performed by ReconnaissanceAgent before migration to new node.

4. *sendRequest()*: This method sends message to AMS for location of the named static agent (InformationAgent, LocalAgent or InterfaceAgent).

5. *setup()*: This method is an overridden and is responsible for getting parameters for ReconnaissanceAgent.

6. *getNode()*: This method is responsible for communication of ReconnaissanceAgent with InformationAgent to get new node where it should migrate in order to perform resource discovery in case number of hops are lesser than maximum number of hops allowed.

This section includes details of realisation of features mentioned in contributions and objectives in Chapter 1. The author has presented algorithms, pseudocodes and implementation details of these features. In addition, the author also presented details of methods implemented by mobile agent in order to realise its functionality.

3.5 Discussion

In flooding-based systems, upon receiving a query, each peer sends a list of all matching resources to the originating node. This results in increase of load on each node that is linearly proportional to the total number of queries. It must be noted that this load will increase with growth in system size making flooding based approach clearly not scalable. To make unscalable systems scalable literature presents DHT-based system that has limitation of search performance because of rigid *key-value* pairing for propagating the query to resource Chawathe et al. (2003). In the proposed system, routing of RA is heuristic based that provides flexible search semantics based on *keyword-node* pairs and supports attaching keywords to shared resources and content-based similarity retrieval thus making it more scalable. Scalability can also be attributed to the proposed resource discovery mechanism that supports exact and similarity search based on *keyword-resource* matrix unlike flooding-based or DHT-based techniques. The author believes that the proposed system provides the necessary flexibility and performance for effective use of LSI for searching and routing on overlay networks.

Furthermore, it must be noted that this implementation has been realised keeping intra-platform mobility in context. In case of inter-platform mobility - the GUIDs will be undermined as container numbers are not unique across multiple platforms. In such case, the author suggests the use of IP address concatenated with agent type and container id to create a globally unique identifier for an agent at global level.

3.6 Summary of the Chapter

The chapter discussed in detail all the design features that implement the characteristics of resource discovery system as understood and informed in Section 2.3. Details of agent communication that include *MessageTemplate* and *MatchPerformative* are described in Section 3.4.1. Furthermore, description of various features, their implementation and the required algorithms have been discussed in Section 3.4. The details about extensive coding have been removed from main report and added to program listings for readers (See Appendix - C).

In next chapter, experimentation is conducted to test the efficiency and effectiveness of Affinity. Also, included in next chapter are tests that compare results from proposed system to current research works. In addition, evaluation of results is provided in detail in following chapter.

Chapter 4

Experiments, Results, and Evaluation

The experiments were conducted to evaluate the effectiveness of proposed method for resource discovery using mobile agents. The experiment is bifurcated into two parts. Part-1 investigates to find out the response time (in secs) that it takes to locate a resource (multiple keywords based query) on an overlay network using RA in MAS as compared to flooding. Part-2 investigates the benefit of using RA for informed search based on LSI as opposed to flooding and other routing algorithm inspired by AntHill (Babaoglu et al. (2002); Babaoglu & Jelasity (2008)) and structured P2P systems by (Dasgupta (2003); Kambayashi & Harada (2009)) by finding out the amount of messages that are on an overlay network.

4.1 Design of Experiments

The design of experiments has been setup in order to compare the proposed technique for content-based resource discovery in terms of heuristic search and search performance. The benchmarks are provided by flooding technique and by term-matching, Jaccard coefficient techniques Chawathe et al. (2003); Crespo & Garcia-Molina (2004); Zhu & Hu (2007); Dasgupta (2008); Kambayashi & Harada (2009). Flooding technique was used as benchmark; as it is widely accepted technique and has been used as backbone for purpose of routing and searching in number of resource discovery techniques including the contemporary techniques as proposed by Dasgupta et al. Dasgupta (2003, 2008). Furthermore, as Dasgupta et al. is using this technique for routing in context to MAS, it becomes all the more important to prove the effectiveness in terms of routing and searching of proposed technique in this context. More contemporary researches from Zhu et al. and Kambayashi et al. have proposed the usage of semantics links based on term-based matching or Jaccard coefficient for resource discovery. Kambayashi et al. uses mobile agent to traverse through overlay network and their technique of preference for matching resources is logical similarity based in Jaccard coefficient Kambayashi & Harada (2009). Kambayashi et al. further uses DHT based structured overlay for migration of mobile agent. Similar approaches has been used in different flavour however (for instance, using DHT for locating nodes, using flooding for routing mobile agent or using term matching for locating relevant results) have been used by many contemporary research works. As Kambayashi et al. is using number of techniques in their approach, the author believes comparing results of proposed method to their technique would provide comparison and evaluation on high degree of intersection of attributes and techniques and a good benchmark. The experiments conducted measure the performance of the proposed method on the basis of following parameters:

- the response time test
- the effectiveness of search technique
- relevance of results
- degree of similarity

4.1.1 Experiment Environment and Test Bed

For comparison to flooding technique as employed by Gnutella, the experimental setup used the open source Java API, JTellav0.7 McCrary & Waters (2000); Forum (2002); Chawathe et al. (2003). This API can be used to create a P2P overlay network and is well documentation on the libraries as well as source code in Java. The setup included 4 peers where 3 peers hosted resources and fourth peer is used for searching resources. Details of each peer including hardware specifications, operating system, IP addresses, number of resources and types of resources is shown in table 4.1.

As seen in table 4.2 total number of nodes participating in Affinity were 10. For the purpose of consistency with benchmark, 4 computers participated in this experiment. In this setup, computer 1 hosted Bootstrap server and 3 computers participated in P2P overlay network. Between these 3 computers, 10 nodes were created, where computer 1 hosted 4 containers hence 4 nodes, computer 2 hosted 3 containers hence 3 nodes and finally computer 3 hosted 3 containers hence 3 nodes. Each container simulated as different node participating in P2P overlay network. The hardware specification of machines is as provided

Peer Name	Peer 1	Peer 2	Peer 3
Operating System	Microsoft Windows Vista Home Premium	Microsoft Windows 7 Home Premium	Microsoft Windows XP Professional Service Pack 2
Processor	AMD Athlon Dual Core QL-62 2.00 GHz	Celeron (R) Dual Core CPU T3000 @ 1.80 GHz	Intel Pentium 4 @ 2.50 GHz
RAM	3 GB	3 GB	512 MB
IP Address	192.168.1.2	192.168.1.5	192.168.1.7
Number of Resources Shared	13	8	8
Type of Resources	8 pdf files 3 docx files 2 doc files	6 pdf files 2 rar files	8 pdf files

TABLE 4.1: Gnutella flooding peers test bed

Specification	Value
Total Number of Computers	4
Bootstrap Server	1
Computers Participating	3
Number of Nodes Participating	10
Maximum Number of Hops	2
Total Number of Shared Resources	27
Minimum Support	0.0

Table 4.2: MAS test bed

in table 4.1. Further specifications regarding MAS and keywords for resources shared are shown in table 4.2 and table 4.3.

Local Agent	Keywords Shared
LA1	sun moon earth mars mercury venus
LA2	moon pluto
LA3	sun stars one two
LA4	one two three four five six mars
LA5	moon
LA6	jupiter saturn neptune pluto
LA7	moon saturn pluto
LA8	two neptune
LA9	pluto earth one
LA10	one sun two moon

Table 4.3: Keywords used for sharing resource on each node

The objective in test 2 is to compare the effectiveness of indexing technique, relevance of results and degree of similarity. The experiments in test 2 used a MEDLINE data set that consisted of 1033 documents University (1999). After removing of stopwords and filtering of nouns, verbs, adjectives, and adverbs, 5735 indexing terms (lexicons) were found. The details of data set can be found on media disc. This data set was used specifically as all resources have already be categorised and attributed with features such as relevance and similarity. The objective was to find out of the proposed technique provides similar results and then to compare the results with techniques used by other research works. Hence, the prepared results served as benchmark for comparing the effectiveness of proposed technique to other relevant works.

4.2 Test 1 - Comparison to Flooding Technique

4.2.1 Experiment 1 - Response Time and Evaluation

The objective of experiment 1 was to calculate the response time for query on an overlay network. The performance metric response time is defined as the time elapsed between a user initiating a request and receiving the results. This includes the time taken for agent creation, time taken to visit the node and the processing time to extract the required information. Once the response time is available it can be concluded that which method is more effective with respect to amount of time it takes to locate a resource on an overlay network. It is observed from the bell curve that amount of time it takes to find a particular resource in proposed method is consistent and ranges between $5s$ to $6s$. Flooding however does not have any consistency in response time. It is observed from bell curve shown in Figure 4.1 that response time can vary from few seconds to few minutes. Furthermore, it is observed that in flooding 14 queries out of potential 28 queries has response time of $< 5s$ which approximates to 42% of total number of queries, where when using proposed method the author observed that 67% of queries were replied with resource location in $< 5s$ and 23% of queries replied in $< 6s$.

It is evaluated that overall response time, when using the proposed method is lesser than the case of Gnutella using flooding technique. But it should be noted that a lower response time does not measure the effectiveness of search technique in terms of successful results as described further in Section 4.2.2. The author concludes that lower response time is attributed to mainly two reasons. Firstly, as Gnutella is pure P2P network, it is required of participating peers to communicate their status using PING and PONG messages on the overlay network. This results in high amount of traffic on overlay network and results in saturation. It is observed, as mentioned in Section 4.2.3, that PING and PONG activity together amount to 97.5% of messages. This results in latency and hence low response time. Secondly, as resources to be located are searched based in multiple keywords do not always

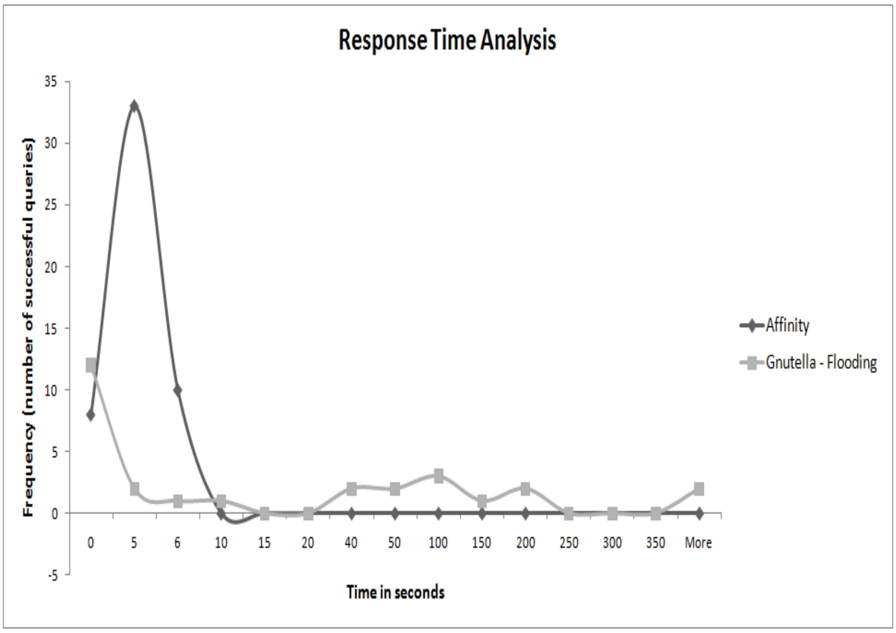

FIGURE 4.1: Frequency distribution of response time analysis - Gnutella vs. Affinity

match the file name of resource to be located, it amount to low response time as query may not match the resource completely.

4.2.2 Experiment 2 - Effectiveness of Search Technique and Evaluation

In experiment 2, the objective was to investigate the effectiveness of search using proposed method as compared to flooding. For achieving this objective, the experiment setup was to compare successful queries to unsuccessful queries. It was realised through experiment using proposed method that out of 30 queries, 24 responded with query hit, 4 queries did not have any response, and 2 queries replied as NaN network (See Figure 4.2).

Furthermore, it was realised that NaN is due to explicit specification of *minimum support* parameter as 0.0. The nodes in similarity with 0.0 did not host the content that was required by user. In case, of flooding, 26 queries were passed through various nodes. 32% of queries had query hit and 68% of queries failed (See Figure 4.3).

4.2.3 Observations

Dasgupta (2003); Kambayashi & Harada (2009) has confirmed that no matter how many peers or resources are there on an overlay network, the flooding technique generates a con-

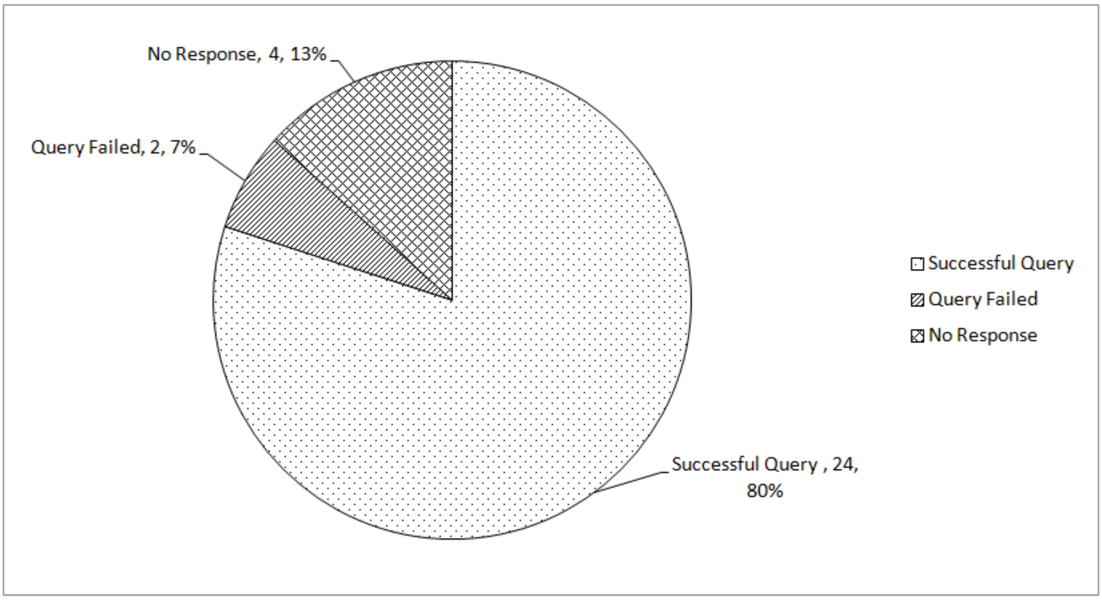

FIGURE 4.2: Query successful vs. unsuccessful - Affinity method

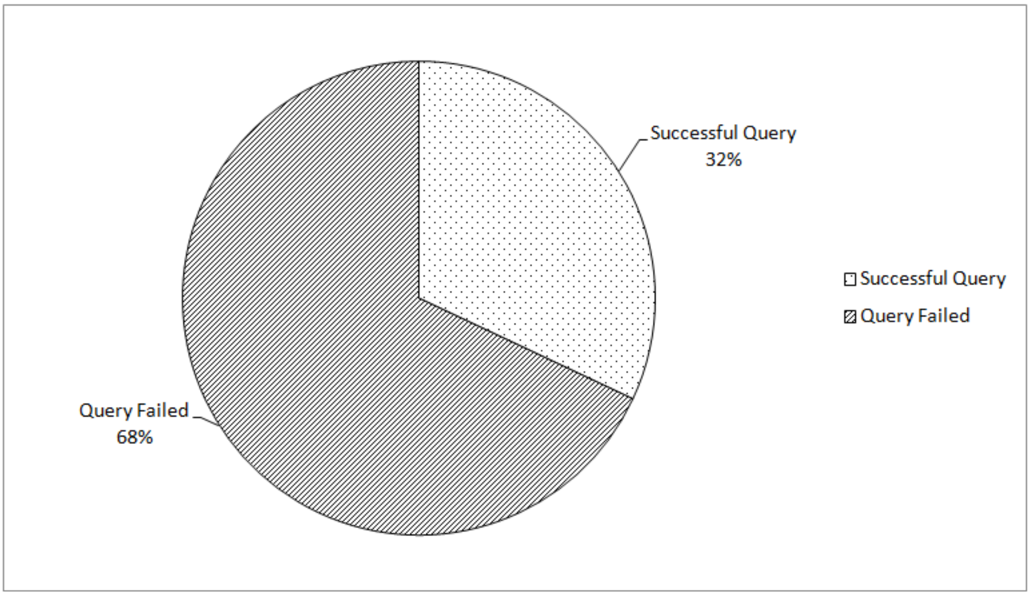

FIGURE 4.3: Query successful vs. unsuccessful - flooding method

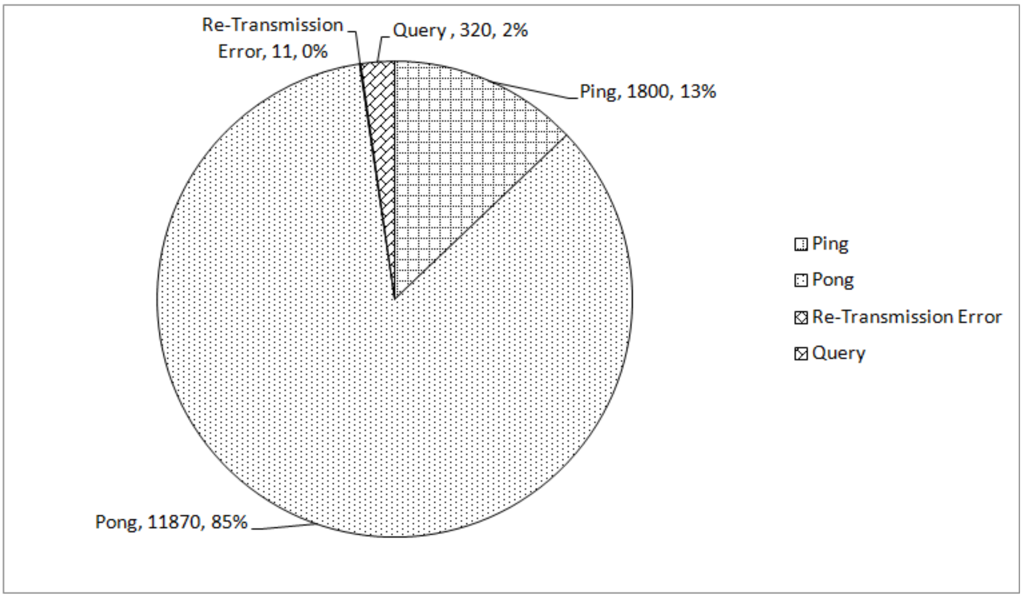

FIGURE 4.4: Division of packets for Gnutella

sistent number of messages on an overlay. The author has observed in flooding that the amount of traffic or messages on an overlay network or even response time increase or decrease is attributed to mainly the PING and PONG activity. This continuous stream of messages is produced by the peers to check existence and current status of other peers. The author used Wireshark to monitor the Gnutella packets Wireshark (2010). The screenshot in Figure shows Gnutella packets upon filtering. A total of 14008 Gnutella packets were analysed when overlay network was subjected to 2 queries. It is clear from the pie chart (Figure 4.4) that 84.7% of traffic is related to PONG descriptors, 12.8% to PING, 2.22% to QUERY, and 0.07% to re-transmission errors. Gnutella connections are relatively unstable, which lead the nodes in iterative effort for discovering other nodes on overlay network as opposed to nodes joining and leaving network autonomously.

It is also observed from the graph in (Kambayashi & Harada (2009)) that no matter what is the number of resources shared, as long as number of peers is constant the number of messages (bytes) will stay constant.

4.2.4 Critical Analysis

However, this raises another issue of why there is a decrease is number of messages also claimed by Dasgupta (2003); Kambayashi & Harada (2007, 2009). The author observed and

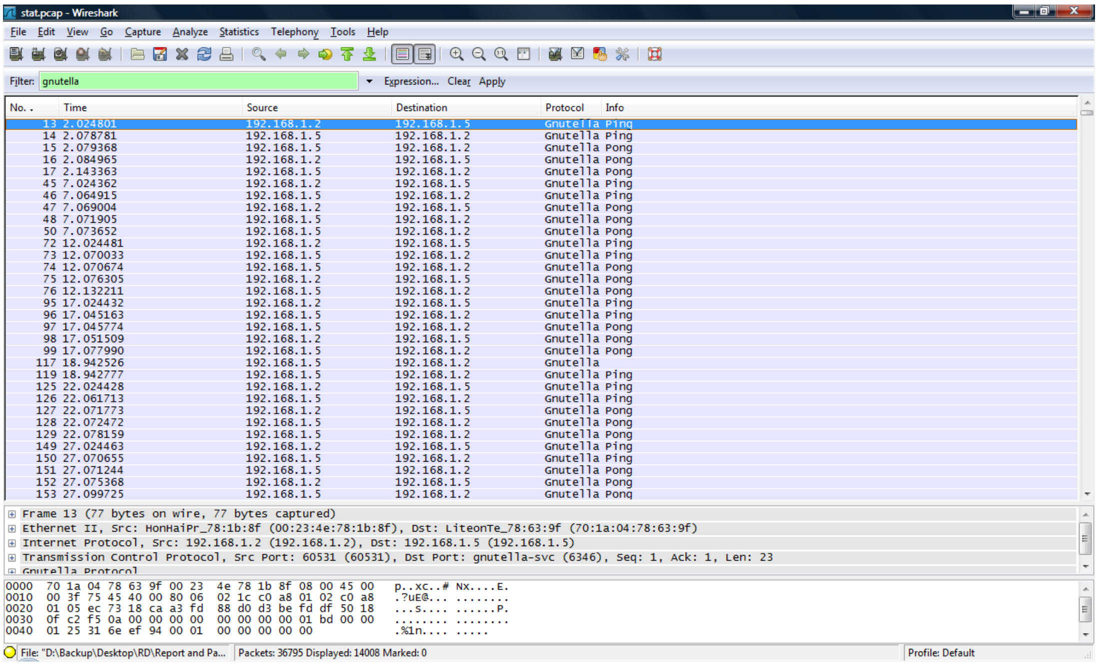

FIGURE 4.5: Gnutella packets analysed using Wireshark

evaluated that the decrease in number of messages in these multi-agent systems is due to decrease in number of hops to locate a resource. Kambayashi & Harada (2009), claims that number of messages on overlay network will decrease with increase in number of resources. This is because the overlay network has become more resourceful and hence almost all peers have links to other peers, which means that when the SA enquires from directory services on NA about peer to migrate to, it is capable of informing SA about the highest possible logical distance value because of its resourcefulness. This is observed in proposed method too and the author agrees with Dasgupta (2003); Kambayashi & Harada (2009). It is evaluated in Section 4.3.1, that number of inter cluster links are on average higher than case where, logical distance value was used to create semantic links between nodes. More number of links makes the overlay network more resourceful thus reducing number of hops and reducing number of messages on network. Furthermore, it is evaluated through precision-recall results where the author defines precision as the ratio of number of relevant resources/nodes found during search to number of search results and recall as the ratio of relevant resources/nodes found to total number of relevant resources/nodes in corpus. Though, number of inter-cluster links are higher that may result in compromise of precision, however, we achieve higher

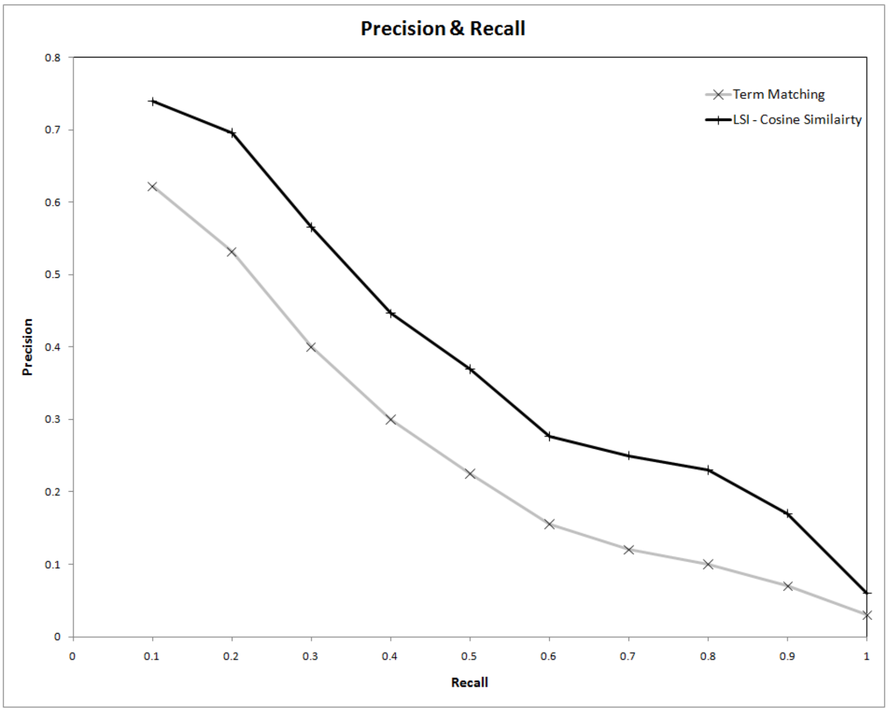

FIGURE 4.6: Precision and recall results comparing LSI to TF-IDF indexing model

recall making degree of relevance higher (See Figure 4.6) due to efficient indexing technique. Together, with results from reduced response time, higher recall and greater number of successful queries it can be concluded, that lesser number of messages exist on network.

In test 2 - Section 4.3, the author investigates the effectiveness of their techniques/algorithms to reduce number of messages and compare them to proposed method.

4.3 Test 2 - Comparison to Other Routing Techniques/Algorithms

The aim of this test is to investigate the effectiveness of the routing mechanism employed by Kambayashi et al. that calculates the logical distance between the nodes based on the resources shared by that node as compared to LSI based clustering of nodes and routing based on calculation of cosine similarity between search query and the lexicons shared by nodes. This experiment also indirectly studies the effect on amount of message on overlay network. Replicating exact environment as used by Kambayashi et al. has been a tedious process as they are using Overlay Weaver and Agent Space both tools developed by them and changed to accommodate messaging between agents through Overlay Weaver Kambayashi

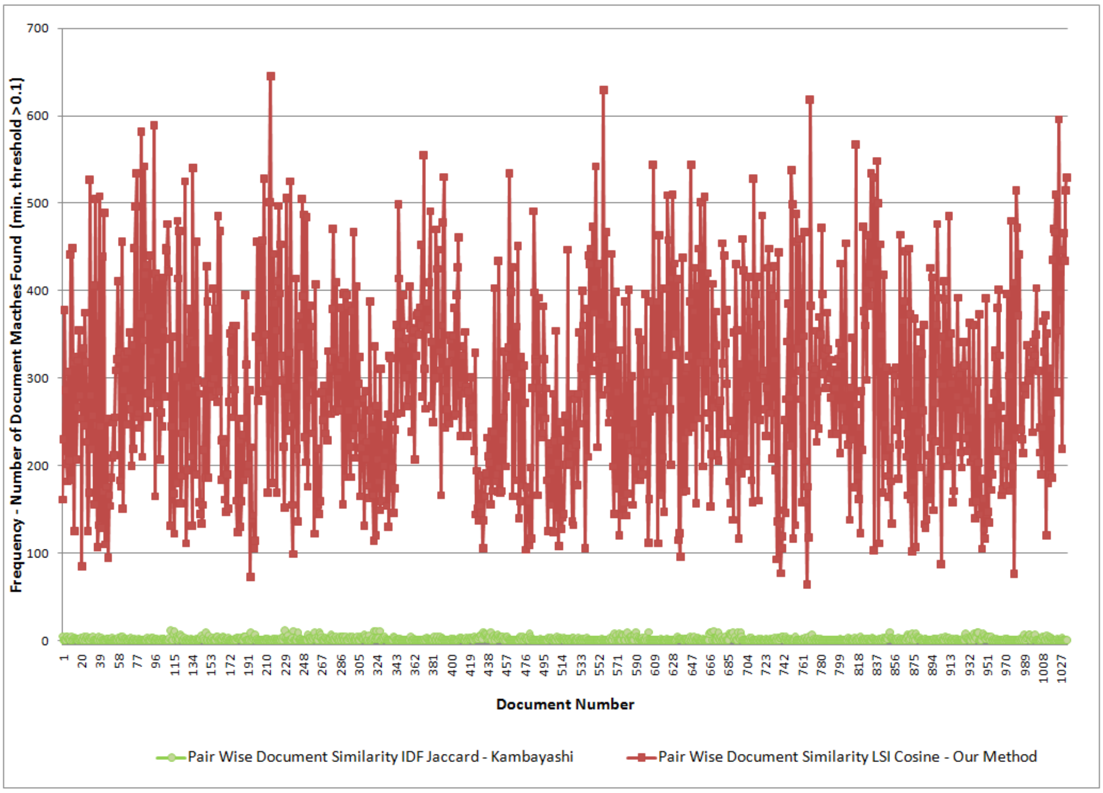

FIGURE 4.7: Pair-wise document similarity TF-IDF Jaccard vs. LSI Cosine

& Harada (2009).

Though, the author evaluated their results and observed that the decrease in number of messages is due to increased similarity score (Jaccard Similarity) between shared lexicons. In this context, the author designed another experiment that would compare their indexing and routing algorithm to the proposed method by comparing its effectiveness on third party data provided by University (1999). The effectiveness was evaluated in different experiments.

4.3.1 Experiment 1 - Pair-Wise Document Similarity And Evaluation

In experiment 1, pair-wise document similarity is investigated by comparing Jaccard similarity (subset used by Kambayashi & Harada (2009)) and Cosine similarity (used by proposed method Singh et al. (2009)). In case of Kambayashi et al., the test required normalising the term-document matrix using term-frequency and inverse document frequency indexing (TF-IDF) for measuring Jaccard similarity Kambayashi & Harada (2009). In proposed case, the test required creating the normalised latent-semantic indexed matrix for measuring Cosine similarity as described in Section 3.2.1. The effectiveness in experiment 1 is studied by finding out number of documents that match where the minimum threshold is > 0.1. The result of number of document will indicate the resourcefulness of overlay network, as that is used to cluster the nodes. In other words, more is the number of matched documents, larger is the cluster, and more are the chance for mobile agent to locate a resource which would mean lesser number of migrations for mobile agent and hence, less number of messages on overlay network. The author, evaluated from the following graph (Figure 4.7) that using LSI and cosine similarity, clearly has larger number of pair-wise matches, between documents and hence, provide larger cluster and links between clusters.

It is evaluated that larger is a set of similar documents, more resourceful is the overlay network, hence lesser number of hops are require by RA to locate a resource. The pair-wise documents similarity is large in case LSI technique used in proposed method, hence number of messages required by RA to locate a resource will be lesser and in this case much lesser than flooding (Aberer et al. (2004); Chawathe et al. (2003)) and logical distance method (Dasgupta (2003); Kambayashi & Harada (2007, 2009)) making proposed method for routing RA through overlay network more efficient in terms of time and bandwidth consumption.

4.3.2 Experiment 2 - Effectiveness of Search Technique And Evaluation

In experiment 2, the aim was to investigate number of documents that found to be similar in to search query. Large number of documents effectively indicate:

1. Large number of nodes for the RA to migrate to for locating resources

2. Better inter-cluster link for routing the RA through overlay network.

It is highly important that mobile agent can traverse through overlay network for locating the resource.

If routing links cannot be established between clusters - it would indicate:

1. Mobile agent cannot locate a resource because of its incapability to migrate to different clusters or

2. Mobile agent will provide results that are less precise.

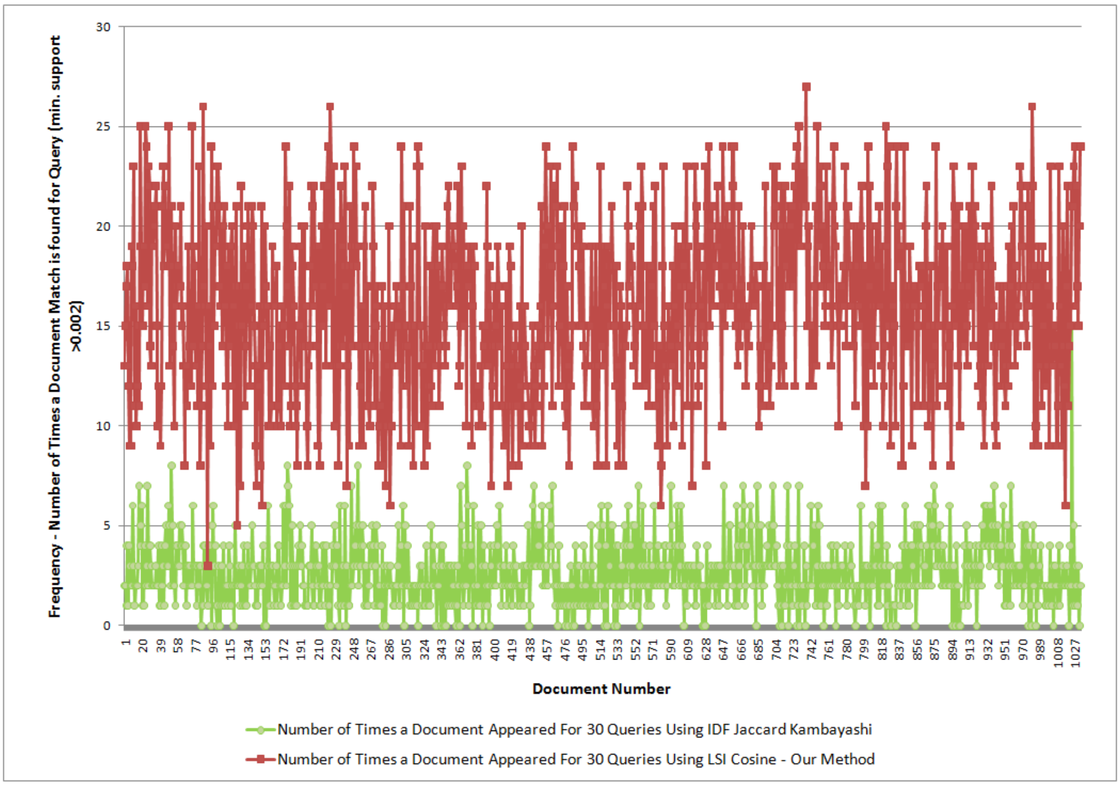

FIGURE 4.8: Number of times a document appears for 30 queries Jaccard similarity vs. Cosine similarity

It must be noted that larger number of matches also mean large number of nodes to be visited by the RA hence more number of message on overlay network which in effect means higher bandwidth consumption. This however is controlled in proposed case by introduction of factor called *minimum support* as mentioned in Section 3.2.1, that is set by user to reduce the number of selected nodes for the RA to visit. The author conducted similarity test on corpus of 1033 documents by subjecting them to 30 different queries University (1999). The following graph (Figure 4.8) was obtained as a result of this experiment, informing number times matched documents is found for 30 queries where *minimum support* is > 0.002.

It is observed that proposed method is resulting in larger inter cluster links and also large number of nodes where the RA can potentially visit as compared to logical distance method used by Kambayashi & Harada (2009).

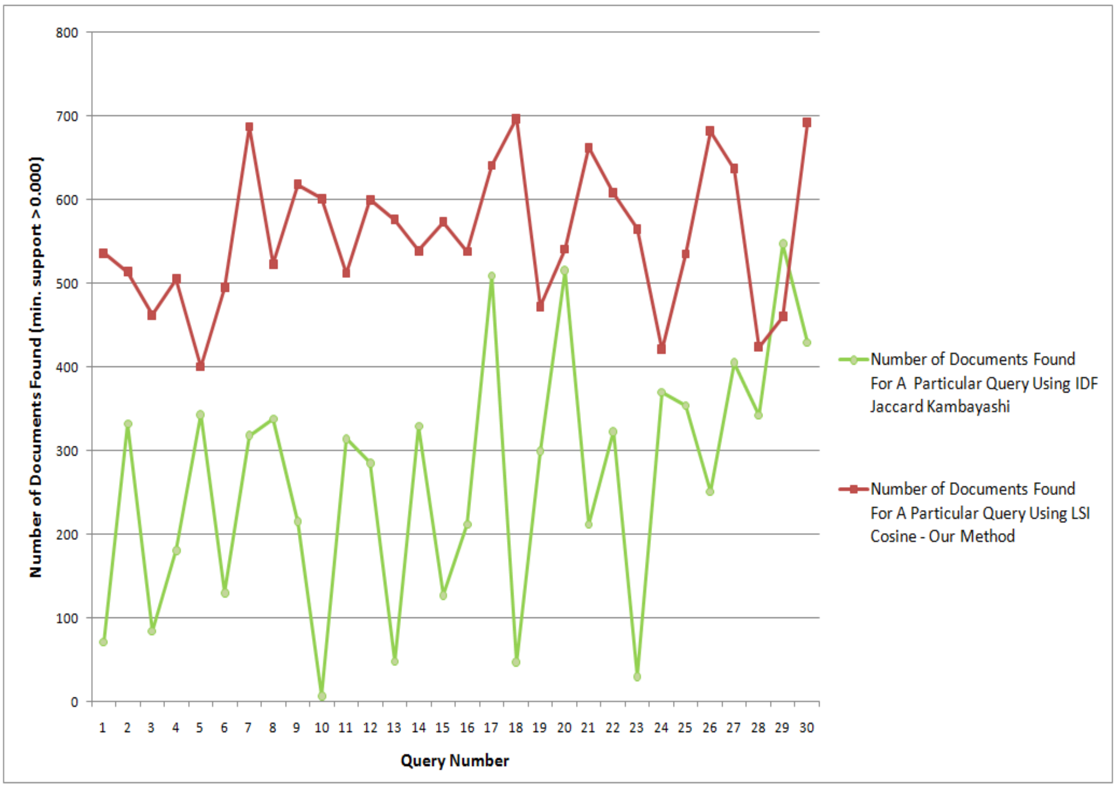

FIGURE 4.9: Number of documents found for 30 separate queries on corpus of documents

4.3.3 Experiment 3 - Effectiveness to Locate Resources and Evaluation

In experiment 3, the aim was to investigate effectiveness of proposed method to locate the resource. Keeping that in context, in general terms it means - number documents found per query using proposed method as compared to the logical distance method. Similar to experiment 2, for achieving the aims of this test, the document corpus was subjected to 30 queries and number of documents found per query were obtained for minimum support > 0. This number was compared for LSI based Cosine similarity and TF-IDF based Jaccard similarity. The graph (Figure 4.9) shows that number of documents using LSI Cosine method used in proposed method is higher than TF-IDF Jaccard method. It is further evaluated, that a larger number of documents associated with a query means 1. higher cluster links 2. larger set of relevant documents found as part of resource discovery. Of course, as mentioned in experiment 2, larger set of documents can also indicate irrelevant information, but this can be capped using parameter *minimum support* as mentioned in section 3.

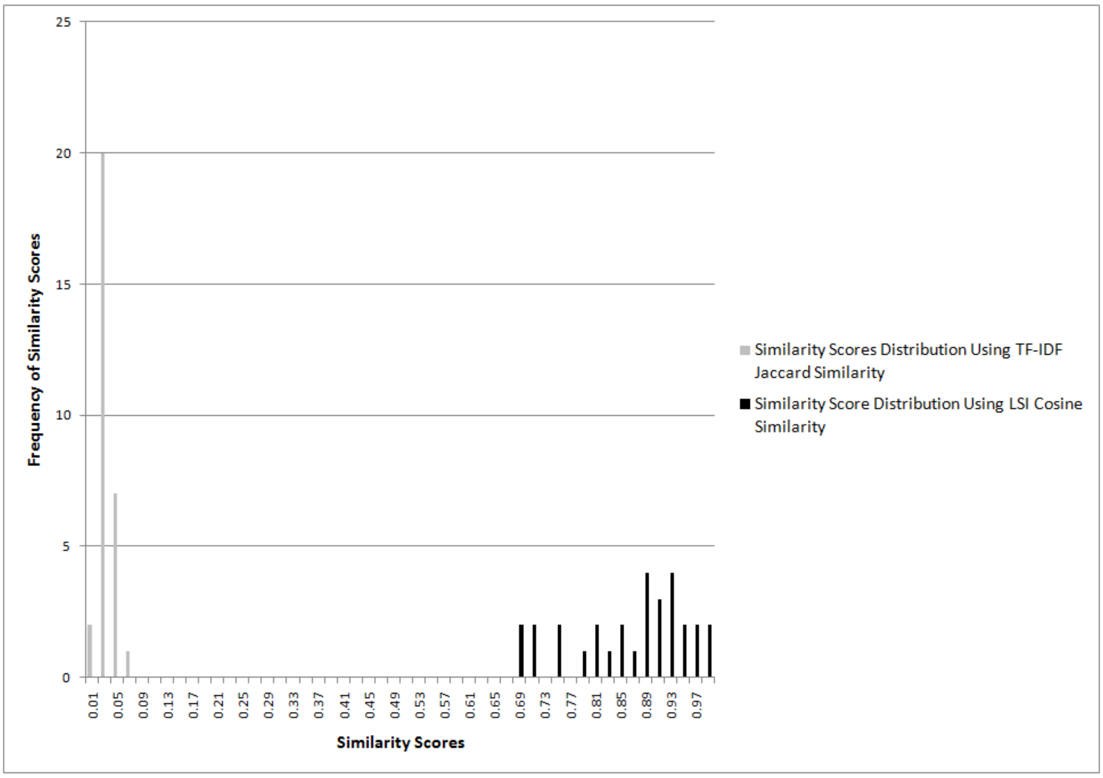

FIGURE 4.10: Similarity score distribution TF-IDF Jaccard vs. LSI Cosine

4.3.4 Experiment 4 - Degree of Relevance of Results and Evaluation

In experiment 4, the aim was to investigate the degree of relevance of results obtained during search process by the RA. Again, similar to experiment 2, the corpus was subjected to 30 queries to find out about similarity scores. The highest similarity score obtained is assumed as resource that is best match to a given query. The objective was to collate the highest similarity scores and find their frequency distribution. This process would:

1. offer insights into relevance of results
2. inform which method is capable of extracting best match documents.

Perhaps, if the same document is found a result of search, using both methods, if logical distance is low, it may safely be assumed that mobile agent may take more time or even more number of hops to reach the node.

It is observed from the graph (Figure 4.10), that using proposed method the similarity scores tend to be on higher end of frequency distribution as opposed to other research works. This indicates that it is of utmost importance the similarity scores are high which would effectively mean fewer messages on overlay network and better response time.

4.4 Discussion

In order to test the proposed method for content-based routing of mobile agents using LSI on large scale network, the literature offers only a few simulation environments.

A simulator called Swarm is a general purpose software package for simulating, distributed artificial worlds written in Tcl Lingnau & Drobnik (1999). It is particularly useful for large number of autonomous entities ("agents" – not to be confused with "mobile agents") with an environment. Using this, the global and adaptive behaviour of the proposed system can be studied Lingnau & Drobnik (1999). Anslem Lingnau et al. Through their research offer an extension to Swarm system by including infrastructure for mobile agents. Their extension, allow mobile agent collaboration other agents and also allowing for computations and migrations Lingnau & Drobnik (1999). Included, in this simulator are some routing techniques for studying network load and response time for agent to complete a given task. This environment is suitable for simulating the proposed technique, as long as it allows new routing techniques to be added. One of the drawbacks of this environment is their non standard use of messaging techniques by use of invocation of remote methods rather than standard agent communication language. This will prevent accurate results with regard to response time and efficiency of network usage.

Another simulator that is written in Java and has been used by some researchers for agent-based simulations is Repast North et al. (2006). Due to object-oriented nature of the underlying programming language, it supports computational elements that make agent autonomous (an important characteristic required for agents) Bandini et al. (2009). Furthermore, object oriented nature of Java offers encapsulation of state, actions and action choice mechanism in agent's class. It also simplifies integration of external APIs such as JADE in this case. This simulation platform does not specifically support realisation of agents and interaction models as standardised by FIPA.

AgentSim developed by IBM has been used by researchers as simulator for simulation of agents Trillo et al. (2007). The simulator is library built only for Aglet - agent development platform Trillo et al. (2007). As mentioned in Chapter 2, Aglet does not support ACL, instead only offer synchronous remote method invocations that are not favourable for simulating proposed technique Trillo et al. (2007).

Chen et al. developed Mobile-C that conforms to the FIPA standards both at agent and platform level. It also extends FIPA standard to support mobile agent protocol to direct agent migration process. Agent migration is achieved through FIPA-ACL messages encoded in XML Bo Chen (2006). FIPA ACL is effective way for inter-platform agent migration in FIPA compliant Agent systems as both agent communication and migration share the same communication mechanism. The development of simulator is done in C or C++ which makes inter-language barrier for communication, as developments have been done in Java and JADE API. However, the author believes that using CORBA, for inter-language communication can be conceived for successful simulation. This may require extensive writing of interfaces for the developed system and various computational models.

The author understands the issue to test scalability of system on large scale network is important, which will be created as part of simulation in further work. The author believes that Mobile-C offers promising simulation platform for simulating the proposed system on large scale network.

4.5 Summary of the Chapter

In this chapter, the author has provided details of various experiments, that were conducted and describe the characteristics of the resource discovery system, using mobile agent - Affinity as well as provide insights into one-on-one comparison with other routing techniques used in older and current research works. The author also discussed the choice of simulators, their features and drawbacks for large scale mobile agent based simulation.

In next chapter, the author provides discussions on concepts provided by researchers and compare them to concept listed by proposed work by benchmarking characteristics of P2P and resource discovery systems using mobile agent.

Chapter 5

Discussion, Conclusions, and Future Work

This chapter discusses other related research works and critically analyse the concepts presented by them. The results obtained as part of experimentation in Chapter 4 are promising and the author believes further discussion of the concepts presented by some of the related works as collated in literature survey is useful for readers.

The works done by Zhu et al ESS, Dasgupta et al, Kambayashi et al, and Crespo et al are related to this research work for development of P2P system for resource discovery and the first sections of chapter provides related discussions Dasgupta (2003); Crespo & Garcia-Molina (2004); Zhu & Hu (2007); Dasgupta (2008); Kambayashi & Harada (2009). In ending sections, the author has collated the future works, that can be undertaken and can be potentially useful with this research work in context. Also, the conclusions have been provided.

5.1 Discussions - Analysis of Other Research Works

In this section, the author discusses related works similar to conducted research work undertaken.

Crespo initially presented the idea of routing indices for controlling the amount of flooding and saturation of overlay network Crespo & Garcia-Molina (2002). The concept however suffered from maintenance of distributed-index on various nodes that itself generated it own large amount of traffic.

Later, Crespo *et al.* introduced the idea of semantic overlay networks (though not in a P2P context) where the nodes can be clustered to form an overlay network Crespo & Garcia-Molina (2004). Crespo *et al.* use explicit term semantics to building routing indices

Crespo & Garcia-Molina (2004). They assign documents with terms indicating related realms, and maintain in each peer a statistic table containing term-based routing indices, which indicates how many documents would be found, if probes the query of that term to a neighbour peer Liu et al. (2004).

The author, understands that Crespo et al brought improvement to searching but as most latent semantics analysis proved, only terms-based statistics cannot fully capture resource characteristics as terms also have underlying correlations and semantics Deerwester et al. (1990); Liu et al. (2004). The author has been inspired from the idea to form relationship between nodes but proposed system uses these relationships for coordination of resources that are managed by nodes and further use it for informed routing of the RA.

Zhu *et al.* presented the use of information retrieval from unstructured and structured P2P system by use of semantic links between the nodes Zhu & Hu (2007). The query flooding on P2P network is controlled using routing based on Jaccard similarity technique. However, as described in tests the results obtained from normalised LSI based cosine similarity technique are far superior on terms of number of document matches and higher similarity scores. Furthermore, their system is not a mobile agent based resource discovery system which as mentioned in literature greatly improves upon the classical unstructured and structured P2P system. Proposed work contributes towards the dynamic organisation overlay network based on resources published by nodes. The relationship between nodes and resources for guidance of agent (direction) on overlay network is central and crucial.

Dasgupta *et al.* (Dasgupta (2003, 2008)) research work is greatly inspired from Babaoglu et al work on Anthill in Babaoglu et al. (2002); Babaoglu & Jelasity (2008). The author here presents analysis of Dasgupta's research work as they have used MAS.

Dasgupta *et al.* introduced the used of mobile agents for P2P resource discovery Dasgupta (2003, 2008). Their system is based on referrals made by search agents. Clearly, in their system the behaviour of search agents evolve and get better, based on the trails established by searches done before. In contrast to proposed work, they do not use the routing tables for guiding the search agent through the overlay network as done in proposed work using directory facility made during initial registration of peer on bootstrap server. Furthermore, they did not introduce the use of peer-keyword semantics to form clusters of semantically similar peers. Clearly, they are using the classical technique of flooding to discovery resources that improves over time based on the search trails left by previous searches.

Kambayashi *et al.* has provided method of resource discovery by using mobile agent and DHT Kambayashi & Harada (2009). Like proposed work, their work also overcomes, use of flooding for finding resources using node management table on each node (similar to directory service on InfA). However, the node management table is constructed by calculating logical similarity of keywords on peers based on primitive form of Jaccard similarity function as opposed to using latent semantics of keywords and finding cosine similarity in our case. Inspired from Crespo *et al.* (Crespo & Garcia-Molina (2004)) Kambayashi *et al.* (Kambayashi & Harada (2009)) also used the terms to capture the realm of resources shared. However, as mentioned before, matching only terms to cannot capture resource characteristics, which is where the author introduced the idea of using latent semantic analysis. In proposed case, the author has introduced the use of minimum support for peer discovery and latent semantic indexing between peers to direct the RA towards resource.

Inspiring from Dasgupta's (Dasgupta (2003, 2008)) and Babaoglu *et al.* (Babaoglu et al. (2002); Babaoglu & Jelasity (2008)) work, Kambayashi *et al.* (Kambayashi & Harada (2009)) introduced the use of pheromone value (AntHill) (that is calculated taking parameters such as number of resources shared by peer and clustering value (logical distance between peers). This feature is expected to guide search agents towards nodes with high correlation by reducing free-riders. The author believes both, the techniques are equally credible, however the work from Kambayashi *et al.* is discriminating free-rider which may hold a resource that is relevant Kambayashi & Harada (2009). The aim in this work has been to create harmony between nodes and relevance of resources to user's query. The author believes that if resource is available it should be locatable. Finally, they also used DHT - Chord structured P2P system for resource discovery, which the author believes is interesting but opposes the original aim that DHTs cannot handle queries that are multi keyword or text based and is also only viable when keyword for finding resource is known exactly.

Kambayashi *et al.* techniques i.e. guiding search agents using pheromone values and DHT for resource discovery may be leaving "ill-effect" Kambayashi & Harada (2009). In former case, the credible peer by removing free-riders from list of peers that may be holding a resource and in latter case to direct the search agent towards exactly known resource keyword. They are undermining the level of ambiguity and introducing too much certainty into searches which is not the case in proposed system, where user can increase or decrease the search ambiguity/certainty by changing value of *minimum support* thus providing bigger/smaller "canopy" for movement of RAs.

One of the more recent works has been presented by Tan & Zheng (2009). This work offers resource discovery solution, but has no indication of using semantic links for routing the mobile agent. The solution offered seems to be in its earlier stages, implying that all characteristics required by resource discovery system are not answered yet. Though, from this early work it is indicated that there solution also seems to be implemented using FIPA standards.

Other classical work from Dimakopoulos *et al.* has indicated the use of mobile agent as architecture for resource discovery but there synchronisation for distributing local directory (information about shared) resources is done using classical method of flooding, that clearly implies bottleneck of bandwidth limitations and hence saturation of network Dimakopoulos & Pitoura (2003); Chawathe et al. (2003).

5.2 Applications for Research Conducted

Following are examples of few applications that can be developed as a result of this research:

1. Organisation of Documents with Reviewers: Hundreds of documents are submitted to publishers for conference or journal publications that need to be reviewed by the reviewers for finding the worthiness of those documents for that specific conference or journal. The task of matching the documents with reviewers based on their research skills is time consuming and tedious. The outcome of this work can be used by the reviewers to setup their profiles and submit them to the publishers. Upon receiving of documents, the publishers can match, create the keyword list based on content of the document which, when submitted as query will find the appropriate reviewer. So, instead of node-keyword matrix in this application reviewer-keyword matrix will be calculated. This system will perform efficiently and at the same time will offer high degree of effectiveness in terms of find appropriate reviewers.

2. Content-Similarity Check: The purpose of such system ranges from targeted e-marketing to creating clustered documents to comparing two or more documents for similarity. In an e-marketing system e.g. the content of email being received by user can be matched against target advertisements that are of similar domain as the content of the email. In clustered documents, documents belonging to same concept/domain can be organised and furthermore checked for similarity among each other. As the core of

this system is based on LSI-SVD on an overlay network, these services can be extended to large number of nodes.

3. Searching and Locating of Resources: It is not always possible to locate a hosted resource using indexing techniques such as, used by Google. It takes time for web crawlers to scan the newly published website and rank it, resulting into null response if such resource cannot be located. On an overlay network, such as one that is powered by mobile agents, the query initiator need not filter the results obtained to find the suitable resource, once the criteria such as *minimum support* has been provided and that the resource provider is participating on an overlay network, mobile agents can locate a resource dynamically without requirement of web crawlers etc.

The author is sure that there can be many other applications where this system can be applied and implemented. The final product is only limited by a conceivable idea.

5.3 Conclusions

The main objective was to design and implement a resource discovery system that uses mobile agent technology for discovering and selecting nodes and for routing the mobile agent through overlay network based on content of query with purpose of minimising response time, reducing possible delays, maximising network performance by reducing the possibility of saturation and maximising the recall by providing relevant results. Through the conducted research work and the evaluations of experiments in Chapter 4, the author concludes that the process of resource discovery can be improved for P2P system in terms of search performance by increase of recall and hence success rate to resolve queries through use of efficient indexing technique viz. LSI and also that the routing of mobile agent to resolve query through overlay network when supported by heuristics viz. offered using clustering technique will reduce saturation due to higher number of inter-cluster links and decrease response time. The author believes that this resolves the original research question mentioned in Chapter 1.

To summarise the author has proposed a novel resource discovery system that uses mobile agent (RA) for discovering resources on an overlay network that is realised based on semantic similarity of keywords that are shared by peers. The author further proposed a flexible multi-agent based approach to P2P network organisation that is based on the similarity of content shared by peers. The author claims that the use of semantic similarity between content shared by peers i.e. clustering effect can be effective technique to route the RA to peers that host content that is similar to a user query and finally, that LSI based resource search by RA to find resources hosted by peers that are best match for a user query (where the user query can be text based or an approximate query) is very effective whether the query contains text, that is certain or ambiguous.

The author further demonstrated that proposed approach for resource discovery is better than flooding and further more that an informed search technique used to guide RAs on an overlay network is better than controlled flooding. The results have demonstrated that the using flooding increases the quantity of messages on a network and it can be reduced by use of proposed technique.

In previous experiments, the author used flooding technique to find resource on the network i.e. the RA migrated from one peer to another in hope of finding the resource Singh et al. (2009). The author has realised the shortcoming of last technique and introduced the use of guidance directory on each peer for providing the RA with better chance of finding a resource.

The author realises that initially as resources are scarce, some clusters may not overlap, resulting into cases where resource cannot be located, but the author does understand that as the peers become more resourceful, the clusters will start overlapping to higher degree, hence resulting into better search results.

5.4 Future Work

Although, in ideal case the RA can migrate to suitable nodes and query them for resource, the aspect of breach of security has not been researched in this project. Agents are open to security lapses and hence can be compromised about what to search or what to deliver as result back to query originator. This can jeopardise the integrity of results as well as the RA. Furthermore, the compromised InfA where agent queries about routing for next node for migration can guide the RA migrate towards nodes that do not hold any relevant results. This is an area of future research work that requires attention.

As mentioned before in Section 3.2.1, *keyword-peer* and *keyword-resource* matrices can be large sparse matrices. Holding these large matrices consumes memory which is not always abundant on systems that are continuously publishing or are dynamic. Dimensionality reduction used in proposed work offers a solution to some extent i.e. reduction in matrix size of an order of around 40%, but that can still be a large matrix. Some research works have been done in this field but are out of scope for this work. Further work can be done in this project to accommodate for this characteristic. The author believes that system architecture presented is very generic and can be further refined in order to support distributed LSI where by the indexing could be decentralised and global search can be conducted for relevant resources on pure P2P overlay network. The problem to generate globally-consistent LSI structure is very challenging as the number of nodes presented by their content is large, dynamic and distributed.

Some research work has been done where local cache is maintained by node to guide the visiting mobile agent so as the computational load for calculating node for migration can be bypassed. It is an interesting feature and can indirectly find its roots in Anthill system used by Babaoglu et al. (2002); Dasgupta (2003); Babaoglu & Jelasity (2008); Kambayashi & Harada (2009). But cache is not always up-to-date and hence can lead to incorrect decisions for migration of mobile agent. In case, the cache can synchronised periodically, this feature can be potentially useful. However, it must be noted that synchronised cache may lead to flooding that increases number of message on network. This area can be studied further to find out its cost-to-benefit ratio.

The author believes that conducted research has far greater potential and can still form foundation for future research work.

References

Aberer, K., Punceva, M., Hauswirth, M., & Schmidt, R. (2004). Peer-to-peer systems. In *In practical handbook of internet computing.* CRC press.

Androutsellis-Theotokis, S., & Spinellis, D. (2004, December). A survey of peer-to-peer content distribution technologies. *ACM Computing Surveys*, *36*(4), 335–371. Available from `http://www.spinellis.gr/pubs/jrnl/2004-ACMCS-p2p/html/AS04.html`

Arabshian, K., Dickmann, C., & Schulzrinne, H. (2009). The semantic web: Research and applications. In (p. 684-696). Springer Berlin / Heidelberg.

Babaoglu, O., & Jelasity, M. (2008). Self-* properties through gossiping. *Philiosophical Transactions of the Royal Society A*, *366*, 3747-3757.

Babaoglu, O., Meling, H., & Montresor, A. (2002). Anthill: A framework for the development of agent-based peer-to-peer systems. In *Ieee proceedings of 22nd international conference on distributed computing systems (icdcs'02)* (pp. 15–22).

Bandini, S., Manzoni, S., & Vizzari, G. (2009). Agent based modeling and simulation: An informatics perspective. *Journal of Artificial Societies and Social Simulation*, *12*(4), 4. Available from `http://jasss.soc.surrey.ac.uk/12/4/4.html`

Bawa, M., Manku, G. S., & Raghavan, P. (2003). Sets: search enhanced by topic segmentation. In *Sigir* (p. 306-313).

Bellifemine, F. L., Caire, G., & Greenwood, D. (2007). *Developing multi-agent systems with jade.* Wiley.

Bo Chen, J. P., Harry Cheng. (2006). Mobile-c: a mobile agent platform for mobile c/c++ agents. *Software: Practice and Experience*, *36*, 1711-1733.

Casey, J., & Zhou, W. (2009). Reducing the bandwidth requirements of p2p keyword indexing. *International Journal of High Performance Computing and Networking*, *6*, 119-129.

Chawathe, Y., Ratnasamy, S., Breslau, L., Lanham, N., & Shenker, S. (2003). *Making gnutella-like p2p systems scalable.*

Chen, H., Jin, H., Liu, Y., & Ni, L. M. (2008). Difficulty-aware hybrid search in peer-to-peer networks. *IEEE Transactions on Parallel and Distributed Systems*, *20*, 71-82.

Crespo, A., & Garcia-Molina, H. (2002). Routing indices for peer-to-peer systems. *Distributed Computing Systems, International Conference on*, *0*, 23.

Crespo, A., & Garcia-Molina, H. (2004). Semantic overlay networks for p2p systems. In *Ap2pc* (p. 1-13).

Dasgupta, P. (2003). Improving peer-to-peer resource discovery using mobile agent based referrals. In *Ap2pc* (p. 186-197).

Dasgupta, P. (2008). A multiagent swarming system for distributed automatic target recognition using unmanned aerial vehicles. *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, *38*(3), 549-563.

Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K., & Harshman, R. (1990). Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, *41*, 391–407.

Dimakopoulos, V. V., & Pitoura, E. (2003). A peer-to-peer approach to resource discovery in multi-agent systems. In *Cooperative information agents* (p. 62-77).

Doval, D., & O'Mahony, D. (2003). Overlay networks: A scalable alternative for p2p. *IEEE Internet Computing*, *7*(4), 79–82.

Dunne, C. R. (2001). Using mobile agents for network resource discovery in peer-to-peer networks. *ACM SIGecom Exchanges*, *2*, 1–9.

Forum, G. D. (2002). *Gnutella protocol specification v0.4.* Available from `http://rfc-gnutella.sourceforge.net/developer/stable/index.html`

Gao, J., & Zhang, J. (2005). Clustered svd strategies in latent semantic indexing. *Information Processing and Management*, *41*(5), 1051–1063.

Golub, G. H., & Loan, C. F. V. (1996). *Matrix computations.* The John Hopkins University Press.

Guvnec, I., & Urdaneta, J. J. (2010). *Peer-to-peer file sharing: A survey.* Available from `http://www.cs.ucr.edu/ michalis/COURSES/179-03/p2psurvey.ppt`

Hasan, M., & Matsumoto, Y. (1999). *Document clustering: before and after the singular value decomposition* (Tech. Rep.). Nara Instritute of Science and Tecjnology. Technical Report TR-99. Processing Society of Japan.

Kambayashi, Y., & Harada, Y. (2007). A resource discovery method based on multi-agents in p2p systems. In *Kes-amsta* (p. 364-374).

Kambayashi, Y., & Harada, Y. (2009). A resource discovery method based on multiple mobile agents in p2p systems. In *Intelligent agents in the evolution of web and applications* (p. 113-135).

Kang, S., Lee, Y., Lee, D., & Youn, H. Y. (2007). A landmark-based scalable semantic resource discovery scheme. *IEICE - Trans. Inf. Syst.*, *E90-D*(6), 986–989.

Karnstedt, M., Hose, K., & Sattler, K. uwe. (2004). Query routing and processing in schema-based p2p systems. In *In proceedings of dexa workshops* (pp. 544–548). IEEE Computer Society.

Killmeyer, J. (2006). *Information security architecture : an integrated approach to security in the organization* (2nd ed. ed.). Auerbach. Boca Raton, Fla. u.a.

Konchady, M. (2006). *Text mining application programming.* Charles River Media.

Li, J., Loo, B. T., Hellerstein, J. M., Kaashoek, M. F., Karger, D. R., & Morris, R. (2003). On the feasibility of peer-to-peer web indexing and search. In *Iptps* (p. 207-215).

Lingnau, A., & Drobnik, O. (1999). Simulating mobile agent systems with swarm. *Agent Systems and Applications, International Symposium on / International Symposium on Mobile Agents*, *0*, 272.

Liu, X., Chen, M., & Yang, G. (2004). Latent semantic indexing in peer-to-peer networks. In *Arcs* (p. 63-77).

Lopes, A. L., & Botelho, L. M. (2008). Improving multi-agent based resource coordination in peer-to-peer networks. *Journal of Networks*, *3*(2), 38-47.

Lv, Q., Cao, P., Cohen, E., Li, K., & Shenker, S. (2002). Search and replication in unstructured peer-to-peer networks. In *Proceedings of the 16th annual acm international conference on supercomputing (ics) 2002.*

Mastroianni, C., Talia, D., & Verta, O. (2005). A super-peer model for building resource discovery services in grids: Design and simulation analysis. In *Egc* (p. 132-143).

McCrary, K., & Waters, B. (2000, October). *Jtella v0.7.* Available from `http://jtella.sourceforge.net/`

Milojicic, D. S., Breugst, M., Busse, I., Campbell, J., Covaci, S., Friedman, B., et al. (1998). Masif: The omg mobile agent system interoperability facility. *Personal and Ubiquitous Computing*, *2*(2).

Napster. (2003). *Napster.* Available from `http://www.napster.com`

North, M. J., Collier, N. T., & Vos, J. R. (2006). Experiences creating three implementations of the repast agent modeling toolkit. *ACM Trans. Model. Comput. Simul.*, *16*(1), 1–25.

Prakash, A. (2006, May). *A survey of advanced search in p2p networks.* Department of Computer Science. Available from `http://www.medianet.kent.edu/surveys/IAD06S-p2psearch-alok/index.html`

Project, T. C. (2010). *Chord faq.* Available from `http://pdos.csail.mit.edu/`

Ratnasamy, S., Francis, P., Shenker, S., Karp, R., & Handley, M. (2001). A scalable content-addressable network. In *In proceedings of acm sigcomm* (pp. 161–172).

Reynolds, P., & Vahdat, A. (2003). Efficient peer-to-peer keyword searching. In *Middleware '03: Proceedings of the acm/ifip/usenix 2003 international conference on middleware* (pp. 21–40). New York, NY, USA: Springer-Verlag New York, Inc.

Rowstron, A., & Druschel, P. (2001). *Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems.*

Schoeman, M., & Cloete, E. (2003). Architectural components for the efficient design of mobile agent systems. In *Saicsit '03: Proceedings of the 2003 annual research conference of the south african institute of computer scientists and information technologists on enablement through technology* (pp. 48–58). , Republic of South Africa: South African Institute for Computer Scientists and Information Technologists.

Singh, M., Cheng, X., & He, X. (2009). *Multimedia resource discovery using mobile agent.* New York,: IGI Global.

Stoica, I., Morris, R., Liben-Nowell, D., Karger, D. R., Kaashoek, M. F., Dabek, F., et al. (2001). Chord: A scalable peer-to-peer lookup protocol for internet applications. In *Acm sigcomm* (pp. 149–160).

Sun, Y., Sun, L., Huang, X., & Lin, Y. (2006). Resource discovery in locality-aware group-based semantic overlay of peer-to-peer networks. In *Infoscale '06: Proceedings of the 1st international conference on scalable information systems* (p. 40). New York, NY, USA: ACM.

Talai, P. T. D., Fragopoulou, P., Mordacchini, M., Pennanen, M., Popov, K., & Haridi, V. V. S. (2006). *Peer-to-peer models for resource discovery on grids* (Tech. Rep.). Institution of System Architecture.

Tan, Y., & Zheng, Z. (2009). A multi-agent based resource discovery scheme for p2p systems. In *International workshop on intelligent systems and applications - isa 2009* (p. 1-4).

Tang, C., & Dwarkadas, S. (2004). Hybrid global-local indexing for efficient peer-to-peer information retrieval. In *Nsdi* (p. 211-224).

Tang, C., Xu, Z., & Dwarkadas, S. (2003). Peer-to-peer information retrieval using self-organizing semantic overlay networks. In *Proceeding of acm sig-comm* (p. 178-186).

Tran, H. M., & Schonwalden, J. (2008, March). *Distributed case-based reasoning for fault management.* 1st EMANICS Workshop on Peer-to-Peer Management.

Trillo, R., Ilarri, S., & Mena, E. (2007). Comparison and performance evaluation of mobile agent platforms. In *Icas '07: Proceedings of the third international conference on autonomic and autonomous systems* (p. 41). Washington, DC, USA: IEEE Computer Society.

University, C. (1999, August). *Cornell university: Medline text collection.* Smart System. USA. Available from `ftp://ftp.cs.cornell.edu/pub/smart/med/`

Vieira, R. (2001). *Foundation of intelligent agents - agent communication language* (Tech. Rep.). FIPA.

Wireshark. (2010). *Wireshark go deep.* Online. Available from `http://www.wireshark.org`

Wong, J., Helmer, G., Naganathan, V., Polavarapu, S., Honavar, V. G., & Miller, L. (2001). Smart mobile agent facility. *Journal of Systems and Software*, *56*(1), 9–22.

Yang, K.-H., Wu, C.-J., & Ho, J.-M. (2007). Antsearch: An ant search algorithm in unstructured peer-to-peer networks. *IEICE Transactions*, *89-B*(9), 2300-2308.

Yingwu Zhu, Y. H. (2005). *Handbook on theoretical and algorithmic aspects of sensor, ad hoc wireless, and peer-to-peer networks - semantic search in peer-to-peer systems* (J. Wu, Ed.). CRC Press.

Zaharia, M., & Keshav, S. (2008). Gossip-based search selection in hybrid peer-to-peer networks: Research articles. *Concurr. Comput. : Pract. Exper.*, *20*(2), 139–153.

Zhao, B. Y., Huang, L., Stribling, J., Rhea, S. C., Joseph, A. D., & Kubiatowicz, J. D. (2004). Tapestry: A resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications*, *22*, 41–53.

Zhao, B. Y., Kubiatowicz, J., Joseph, A. D., Zhao, B. Y., Kubiatowicz, J., & Joseph, A. D. (2001). *Tapestry: An infrastructure for fault-tolerant wide-area location and routing* (Tech. Rep.).

Zhong, Y., & Liu, J. (2003). *The mobile agent technology.* ISBN 7-89494-143-3.

Zhu, Y., & Hu, Y. (2004). *Ess: Efficient semantic search on gnutella-like p2p systems* (Tech. Rep.). Department of ECECS, University of Cincinnati.

Zhu, Y., & Hu, Y. (2005). Efficient, proximity-aware load balancing for dht-based p2p systems. *IEEE Trans. Parallel Distrib. Syst.*, *16*(4), 349–361.

Zhu, Y., & Hu, Y. (2006). Handbook of theoretical and algorithimic aspects of ad hoc, sensor, and peer-to-peer networks. In J. Wu (Ed.), (p. 634-664). Auerbach Publications.

Zhu, Y., & Hu, Y. (2007). Efficient semantic search on dht overlays. *Journal of Parallel and Distributed Computing*, *67*(5), 604 - 616.

Zhu, Y., Wang, H., & Hu, Y. (2003). Intergrating semantics-based access mechanisms with p2p file systems. In *Proceedings of third international conference on peer-to-peer computing.*

Appendix A

Similarity Measures and Weighting Functions

Assuming two n-dimensional vectors $X = (x_1, x_2, x_3, \ldots, x_n)$ and $Y = (y_1, y_2, y_3, \ldots, y_n)$.

Name of Measure	Formula
Euclidean Distance	$\sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$
Dot Product	$\sum_{i=1}^{n} x_i y_i$
Jaccard Similarity	$\frac{\lvert X \cap Y \rvert}{\lvert X \cup Y \rvert}$
Cosine Similarity	$\frac{\sum_{i=1}^{n} x_i y_i}{\sqrt{\sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i}}$

Table A.1: Similarity measures

Local Weighting Functions $L(m, n)$ and Global Weighing Functions $G(m)$:

Type	$L(m, n)$	$G(m)$
Binary	$\begin{cases} 0 & tf_{mn} = 0 \\ 1 & tf_{mn} > 0 \end{cases}$	$\sqrt{\frac{1}{\sum_n (tf_{mn})^2}}$
Term-Frequency	tf_{mn}	$\frac{GlobalFreqencyOfTerm"m"}{FrequencyOfNodesInWhichTerm"m"Appears}$
log	$\ln(tf_{mn} + 1)$	$ln(\frac{NumberOfDocuments}{FrequencyOfNodesInWhichTerm"m"Appears}) + 1$

Table A.2: Local and global weighting functions

Where

tf_{mn} =Frequency of term m in node n

Appendix B

Classes Realised - Affinity

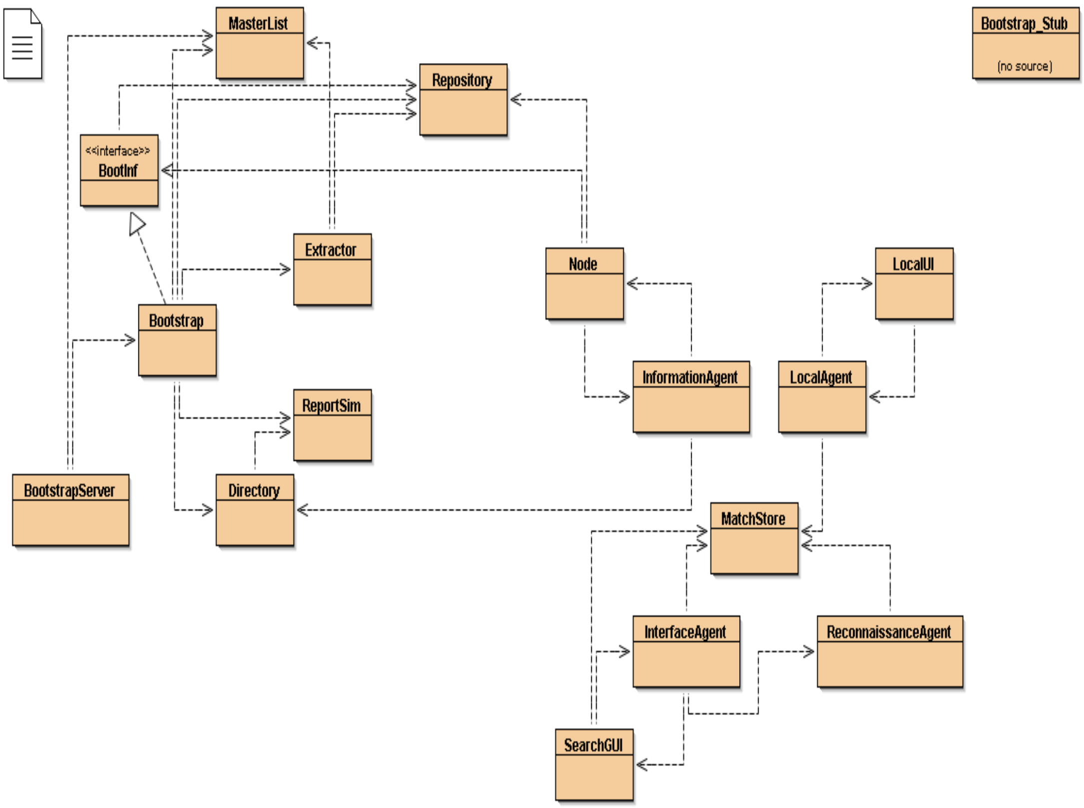

Figure B.1: Classes for resource discovery system - Affinity

Appendix C

Program Listings - Affinity

C.1 Interface BootInf.java

```
import java.rmi.*;
import java.util.*;
/**
 * Remote RMI Interface for Bootstrap Server
 *
 * @author M. Singh
 * @version 1.0
 */

public interface BootInf extends Remote
{
    public HashMap<String,Directory> register(Repository message, double
        min_Sup) throws RemoteException;
    public void disconnect(String registrationIP) throws RemoteException;
}
```

C.2 Class Bootstrap.java

```
import java.rmi.*;
import java.rmi.server.*;
import java.util.*;
import Jama.*;
/**
 * Implementation of BootInf Remote Methods. These methods available to
 * Information Agent for registering the Node
 *
 * @author M. Singh
 * @version 2.0
 */
public class Bootstrap extends UnicastRemoteObject implements BootInf
{
    private MasterList database;
    private Extractor extractor;
    private HashMap<String,Directory> map;

    public Bootstrap(MasterList database) throws RemoteException
    {
        this.database=database;
        /**
         * Dummy Repository added to compensate for null pointer
         */
        String dummyIP="bootstrap";
        ArrayList dummyKeyword = new ArrayList();
        dummyKeyword.add("ytiniffa");
        dummyKeyword.add("metsys");
        //dummyKeyword.add("shipment");
        //dummyKeyword.add("of");
        //dummyKeyword.add("gold");
        //dummyKeyword.add("damaged");
        //dummyKeyword.add("in");
        //dummyKeyword.add("a");
        //dummyKeyword.add("fire");
        Repository dummyMessage = new Repository(dummyIP,dummyKeyword);
        database.add(dummyIP);
        extractor = new Extractor(database, dummyMessage);
    }
```

```java

    public  HashMap<String,Directory> register(Repository message, double
        min_Sup) throws RemoteException
    {
        System.out.println("\nSubmitted keywords by client "+message.getIP()+"
            \n"+message.getKeywords());
        //Add ip to MasterList
        database.add(message.getIP());
        extractor = new Extractor(database, message);
        //Prepare Reply for the Client based on its preferences
        HashMap<String,Directory> hashmap = prepareSimReply(min_Sup);
        return hashmap;
    }

    public HashMap<String,Directory> prepareSimReply(double min_Sup)
    {
        map=new HashMap<String,Directory>();
        //ReportSim report = new ReportSim();
        Matrix sim = database.getSimMatrix();
        ArrayList clients = database.getClientList();
        Matrix master = database.getMasterKeywordList();

        ArrayList indexHolder = new ArrayList();

        //find index in sim report that is higher than user minimum support
            preference
        double[][] simArray = sim.getArray();
        for(int a=0;a<sim.getRowDimension();a++)
        {
            if(simArray[a][0]>min_Sup)
            {
                indexHolder.add(a);
            }
        }
        ArrayList<Directory> directoryHolder=new ArrayList<Directory>();
        for(int i=0;i<indexHolder.size();i++)
        {
            Directory directory = new Directory();
            directoryHolder.add(directory);
        }
```

```
        //set similarity sub matrix based on index holder
        Object[] rowI = indexHolder.toArray();
        int[] rows = new int[rowI.length];
        int[] cols = {0};
        for(int i=0;i<rows.length;i++)
        {
            Integer r = (Integer)rowI[i];
            int rs = r.intValue();
            rows[i]=rs;
        }
        Matrix tempSim = sim.getMatrix(rows,cols);
        System.out.println("SIMILARITY MATRIX FOR LATEST CLIENT");
        tempSim.print(tempSim.getColumnDimension(),3);
        //report.setSimilarity(tempSim);
        double[][] arraySim = tempSim.getArray();
        for(int i=0;i<directoryHolder.size();i++)
        {
            Directory d = directoryHolder.get(i);
            d.similarityValue=arraySim[i][0];
            directoryHolder.set(i,d);
        }

        //set clients based on index holder
        ArrayList tempClients = new ArrayList();
        for(int i=0;i<indexHolder.size();i++)
        {
            tempClients.add((String)clients.get(((Integer)indexHolder.get(i)).
                intValue()));
        }
        System.out.println("CLIENTS WITH BEST SIMILARITY - IN CLUSTER");
        System.out.println(tempClients);
        //report.setClients(tempClients);

        //set temp master sub matrix based on index holder
        Object[] colI = indexHolder.toArray();
        int[] colm = new int[colI.length];
        for(int i=0;i<colm.length;i++)
        {
            Integer c = (Integer)colI[i];
            int cs= c.intValue();
```

```java
            colm[i]=cs;
        }
        int[] rowm = new int[master.getRowDimension()];
        for(int i=0;i<rowm.length;i++)
        {
            rowm[i]=i;
        }
        Matrix tempWeights = master.getMatrix(rowm,colm);
        System.out.println("WEIGHTS FOR CLIENTS KEYWORDS - IN CLUSTER");
        tempWeights.print(tempWeights.getColumnDimension(),1);
        //report.setKeywordsWeights(tempWeights);
        double[][] tempWeightsArray = tempWeights.getArray();
        for(int m=0;m<tempWeightsArray[0].length;m++)
        {
            Directory d = directoryHolder.get(m);
            double[][] keywordWeight = new double[tempWeightsArray.length][1];
            for(int n=0;n<tempWeightsArray.length;n++)
            {
                keywordWeight[n][0]=tempWeightsArray[n][m];
            }
            Matrix keyWeight = new Matrix(keywordWeight);
            d.keyWeights= keyWeight;
            directoryHolder.set(m,d);
        }

        //set keyword List
        //report.setKeywordList(database.getKeywordList());
        for(int i=0;i<directoryHolder.size();i++)
        {
            Directory d = directoryHolder.get(i);
            d.keywords=database.getKeywordList();
            directoryHolder.set(i,d);
        }

        /**
         * Test Purpose Only
         */
        /*
        for(int h=0;h<directoryHolder.size();h++)
        {
```

```
            Directory b = directoryHolder.get(h);
            System.out.println("SEEMS UPDATED");
            System.out.println(b.similarityValue);
            Matrix a = b.keyWeights;
            a.print(a.getColumnDimension(),1);
            System.out.println(b.keywords);
        }
        */

        for(int i=0;i<directoryHolder.size();i++)
        {
            Directory d = directoryHolder.get(i);
            //d.report = report;
            directoryHolder.set(i,d);
        }

        for(int i=0;i<tempClients.size();i++)
        {
            map.put((String)tempClients.get(i),directoryHolder.get(i));
        }

        return map;
    }

    public void disconnect(String registrationIP) throws RemoteException
    {
        System.out.println("Removed IP "+registrationIP);
        map.remove(registrationIP);
        database.remove(registrationIP);
    }

}//end class
```

C.3 Class BootstrapServer.java

```java
import java.rmi.*;
import java.util.*;
import java.net.*;
/**
 * RMI Bootstrap Server
 *
 * @author M. Singh
 * @version 1.1
 */
public class BootstrapServer
{
    public static void main(String argv[])
    {
        String localIP="";
        String reference="";
        try
        {
            InetAddress local_Address = InetAddress.getLocalHost();
            localIP = local_Address.getHostAddress();
        }catch(java.net.UnknownHostException e)
        {
            System.out.println("Error getting IP Address "+e);
        }

        try
        {
            //ArrayList<Repository> database = new ArrayList<Repository>();
            MasterList database = new MasterList();
            Bootstrap bootstrap = new Bootstrap(database);
            reference = "rmi://"+localIP+"/Server_1";
            Naming.rebind(reference,bootstrap);
            System.out.println("Bootstrap server instance "+reference+" Running \nWaiting for Nodes to regsiter");
        }catch(Exception e)
        {
            System.out.println("Error Starting Bootstrap Server "+e);
        }
    }
```

```
}//end class
```

C.4 Class Extractor.java

```
import java.util.*;
/**
 * Extractor is helper class for MasterList used for extracting keywords from
     Repository
 *
 * @author M. Singh
 * @version 1.5
 */
public class Extractor
{
    private MasterList database;
    private Repository message;
    private String tk[];
    private int size;
    private SortedMap map;

    public Extractor(MasterList database, Repository message)
    {
        this.database=database;
        this.message=message;
        map=new TreeMap();
        getKeywords();
    }

    public void getKeywords()
    {
        int index=0;
        String keywordSet="";
        ArrayList keywords = message.getKeywords();
        for(int i=0;i<keywords.size();i++)
        {
            keywordSet+=(String)keywords.get(i)+" ";
        }
        //Tokenize
        StringTokenizer token = new StringTokenizer(keywordSet);
        size=token.countTokens();
        makeTKArray(size);
        while(token.hasMoreTokens())
```

```
            {
                tk[index]=token.nextToken();
                findTokenFrequency(tk[index]);
            }
            database.addKeywords(map);
        }

        public void makeTKArray(int size)
        {
            tk = new String[size];
        }

        public void findTokenFrequency(String token)
        {
            if(!map.containsKey(token))
            {
                map.put(token.toLowerCase(),1);
            }else
            {
                Integer frequency = (Integer)map.get(token);
                int freqVal = frequency.intValue();
                freqVal+=1;
                map.remove(token);
                map.put(token,freqVal);
            }
        }

}
```

C.5 Class ReportSim.java

```java
import Jama.*;
import java.util.*;
import java.io.*;
/**
 * Serialized class Similarity Report sent between Bootstrap Server and Node
 *
 * @author M. Singh
 * @version 1.1
 */
public class ReportSim implements Serializable
{
    private ArrayList clients;
    private Matrix similarity;
    private Matrix weights;
    private ArrayList keywordList;

    public ReportSim(ArrayList clients, Matrix similarity, Matrix weights,
        ArrayList keywordList)
    {
        this.clients=clients;
        this.similarity=similarity;
        this.weights=weights;
        this.keywordList=keywordList;
    }

    public ReportSim()
    {
    }

    public void setClients(ArrayList clients)
    {
        this.clients=clients;
    }

    public ArrayList getClients()
    {
        return clients;
    }
```

```

    public void setSimilarity(Matrix similarity)
    {
        this.similarity=similarity;
    }

    public Matrix getSimilarity()
    {
        return similarity;
    }

    public void setKeywordsWeights(Matrix weights)
    {
        this.weights=weights;
    }

    public Matrix getKeywordsWeights()
    {
        return weights;
    }

    public void setKeywordList(ArrayList keywordList)
    {
        this.keywordList=keywordList;
    }

    public ArrayList getKeywordList()
    {
        return keywordList;
    }
}
```

C.6 Class Directory.java

```
import java.io.*;
import Jama.*;
import java.util.*;
/**
 * Data Structire for holding the directory peer - keyword matrix used by
 * Information Agent and Bootstrap
 *
 * @author M. Singh
 * @version 1.1
 */
public class Directory implements Serializable
{
    double similarityValue;
    Matrix keyWeights;
    ArrayList keywords;
    //ReportSim report;
}
```

C.7 Class MasterList.java

```
import java.util.*;
import Jama.*;
/**
 * Master List class holds the global peer - keyword matrix
 *
 * @author M. Singh
 * @version 2.1
 */
public class MasterList
{
    private ArrayList ipCols;
    private ArrayList<SortedMap> tempMaster;
    private double[][] masterKeywordMatrix;
    private double[][] joiningNodeKeywords;
    private ArrayList<String> listOfKeywords;
    private Matrix U;
    private Matrix S;
    private Matrix V;
    private Matrix S_inverse;
    private Matrix V_transpose;
    private Matrix Q_transpose;
    private Matrix q;
    private Matrix simM;
    private String forIP="";

    public MasterList()
    {
        ipCols = new ArrayList();
        tempMaster = new ArrayList<SortedMap>();
    }

    public void add(String ip)
    {
        //ipCols.add(ip);
        if(!ipCols.contains(ip))
        {
            ipCols.add(ip);
            forIP=ip;
```

```
        }else
        {
            int index=ipCols.indexOf(ip);
            ipCols.remove(index);
            tempMaster.remove(index);
            ipCols.add(ip);
            forIP=ip;
        }
    }

    public void remove(String ip)
    {
        if(ipCols.contains(ip))
        {
            int index=ipCols.indexOf(ip);
            ipCols.remove(index);
            tempMaster.remove(index);
        }
    }

    public void addKeywords(SortedMap map)
    {
        tempMaster.add(map);
        prepareMatrix();
    }

    public void prepareMatrix()
    {
        SortedMap completeList = new TreeMap();
        SortedMap temp = new TreeMap();
        for(int i=0;i<tempMaster.size();i++)
        {
            temp=tempMaster.get(i);
            Set keywords = temp.keySet();
            Iterator itKeys = keywords.iterator();
            while(itKeys.hasNext())
            {
                String key = (String)itKeys.next();
                if(!completeList.containsKey(key))
                {
```

```
                    completeList.put(key,0);
                }
            }//end while
        }//end loop

        setCompleteList(completeList);

        masterKeywordMatrix = new double[completeList.size()][ipCols.size()];
        int rows = masterKeywordMatrix.length;
        int cols = masterKeywordMatrix[0].length;
        for(int n=0;n<cols;n++)
        {
            temp= new TreeMap();
            Set completeKeys = completeList.keySet();
            Iterator it = completeKeys.iterator();
            temp=tempMaster.get(n);
            for(int m=0;m<rows;m++)
            {
                while(it.hasNext())
                {
                    String key = (String)it.next();
                    if(temp.containsKey(key))
                    {
                        int val = ((Integer)temp.get(key)).intValue();
                        masterKeywordMatrix[m][n]=val;
                    }else
                    {
                        masterKeywordMatrix[m][n]=0.0;
                    }
                    m++;
                }
            }
        }

        if(cols==1)
        {
            //do nothing
            //to protect system from issuing dummy
        }else
        {
```

```
            //display
            System.out.println("Master␣Matrix");
            displayMatrix(masterKeywordMatrix,cols);
            //calculate SVD considering the joining column ip address and its
                keywords is the new query(joining node keywords).
            double[][] tempKeywordMatrix = new double[rows][cols-1];
            joiningNodeKeywords=new double[rows][1];
            int runner=0;
            for(int b=0;b<cols;b++)
            {
                for(int a=0;a<rows;a++)
                {
                    if(b==ipCols.indexOf(forIP))
                    {
                        //do not get that column
                        //make it joining node
                        //joined keywords
                        for(int z=0;z<rows;z++)
                        {
                            joiningNodeKeywords[z][0]=masterKeywordMatrix[z][b
                                ];
                        }
                    }else
                    {
                        tempKeywordMatrix[a][runner]=masterKeywordMatrix[a][b
                            ];
                    }
                }
                runner++;
            }

            /*
            for(int b=0;b<cols-1;b++)
            {
                for(int a=0;a<rows;a++)
                {
                    tempKeywordMatrix[a][b]=masterKeywordMatrix[a][b];
                }
            }

```

```
            //joined keywords
            for(int a=0;a<rows;a++)
            {
                joiningNodeKeywords[a][0]=masterKeywordMatrix[a][ipCols.size()
                    -1];
            }
             */

            System.out.println("Compared␣Against␣Matrix");
            displayMatrix(tempKeywordMatrix,tempKeywordMatrix[0].length);
            System.out.println("Joining␣Matrix");
            displayMatrix(joiningNodeKeywords,joiningNodeKeywords[0].length);
            Matrix Q = new Matrix(joiningNodeKeywords);
            Q_transpose=Q.transpose();
            calculateSVD(tempKeywordMatrix);
            calculateq();
            Matrix sim = calculateSim();
            System.out.println("Similarity␣Report");
            sim.print(sim.getColumnDimension(),3);
        }
    }

    /**
     * Displays matrix
     */
    public void displayMatrix(double[][] matrix,int cols)
    {
        Matrix mat = new Matrix(matrix);
        mat.print(cols,1);
    }

    /**
     * SVD calculation
     */
    public void calculateSVD(double[][] matrix)
    {
        Matrix mat = new Matrix(matrix);
        SingularValueDecomposition svd = mat.svd();
        U = svd.getU();
        //U.print(ipCols.size(),3);
```

```
            S = svd.getS();
            //S.print(ipCols.size(),3);
            S_inverse = S.inverse();
            V = svd.getV();
            //V.print(ipCols.size(),3);
            V_transpose= V.transpose();
        }

        /**
         * Computing query vector
         */
        public void calculateq()
        {
            q = (Q_transpose.times(U)).times(S_inverse);
        }

        public Matrix calculateSim()
        {
            double[][] vofQuery = q.getArray();
            double[][] vofTerm = V_transpose.getArray();
            double[][] sim=new double[vofQuery[0].length][1];
            double[] num=new double[vofTerm.length];
            double den1=0;
            double[] den2=new double[vofTerm.length];

            for(int x =0;x<vofTerm.length;x++)
            {
                for(int i =0;i<vofQuery[0].length;i++)
                {
                    num[x] +=vofQuery[0][i]*vofTerm[i][x];
                }
            }

            for(int x=0;x<vofQuery[0].length;x++)
            {
                den1+=vofQuery[0][x]*vofQuery[0][x];
            }

            den1 = Math.sqrt(den1);

```

```
        for(int i=0;i<den2.length;i++)
        {
            for(int x=0;x<vofTerm.length;x++)
            {
                den2[i]+=vofTerm[x][i]*vofTerm[x][i];
            }
        }

        for(int x=0;x<den2.length;x++)
        {
            den2[x] = Math.sqrt(den2[x]);
        }

        for(int i=0;i<sim.length;i++)
        {
            sim[i][0]=num[i]/(den1*den2[i]);
        }

        simM = new Matrix(sim);
        return simM;
    }

    public Matrix getSimMatrix()
    {
        return simM;
    }

    public ArrayList getClientList()
    {
        return ipCols;
    }

    public Matrix getMasterKeywordList()
    {
        Matrix master = new Matrix(masterKeywordMatrix);
        return master;
    }

    public void setCompleteList(SortedMap completeList)
    {
```

```
            Set keys = completeList.keySet();
            listOfKeywords = new ArrayList<String>();
            Iterator it = keys.iterator();
            while(it.hasNext())
            {
                listOfKeywords.add((String)it.next());
            }
        }

        public ArrayList getKeywordList()
        {
            return listOfKeywords;
        }
}//end class
```

C.8 Class Repository.java

```
import java.io.*;
import java.util.*;
/**
 * Serialised Repository Data Structure
 *
 * @author M. Singh
 * @version 1.0
 */
public class Repository implements Serializable
{
    private String ip_Address;
    private ArrayList keywords;

    public Repository(String ip_Address, ArrayList keywords)
    {
        this.ip_Address=ip_Address;;
        this.keywords=keywords;
    }

    public void setIP(String ip_Address)
    {
        this.ip_Address=ip_Address;
    }

    public String getIP()
    {
        return ip_Address;
    }

    public void setKeywords(ArrayList keywords)
    {
        this.keywords=keywords;
    }

    public ArrayList getKeywords()
    {
        return keywords;
    }
```

```
}//end class
```

C.9 Class Node.java

```
import java.rmi.*;
import java.net.*;
import java.util.*;
import jade.core.*;
/**
 * Node class presents the peer and is responsible for communication with RMI Bootstrap server
 * and register the peer and keyword matrix
 *
 * @author M. Singh
 * @version 1.0
 */
public class Node
{
    String name="";
    String reference="";
    BootInf boot=null;
    InformationAgent ia;
    HashMap<String,Directory> clusterNeighbours;

    public Node()
    {
        try
        {
            reference = "rmi://192.168.1.144/Server_1";
            boot = (BootInf)Naming.lookup(reference);
            System.out.println("Connected to "+reference+" successfully.");
        }catch(Exception e)
        {
            System.out.println("Error on Node "+e);
            try
            {
                //boot.disconnect(name);
            }catch(Exception ee)
            {
                System.out.println(e);
            }
        }
```

```
    }

    public void setAgent(InformationAgent agent)
    {
        this.ia=ia;
    }

    public void connectToBootStrap(String nameLocalAgent, ArrayList<String>
        keywords)
    {
        name=nameLocalAgent;
        try
        {
            Repository message = new Repository(nameLocalAgent,keywords);
            clusterNeighbours = boot.register(message,0.0);
        }catch(Exception e)
        {
            try
            {
                //boot.disconnect(nameLocalAgent);
            }catch(Exception ee)
            {
                System.out.println("Error Disconnecting "+ee);
            }
        }
    }

    public void remove()
    {
        try
        {
            boot.disconnect(name);
            System.exit(0);
        }catch(Exception e)
        {
            System.out.println(e);
        }
    }

    public HashMap<String,Directory> getNeighbours()
```

```
        {
            return clusterNeighbours;
        }
}//end class
```

C.10 Class InformationAgent.java

```java
import jade.core.*;
import jade.lang.acl.*;
import jade.core.behaviours.*;
import java.util.*;
import javax.swing.*;
import Jama.*;

/**
 * Information Agent holds information about peers that are semantically similar to this peer.
 *
 * @author M. Singh
 * @version 1.5
 */
public class InformationAgent extends Agent
{
    private String nameLA="";
    private ArrayList<String> keywords;
    private Node node=new Node();
    private HashMap<String,Directory> clusterNeighbours;
    private String cont;
    private String query;
    private String minSup;
    private Database database = new Database();
    private Finder finder;

    protected void setup()
    {
        //Display the GUID name of agent
        String name = getAID().getName();
        System.out.println("Information-agent GUID "+name+" started.");
        //there for the IA name must be
        if(name.startsWith("I"))
        {
            nameLA=name;
            nameLA=nameLA.replace("I","L");
        }
        node.setAgent(this);
```

```java

        addBehaviour(new KeywordRequestor(this));
        addBehaviour(new NodeRequestor());
    }

    protected void takeDown()
    {
        doDelete();
    }

    public void callNodeRegistry()
    {
        if(keywords!=null)
        {
            node.connectToBootStrap(nameLA,keywords);
        }
    }

    //inner class Keyword Requestor
    private class KeywordRequestor extends TickerBehaviour
    {
        private KeywordRequestor(Agent a)
        {
            super(a,20000);
        }

        public void onStart()
        {
            // some thing for start
        }

        public void onTick()
        {
            //Send Message
            ACLMessage request=new ACLMessage(ACLMessage.REQUEST);
            request.addReceiver(new AID(nameLA,AID.ISGUID));
            request.setConversationId("keywords-request");
            request.setReplyWith("request"+System.currentTimeMillis());
            myAgent.send(request);

```

```
            //Prepare message receiving template
            MessageTemplate mt = MessageTemplate.and(MessageTemplate.MatchConversationId("keywords-request"),MessageTemplate.MatchInReplyTo(request.getReplyWith()));
            ACLMessage reply = myAgent.receive();

            if(reply!=null)
            {
                if(reply.getPerformative()==ACLMessage.INFORM)
                {
                    try
                    {
                        keywords=(ArrayList<String>)reply.getContentObject();
                    }catch(Exception e)
                    {
                        e.printStackTrace();
                    }
                }
                callNodeRegistry();
            }
            else
            {
                //System.out.println("This will take 60000 msecs - Current State Block");
                //block();
            }
        }
    }//end inner class

    //inner class Node Requestor
    private class NodeRequestor extends CyclicBehaviour
    {
        private MessageTemplate mt = MessageTemplate.MatchPerformative(ACLMessage.REQUEST);
        public void action()
        {
            clusterNeighbours=node.getNeighbours();
            try
            {
                ACLMessage messagerec=myAgent.receive(mt);
```

```
        if(messagerec!=null)
        {
            System.out.println("Request from "+messagerec.getSender().getLocalName()+"\n"+"reconnaissance agent to issue the node");
            String cont = messagerec.getContent();
            String[] myCont = cont.split(":");
            query=myCont[0];
            minSup=myCont[1];
            System.out.println("QUERY ------------------------------------ "+query);
            //Finding Node with best match using the directory received from the Boot strap server
            //update arraylist to inclue query words
            if(clusterNeighbours!=null)
            {
                Set keysIPS = clusterNeighbours.keySet();
                Object[] keysArray = (Object[])keysIPS.toArray();
                Directory dir=null;
                for(int j=0;j<keysArray.length;j++)
                {
                    ArrayList updated=new ArrayList();
                    dir = clusterNeighbours.get((String)keysArray[j]);
                    ArrayList a = dir.keywords;
                    System.out.println(a);
                    Matrix weights = dir.keyWeights;
                    double[][] weightsMatrix = weights.getArray();

                    for(int k=0;k<weights.getRowDimension();k++)
                    {
                        if(weightsMatrix[k][0]==0)
                        {
                        }else if(weightsMatrix[k][0]>=1)
                        {
                            for(int u=0;u<weightsMatrix[k][0];u++)
                            {
                                updated.add(a.get(k));
                            }
                        }
                    }
```

```

                Message message = new Message((String)keysArray[j],updated);
                database.add(message.getIP());
                finder = new Finder(message);
            }
 //query
 StringTokenizer st = new StringTokenizer(query);
 ArrayList queryList = new ArrayList();
 while(st.hasMoreTokens())
 {
     queryList.add((String)st.nextToken());
 }
Message mQ = new Message("QUERY",queryList);
database.add(mQ.getIP());
finder = new Finder(mQ);
database.compute();
cont="";
query="";
}else
{
    System.out.println("Cluster Neighbours got issue or the Information Agent is not online yet.");
}

Matrix simR=database.getSimMatrix();
double[][] simRArray = simR.getArray();
double mins = Double.parseDouble(minSup);
int indexer=0;
double simVal=0.0;
double temp=0.0;

for(int q=0;q<simRArray.length;q++)
{
    simVal=simRArray[q][0];
    if(simVal>mins && simVal>temp)
    {
        indexer=q;
        temp=simVal;
    }
```

```
                        }

                        ArrayList clientAgents=database.getClientList();
                        String chosen = (String)clientAgents.get(indexer);
                        System.out.println("THE CHOSEN ONE IS "+chosen);

                        ACLMessage reply = messagerec.createReply();
                        if(keywords!=null)
                        {
                            reply.setPerformative(ACLMessage.INFORM);
                            reply.setContent(chosen);
                        }
                        myAgent.send(reply);
                        System.out.println(keywords!=null ? "Informed "+messagerec
                            .getSender().getLocalName()+" about chosen node - "+
                            reply.getContent() : "Node did not exist");
                    }else
                    {
                        block();
                    }
                }catch(Exception e)
                {
                    e.printStackTrace();
                }
            }
        }//end inner class Node Requestor

        private class Message
        {
            private String ip_Address;
            private ArrayList keywords;

            public Message(String ip_Address, ArrayList keywords)
            {
                this.ip_Address=ip_Address;;
                this.keywords=keywords;
            }

            public void setIP(String ip_Address)
            {
```

```
            this.ip_Address=ip_Address;
        }

        public String getIP()
        {
            return ip_Address;
        }

        public void setKeywords(ArrayList keywords)
        {
            this.keywords=keywords;
        }

        public ArrayList getKeywords()
        {
            return keywords;
        }

    }//end inner class

    //inner class to find node
    private class Finder
    {
        private Message message;
        private String tk[];
        private int size;
        private SortedMap map;

        public Finder(Message message)
        {
            this.message=message;
            map=new TreeMap();
            getKeywords();
        }

        public void getKeywords()
        {
            int index=0;
            String keywordSet="";
            ArrayList keywords1 = message.getKeywords();
```

```
                for(int i=0;i<keywords1.size();i++)
                {
                    keywordSet+=(String)keywords1.get(i)+" ";
                }
                //Tokenize
                StringTokenizer token = new StringTokenizer(keywordSet);
                size=token.countTokens();
                makeTKArray(size);
                while(token.hasMoreTokens())
                {
                    tk[index]=token.nextToken();
                    findTokenFrequency(tk[index]);
                }
                database.addKeywords(map);
            }

            public void makeTKArray(int size)
            {
                tk = new String[size];
            }

            public void findTokenFrequency(String token)
            {
                if(!map.containsKey(token))
                {
                    map.put(token.toLowerCase(),1);
                }else
                {
                    Integer frequency = (Integer)map.get(token);
                    int freqVal = frequency.intValue();
                    freqVal+=1;
                    map.remove(token);
                    map.put(token,freqVal);
                }
            }
        }//end inner class

        private class Database
        {
            private ArrayList ipCols;
```

```
    private ArrayList<SortedMap> tempMaster;
    private double[][] masterKeywordMatrix;
    private double[][] joiningNodeKeywords;
    private ArrayList<String> listOfKeywords;
    private Matrix U;
    private Matrix S;
    private Matrix V;
    private Matrix S_inverse;
    private Matrix V_transpose;
    private Matrix Q_transpose;
    private Matrix q;
    private Matrix simM;
    private String forIP="";

    public Database()
    {
        ipCols = new ArrayList();
        tempMaster = new ArrayList<SortedMap>();
    }

    public void add(String ip)
    {
        //ipCols.add(ip);
        if(!ipCols.contains(ip))
        {
            ipCols.add(ip);
            forIP=ip;
        }else
        {
            int index=ipCols.indexOf(ip);
            ipCols.remove(index);
            tempMaster.remove(index);
            ipCols.add(ip);
            forIP=ip;
        }
    }

    public void remove(String ip)
    {
        if(ipCols.contains(ip))
```

```
        {
            int index=ipCols.indexOf(ip);
            ipCols.remove(index);
            tempMaster.remove(index);
        }
    }

    public void addKeywords(SortedMap map)
    {
        tempMaster.add(map);
        prepareMatrix();
    }

    public void prepareMatrix()
    {
        SortedMap completeList = new TreeMap();
        SortedMap temp = new TreeMap();
        for(int i=0;i<tempMaster.size();i++)
        {
            temp=tempMaster.get(i);
            Set keywords = temp.keySet();
            Iterator itKeys = keywords.iterator();
            while(itKeys.hasNext())
            {
                String key = (String)itKeys.next();
                if(!completeList.containsKey(key))
                {
                    completeList.put(key,0);
                }
            }//end while
        }//end loop

        setCompleteList(completeList);

        masterKeywordMatrix = new double[completeList.size()][ipCols.size
            ()];
        int rows = masterKeywordMatrix.length;
        int cols = masterKeywordMatrix[0].length;
        for(int n=0;n<cols;n++)
        {
```

```
                temp= new TreeMap();
                Set completeKeys = completeList.keySet();
                Iterator it = completeKeys.iterator();
                temp=tempMaster.get(n);
                for(int m=0;m<rows;m++)
                {
                    while(it.hasNext())
                    {
                        String key = (String)it.next();
                        if(temp.containsKey(key))
                        {
                            int val = ((Integer)temp.get(key)).intValue();
                            masterKeywordMatrix[m][n]=val;
                        }else
                        {
                            masterKeywordMatrix[m][n]=0.0;
                        }
                        m++;
                    }
                }
            }
        }

        public void compute()
        {
            int rows = masterKeywordMatrix.length;
            int cols = masterKeywordMatrix[0].length;
            if(cols==1)
            {
                //do nothing
                //to protect system from issuing dummy
            }else
            {
                //display
                System.out.println("Master Matrix");
                displayMatrix(masterKeywordMatrix, cols);
                //calculate SVD considering the joining column ip address and
                    its keywords is the new query(joining node keywords).
                double[][] tempKeywordMatrix = new double[rows][cols-1];
                joiningNodeKeywords=new double[rows][1];
```

```
            int runner=0;
            for(int b=0;b<cols;b++)
            {
                for(int a=0;a<rows;a++)
                {
                    if(b==ipCols.indexOf(forIP))
                    {
                        //do not get that column
                        //make it joining node
                        //joined keywords
                        for(int z=0;z<rows;z++)
                        {
                            joiningNodeKeywords[z][0]=masterKeywordMatrix[z][b];
                        }
                    }else
                    {
                        tempKeywordMatrix[a][runner]=masterKeywordMatrix[a][b];
                    }
                }
                runner++;
            }

            System.out.println("Compared Against Matrix");
            displayMatrix(tempKeywordMatrix,tempKeywordMatrix[0].length);
            System.out.println("Joining Matrix");
            displayMatrix(joiningNodeKeywords,joiningNodeKeywords[0].length);
            Matrix Q = new Matrix(joiningNodeKeywords);
            Q_transpose=Q.transpose();
            calculateSVD(tempKeywordMatrix);
            calculateq();
            Matrix sim = calculateSim();
            System.out.println("Similarity Report");
            sim.print(sim.getColumnDimension(),3);
        }
    }

    /**
```

```
     * Displays matrix
     */
    public void displayMatrix(double[][] matrix,int cols)
    {
        Matrix mat = new Matrix(matrix);
        mat.print(cols,1);
    }

    /**
     * SVD calculation
     */
    public void calculateSVD(double[][] matrix)
    {
        Matrix mat = new Matrix(matrix);
        SingularValueDecomposition svd = mat.svd();
        U = svd.getU();
        //U.print(ipCols.size(),3);
        S = svd.getS();
        //S.print(ipCols.size(),3);
        S_inverse = S.inverse();
        V = svd.getV();
        //V.print(ipCols.size(),3);
        V_transpose= V.transpose();
    }

    /**
     * Computing query vector
     */
    public void calculateq()
    {
        q = (Q_transpose.times(U)).times(S_inverse);
    }

    public Matrix calculateSim()
    {
        double[][] vofQuery = q.getArray();
        double[][] vofTerm = V_transpose.getArray();
        double[][] sim=new double[vofQuery[0].length][1];
        double[] num=new double[vofTerm.length];
        double den1=0;
```

```
            double[] den2=new double[vofTerm.length];

            for(int x =0;x<vofTerm.length;x++)
            {
                for(int i =0;i<vofQuery[0].length;i++)
                {
                    num[x] +=vofQuery[0][i]*vofTerm[i][x];
                }
            }

            for(int x=0;x<vofQuery[0].length;x++)
            {
                den1+=vofQuery[0][x]*vofQuery[0][x];
            }

            den1 = Math.sqrt(den1);

            for(int i=0;i<den2.length;i++)
            {
                for(int x=0;x<vofTerm.length;x++)
                {
                    den2[i]+=vofTerm[x][i]*vofTerm[x][i];
                }
            }

            for(int x=0;x<den2.length;x++)
            {
                den2[x] = Math.sqrt(den2[x]);
            }

            for(int i=0;i<sim.length;i++)
            {
                sim[i][0]=num[i]/(den1*den2[i]);
            }

            simM = new Matrix(sim);
            return simM;
        }

        public Matrix getSimMatrix()
```

```
        {
            return simM;
        }

        public ArrayList getClientList()
        {
            return ipCols;
        }

        public Matrix getMasterKeywordList()
        {
            Matrix master = new Matrix(masterKeywordMatrix);
            return master;
        }

        public void setCompleteList(SortedMap completeList)
        {
            Set keys = completeList.keySet();
            listOfKeywords = new ArrayList<String>();
            Iterator it = keys.iterator();
            while(it.hasNext())
            {
                listOfKeywords.add((String)it.next());
            }
        }

        public ArrayList getKeywordList()
        {
            return listOfKeywords;
        }
    }//end inner class Message
}//end class
```

C.11 Class LocalAgent.java

```java
import jade.core.*;
import java.util.*;
import jade.core.behaviours.*;
import jade.lang.acl.*;
import Jama.*;

/**
 * Local Agent is an agent that holds information i.e.
 * keys for defining local resources and the corresponding location of
    resource on the peer.
 *
 * @author M. Singh
 * @version 1.3
 */
public class LocalAgent extends Agent
{
    private String hostaddress="";
    private String name="";
    private LocalUI ui;
    private Hashtable<String,String> table = new Hashtable<String,String>();
    private ArrayList<String> keywords = new ArrayList<String>();
    private LocalDatabase database = new LocalDatabase();
    private FrequencyFinder finder;
    protected void setup()
    {
        //welcome
        name = getAID().getName();
        System.out.println("Hello I am Local Agent and my name is "+name);

        //instance of GUI
        ui=new LocalUI();
        ui.setAgent(this);
        callAskUser();

        //behaviour
        addBehaviour(new CallForRegistration());

        //behaviour
```

```
            addBehaviour(new ServeIncomingMessage());
        }

        protected void takeDown()
        {
            ui.dispose();
            System.out.println("Local Agent "+getAID().getName()+" Terminating ");
        }

        public void callAskUser()
        {
            ui.askUser();
        }

        public void updateTable(Hashtable<String,String> catalog)
        {
            addBehaviour(new FileManager(this,catalog));
        }

        //inner class File Manager
        private class FileManager extends TickerBehaviour
        {
            private FileManager(Agent a, Hashtable<String,String> catalog)
            {
                super(a,300000);
                table=catalog;
            }

            public void onStart()
            {
                Set keys = table.keySet();
                Iterator<String> it = keys.iterator();
                while(it.hasNext())
                {
                    String key=it.next();
                    String values=table.get(key);
                    StringTokenizer st = new StringTokenizer(values);
                    while(st.hasMoreTokens())
                    {
                        keywords.add(st.nextToken());
```

```
                }
            }//end while
            System.out.println(keywords);
        }//end onstart

        public void onTick()
        {
            callAskUser();
        }
    }//end inner class File Manager

    //inner class Call for Registration
    private class CallForRegistration extends SimpleBehaviour
    {
        private MessageTemplate mt = MessageTemplate.and(MessageTemplate.MatchConversationId("keywords-request"),MessageTemplate.MatchPerformative(ACLMessage.REQUEST));
        public boolean done()
        {
            return false;
        }

        public void action()
        {
            try
            {
                ACLMessage message=myAgent.receive(mt);
                if(message!=null)
                {
                    ui.informUser("Request from "+message.getSender().getLocalName()+"\n"+"information agent to issue the keywords");
                    ACLMessage reply = message.createReply();
                    if(keywords!=null)
                    {
                        reply.setPerformative(ACLMessage.INFORM);
                        reply.setContentObject(keywords);
                    }
                    myAgent.send(reply);
                    ui.informUser(keywords!=null ? "Informed "+message.
```

```
                    getSender().getLocalName()+" about "+reply.
                    getContentObject() : "Keywords did not exist");
                }
                /*else
                {
                    block();
                }
                */
            }catch(Exception e)
            {
                e.printStackTrace();
            }
        }
    }//end inner class CallForRegistration

    //inner class Serve Incoming Message
    private class ServeIncomingMessage extends Behaviour
    {
        private MessageTemplate mt = MessageTemplate.and(MessageTemplate.
            MatchConversationId("search-request"),MessageTemplate.
            MatchPerformative(ACLMessage.REQUEST));

        public boolean done()
        {
            return false;
        }

        public void action()
        {
            try
            {
                ACLMessage request = receive(mt);
                //while(request==null)
                {
                    //request=receive(mt);
                    if(request!=null)
                    {
                        ui.informUser("Request received from "+request.
                            getSender().getLocalName());
                        String cont = request.getContent();
```

```
String[] myCont = cont.split(":");
String query=myCont[0];
double minSup=Double.parseDouble(myCont[1]);
System.out.println("THE QUERY RECEIVED BY LOCAL AGENT "+query);

//all keywords for all documents are stored in hash table -> table as (filename->keywords) as key-> value pairs
Set keys=table.keySet();
Object[] key=(Object[])keys.toArray();
for(int i=0;i<key.length;i++)
{
    ArrayList keySet = new ArrayList();
    String tempKey = (String)table.get(key[i]);
    StringTokenizer st = new StringTokenizer(tempKey);
    while(st.hasMoreTokens())
    {
        keySet.add((String)st.nextToken());
    }
    Transport message = new Transport((String)key[i], keySet);
    database.add(message.getIP());
    finder = new FrequencyFinder(message);
}
//query
StringTokenizer st = new StringTokenizer(query);
ArrayList queryList = new ArrayList();
while(st.hasMoreTokens())
{
    queryList.add((String)st.nextToken());
}
Transport mQ = new Transport("QUERY",queryList);
database.add(mQ.getIP());
finder = new FrequencyFinder(mQ);
database.compute();

//prepare reply
Matrix simR=database.getSimMatrix();
double[][] simRArray = simR.getArray();
```

```
                    int indexer=0;
                    double simVal=0.0;
                    double temp=0.0;

                    for(int q=0;q<simRArray.length;q++)
                    {
                        simVal=simRArray[q][0];
                        if(simVal>minSup && simVal>temp)
                        {
                            indexer=q;
                            temp=simVal;
                        }
                    }

                    ArrayList docs=database.getClientList();
                    String chosen = (String)docs.get(indexer);
                    System.out.println("THE CHOSEN DOCUMENT IS "+chosen);
                    MatchStore matchStore = new MatchStore(chosen,
                        getLocalName(),simVal);

                    //reply
                    ACLMessage reply = request.createReply();
                    if(chosen!=null)
                    {
                        reply.setPerformative(ACLMessage.INFORM);
                        reply.setContentObject(matchStore);
                    }
                    myAgent.send(reply);
                    System.out.println(keywords!=null ? "Informed "+
                        request.getSender().getLocalName()+" about chosen 
                        document " : "Document did not exist");
                }else
                {
                    System.out.println("No message yet");
                    block();
                }
            }//end while
        }catch(Exception e)
        {
            e.printStackTrace();
```

```
                }
            }
        }//end inner class serve incoming message

        private class Transport
        {
            private String ip_Address;
            private ArrayList keywords;

            public Transport(String ip_Address, ArrayList keywords)
            {
                this.ip_Address=ip_Address;;
                this.keywords=keywords;
            }

            public void setIP(String ip_Address)
            {
                this.ip_Address=ip_Address;
            }

            public String getIP()
            {
                return ip_Address;
            }

            public void setKeywords(ArrayList keywords)
            {
                this.keywords=keywords;
            }

            public ArrayList getKeywords()
            {
                return keywords;
            }

        }//end inner class

        //inner class to find resource
        private class FrequencyFinder
        {
```

```
    private Transport message;
    private String tk[];
    private int size;
    private SortedMap map;

    public FrequencyFinder(Transport message)
    {
        this.message=message;
        map=new TreeMap();
        getKeywords();
    }

    public void getKeywords()
    {
        int index=0;
        String keywordSet="";
        ArrayList keywords1 = message.getKeywords();
        for(int i=0;i<keywords1.size();i++)
        {
            keywordSet+=(String)keywords1.get(i)+" ";
        }
        //Tokenize
        StringTokenizer token = new StringTokenizer(keywordSet);
        size=token.countTokens();
        makeTKArray(size);
        while(token.hasMoreTokens())
        {
            tk[index]=token.nextToken();
            findTokenFrequency(tk[index]);
        }
        database.addKeywords(map);
    }

    public void makeTKArray(int size)
    {
        tk = new String[size];
    }

    public void findTokenFrequency(String token)
    {
```

```
                if(!map.containsKey(token))
                {
                    map.put(token.toLowerCase(),1);
                }else
                {
                    Integer frequency = (Integer)map.get(token);
                    int freqVal = frequency.intValue();
                    freqVal+=1;
                    map.remove(token);
                    map.put(token,freqVal);
                }
            }
        }//end inner class

        private class LocalDatabase
        {
            private ArrayList ipCols;
            private ArrayList<SortedMap> tempMaster;
            private double[][] masterKeywordMatrix;
            private double[][] joiningNodeKeywords;
            private ArrayList<String> listOfKeywords;
            private Matrix U;
            private Matrix S;
            private Matrix V;
            private Matrix S_inverse;
            private Matrix V_transpose;
            private Matrix Q_transpose;
            private Matrix q;
            private Matrix simM;
            private String forIP="";

            public LocalDatabase()
            {
                ipCols = new ArrayList();
                tempMaster = new ArrayList<SortedMap>();
            }

            public void add(String ip)
            {
                //ipCols.add(ip);
```

```
            if(!ipCols.contains(ip))
            {
                ipCols.add(ip);
                forIP=ip;
            }else
            {
                int index=ipCols.indexOf(ip);
                ipCols.remove(index);
                tempMaster.remove(index);
                ipCols.add(ip);
                forIP=ip;
            }
        }

        public void remove(String ip)
        {
            if(ipCols.contains(ip))
            {
                int index=ipCols.indexOf(ip);
                ipCols.remove(index);
                tempMaster.remove(index);
            }
        }

        public void addKeywords(SortedMap map)
        {
            tempMaster.add(map);
            prepareMatrix();
        }

        public void prepareMatrix()
        {
            SortedMap completeList = new TreeMap();
            SortedMap temp = new TreeMap();
            for(int i=0;i<tempMaster.size();i++)
            {
                temp=tempMaster.get(i);
                Set keywords = temp.keySet();
                Iterator itKeys = keywords.iterator();
                while(itKeys.hasNext())
```

```
                {
                    String key = (String)itKeys.next();
                    if(!completeList.containsKey(key))
                    {
                        completeList.put(key,0);
                    }
                }//end while
            }//end loop

            setCompleteList(completeList);

            masterKeywordMatrix = new double[completeList.size()][ipCols.size
                ()];
            int rows = masterKeywordMatrix.length;
            int cols = masterKeywordMatrix[0].length;
            for(int n=0;n<cols;n++)
            {
                temp= new TreeMap();
                Set completeKeys = completeList.keySet();
                Iterator it = completeKeys.iterator();
                temp=tempMaster.get(n);
                for(int m=0;m<rows;m++)
                {
                    while(it.hasNext())
                    {
                        String key = (String)it.next();
                        if(temp.containsKey(key))
                        {
                            int val = ((Integer)temp.get(key)).intValue();
                            masterKeywordMatrix[m][n]=val;
                        }else
                        {
                            masterKeywordMatrix[m][n]=0.0;
                        }
                        m++;
                    }
                }
            }
        }

```

```
        public void compute()
        {
            int rows = masterKeywordMatrix.length;
            int cols = masterKeywordMatrix[0].length;
            if(cols==1)
            {
                //do nothing
                //to protect system from issuing dummy
            }else
            {
                //display
                System.out.println("Master Matrix");
                displayMatrix(masterKeywordMatrix, cols);
                //calculate SVD considering the joining column ip address and
                    its keywords is the new query(joining node keywords).
                double[][] tempKeywordMatrix = new double[rows][cols-1];
                joiningNodeKeywords=new double[rows][1];
                int runner=0;
                for(int b=0;b<cols;b++)
                {
                    for(int a=0;a<rows;a++)
                    {
                        if(b==ipCols.indexOf(forIP))
                        {
                            //do not get that column
                            //make it joining node
                            //joined keywords
                            for(int z=0;z<rows;z++)
                            {
                                joiningNodeKeywords[z][0]=masterKeywordMatrix[
                                    z][b];
                            }
                        }else
                        {
                            tempKeywordMatrix[a][runner]=masterKeywordMatrix[a
                                ][b];
                        }
                    }
                    runner++;
                }
```

```
                    System.out.println("Compared Against Matrix");
                    displayMatrix(tempKeywordMatrix,tempKeywordMatrix[0].length);
                    System.out.println("Joining Matrix");
                    displayMatrix(joiningNodeKeywords,joiningNodeKeywords[0].
                        length);
                    Matrix Q = new Matrix(joiningNodeKeywords);
                    Q_transpose=Q.transpose();
                    calculateSVD(tempKeywordMatrix);
                    calculateq();
                    Matrix sim = calculateSim();
                    System.out.println("Similarity Report");
                    sim.print(sim.getColumnDimension(),3);
                }
            }

            /**
             * Displays matrix
             */
            public void displayMatrix(double[][] matrix,int cols)
            {
                Matrix mat = new Matrix(matrix);
                mat.print(cols,1);
            }

            /**
             * SVD calculation
             */
            public void calculateSVD(double[][] matrix)
            {
                Matrix mat = new Matrix(matrix);
                SingularValueDecomposition svd = mat.svd();
                U = svd.getU();
                //U.print(ipCols.size(),3);
                S = svd.getS();
                //S.print(ipCols.size(),3);
                S_inverse = S.inverse();
                V = svd.getV();
                //V.print(ipCols.size(),3);
                V_transpose= V.transpose();
```

```
    }

    /**
     * Computing query vector
     */
    public void calculateq()
    {
        q = (Q_transpose.times(U)).times(S_inverse);
    }

    public Matrix calculateSim()
    {
        double[][] vofQuery = q.getArray();
        double[][] vofTerm = V_transpose.getArray();
        double[][] sim=new double[vofQuery[0].length][1];
        double[] num=new double[vofTerm.length];
        double den1=0;
        double[] den2=new double[vofTerm.length];

        for(int x =0;x<vofTerm.length;x++)
        {
            for(int i =0;i<vofQuery[0].length;i++)
            {
                num[x] +=vofQuery[0][i]*vofTerm[i][x];
            }
        }
        System.out.println("NUMERATOR "+num[0]);
        //----------------
        for(int x=0;x<vofQuery[0].length;x++)
        {
            den1+=vofQuery[0][x]*vofQuery[0][x];
        }

        den1 = Math.sqrt(den1);
        System.out.println("DENOMINATOR PART 1 "+den1);

        for(int i=0;i<den2.length;i++)
        {
            for(int x=0;x<vofTerm.length;x++)
            {
```

```
                    den2[i]+=vofTerm[x][i]*vofTerm[x][i];
                }
            }

            for(int x=0;x<den2.length;x++)
            {
                den2[x] = Math.sqrt(den2[x]);
                System.out.println("DENOMINATOR PART 2 "+den2[x]);
            }

            for(int i=0;i<sim.length;i++)
            {
                sim[i][0]=num[i]/(den1*den2[i]);
                System.out.println("SIMILARITY CALC "+sim[i][0]);
            }

            simM = new Matrix(sim);
            return simM;
        }

        public Matrix getSimMatrix()
        {
            return simM;
        }

        public ArrayList getClientList()
        {
            return ipCols;
        }

        public Matrix getMasterKeywordList()
        {
            Matrix master = new Matrix(masterKeywordMatrix);
            return master;
        }

        public void setCompleteList(SortedMap completeList)
        {
            Set keys = completeList.keySet();
            listOfKeywords = new ArrayList<String>();
```

```
            Iterator it = keys.iterator();
            while(it.hasNext())
            {
                listOfKeywords.add((String)it.next());
            }
        }

        public ArrayList getKeywordList()
        {
            return listOfKeywords;
        }
    }//end inner class

}//end class
```

C.12 Class LocalUI.java

```
import java.io.*;
import java.util.*;
import javax.swing.*;
import jade.core.*;
/**
 * User Interface for Local Agent
 *
 * @author M. Singh
 * @version 1.1
 */
public class LocalUI extends JFrame
{
    private Hashtable<String,String> catalog;
    private LocalAgent myAgent;
    public LocalUI()
    {
        catalog=new Hashtable<String,String>();
    }

    public void setAgent(LocalAgent agent)
    {
        myAgent=agent;
    }

    public void askUser()
    {
        String option = JOptionPane.showInputDialog("Please enter YES/NO for 
            updating the catalog");
        if(option.toLowerCase().equals("yes"))
        {
            try
            {
                File folder = new File("Shared");
                File[] listOfFiles = folder.listFiles();

                for(int i=0;i<listOfFiles.length;i++)
                {
                    if(listOfFiles[i].isFile())
```

```
                    {
                        System.out.println("File "+listOfFiles[i].getName());
                        String keywords = JOptionPane.showInputDialog("Please enter the keywords describing file - "+listOfFiles[i].getName()+"\n"+"and separate using space.");
                        catalog.put(listOfFiles[i].getName(),keywords.toLowerCase());
                    }
                }
            }catch(Exception e)
            {
                e.printStackTrace();
            }
            myAgent.updateTable(catalog);
        }
    }

    public void informUser(String message)
    {
        JOptionPane.showMessageDialog(null,message);
        //System.out.println(message);
    }

}//end class
```

C.13 Class MatchStore.java

```
import java.io.*;
import java.util.*;
/**
 * Serialised Data Structure
 *
 * @author M. Singh
 * @version 1.2
 */
public class MatchStore implements Serializable
{
    String chosenDocs;
    String nameLA;
    double similarityValues;

    public MatchStore(String chosenDocs, String nameLA, double
        similarityValues)
    {
        this.chosenDocs=chosenDocs;
        this.nameLA=nameLA;
        this.similarityValues=similarityValues;
    }
}
```

C.14 Class InterfaceAgent.java

```java
import jade.core.*;
import jade.core.behaviours.*;
import jade.lang.acl.*;
import jade.gui.*;
import jade.content.*;
import jade.content.onto.basic.*;
import jade.content.lang.*;
import jade.content.lang.sl.*;
import jade.domain.*;
import jade.domain.mobility.*;
import jade.domain.JADEAgentManagement.*;
import java.util.*;

/**
 * Interface Agent is an agent that provides user interaction to the system.
 *
 * @author M. Singh
 * @version 1.7
 */
public class InterfaceAgent extends GuiAgent
{
    private String name="";
    private SearchGUI gui;
    jade.core.Runtime runtime=jade.core.Runtime.instance();
    private jade.wrapper.AgentContainer home;
    private int command;
    private int count=(int)(Math.random()*100)+3000;
    Vector agents=new Vector();

    public static final int QUIT=0;
    public static final int NEW_RECON_AGENT=1;
    public static final int KILL_AGENT=4;

    protected void setup()
    {
        //welcome
        name=getAID().getName();
        System.out.println("Interface Agent "+name+" started.");
```

```java

        //register language and ontology
        getContentManager().registerLanguage(new SLCodec());
        getContentManager().registerOntology(MobilityOntology.getInstance());

        //create agent container
        home = runtime.createAgentContainer(new ProfileImpl());
        doWait(2000);

        //start gui
        gui=new SearchGUI();
        gui.setAgent(this);
        gui.show();

        addBehaviour(new ReceiveMessageRecon());

        addBehaviour(new ReceiveTerminationRecon());
    }

    protected void onGuiEvent(GuiEvent ev)
    {
        command=ev.getType();
        if(command==QUIT)
        {
            try
            {
                home.kill();
            }catch(Exception e)
            {
                e.printStackTrace();
            }
            gui.setVisible(false);
            gui.dispose();
            doDelete();
            System.exit(0);
        }
        if(command==NEW_RECON_AGENT)
        {
            jade.wrapper.AgentController a = null;
            System.out.println("MUST_BE_CREATED");
```

```
            try
            {
                Object[] args=new Object[5];
                args[0]=getAID();
                System.out.println(args[0]);
                args[1]=gui.getQuery();//query
                args[2]="0.0";//minimum support
                args[3]=(Object)name;
                args[4]="2";//number of hops
                String name_of_Agent="Reconnaissance_Agent_"+(count++);
                a=home.createNewAgent(name_of_Agent,ReconnaissanceAgent.class.getName(),args);
                a.start();
                agents.add(name_of_Agent);
                gui.activeAgents(agents);
            }catch(Exception ee)
            {
                System.out.println("Problem while creating new agent "+ee);
            }
            return;
        }
    }

    protected void takeDown()
    {
        if(gui!=null)
        {
            gui.setVisible(false);
            gui.dispose();
        }

        System.out.println("Interface terminating for "+name+"\n"+"Thank you for using AFFINITY.");
        System.exit(0);
    }

    //inner class
    private class ReceiveMessageRecon extends CyclicBehaviour
    {
        MatchStore matchStore=null;
```

```
            MessageTemplate mt = MessageTemplate.and(MessageTemplate.
                MatchConversationId("results"),MessageTemplate.MatchPerformative(
                ACLMessage.INFORM));
            public void action()
            {
                try
                {
                    ACLMessage reply = receive(mt);
                    if(reply!=null)
                    {
                        matchStore=(MatchStore)reply.getContentObject();
                        gui.setResult(matchStore);
                    }else
                    {
                        block();
                    }
                }catch(Exception e)
                {
                    e.printStackTrace();
                }
            }
        }

        //inner class
        private class ReceiveTerminationRecon extends SimpleBehaviour
        {
            private boolean check=false;
            MessageTemplate mt = MessageTemplate.and(MessageTemplate.
                MatchConversationId("termination-instruction"),MessageTemplate.
                MatchPerformative(ACLMessage.INFORM));

            public boolean done()
            {
                return check;
            }

            public void action()
            {
                try
                {
```

```
                ACLMessage message = receive(mt);
                if(message!=null)
                {
                    agents.remove(message.getSender().getLocalName());
                    gui.activeAgents(agents);
                    check=true;
                }else
                {
                    block();
                }
            }catch(Exception e)
            {
                e.printStackTrace();
            }
        }
    }
}//end class
```

C.15 Class SearchGUI.java

```java
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.*;
import javax.swing.event.*;
import jade.core.*;
import java.util.*;
import jade.gui.*;

/**
 * SearchGUI is User interface for Interface Agent
 *
 */
public class SearchGUI extends JFrame
{
        private InterfaceAgent myAgent;
        private String query="";
    // Variables declaration
        private JLabel jLabel2;
        private JLabel jLabel3;
        private JTextArea jTextArea1;
        private JScrollPane jScrollPane3;
        private JList jList1;
        private DefaultListModel listModel1;
        private JScrollPane jScrollPane2;
        private JTabbedPane jTabbedPane1;
        private JPanel contentPane;
        //-----
        private JLabel jLabel1;
        private JTextField jTextField1;
        private JButton jButton1;
        private JPanel jPanel1;
        //-----
        // End of variables declaration

        public void setAgent(InterfaceAgent a)
        {
            myAgent=a;
```

```
            setTitle("Affinity - Search Node - "+myAgent.getName());
        }

        public SearchGUI()
        {
                super();
                JFrame.setDefaultLookAndFeelDecorated(true);
                JDialog.setDefaultLookAndFeelDecorated(true);
                try
                {
                        UIManager.setLookAndFeel("com.sun.java.swing.plaf.windows.WindowsLookAndFeel");
                }
                catch (Exception ex)
                {
                        System.out.println("Failed loading L&F: ");
                        System.out.println(ex);
                }
                addWindowListener(new WindowAdapter() {
                    public void windowClosing(WindowEvent e) {
                        myAgent.doDelete();
                    }
                });
                initializeComponent();
        }

        private void initializeComponent()
        {
                jLabel2 = new JLabel();
                jLabel3 = new JLabel();
                jTextArea1 = new JTextArea();
                jScrollPane3 = new JScrollPane();
                listModel1 = new DefaultListModel();
                jList1 = new JList(listModel1);
                jScrollPane2 = new JScrollPane();
                jTabbedPane1 = new JTabbedPane();
                contentPane = (JPanel)this.getContentPane();
                //-----
                jLabel1 = new JLabel();
                jTextField1 = new JTextField();
```

```
jButton1 = new JButton();
jPanel1 = new JPanel();
//-------

//
// jLabel2
//
jLabel2.setText("Search Results for Query: ");
//
// jLabel2
//
jLabel3.setText("Active Reconnaissance Agents");
//
// jTextArea1
//
jTextArea1.setFont(new java.awt.Font("Tahoma", 0, 11));
jTextArea1.setToolTipText("Search Results");
jTextArea1.setEditable(false);
jTextArea1.setLineWrap(true);
//
// jScrollPane3
//
jScrollPane3.setViewportView(jTextArea1);
//
// jList1
//
jList1.setVisibleRowCount(7);
jList1.setFixedCellHeight(18);
jList1.setSelectionMode(ListSelectionModel.
    SINGLE_INTERVAL_SELECTION);
//
// jScrollPane2
//
jScrollPane2.setViewportView(jList1);
//
// jTabbedPane1
//
jTabbedPane1.addTab("Search", jPanel1);
jTabbedPane1.setBackground(new Color(255, 255, 255));
jTabbedPane1.addChangeListener(new ChangeListener() {
```

```
                        public void stateChanged(ChangeEvent e)
                        {
                                jTabbedPane1_stateChanged(e);
                        }

                });
                //
                // contentPane
                //
                contentPane.setLayout(null);
                contentPane.setBorder(BorderFactory.createRaisedBevelBorder())
                    ;
                addComponent(contentPane, jLabel2, 211,15,337,18);
                addComponent(contentPane, jLabel3, 550,15,157,18);
                addComponent(contentPane, jScrollPane3, 210,33,337,328);
                addComponent(contentPane, jScrollPane2, 550,33,157,100);
                addComponent(contentPane, jTabbedPane1, 4,11,200,350);
                //
                // jLabel1
                //
                jLabel1.setText("Enter Search Query");
                //
                // jButton1
                //
                jButton1.setText("Search");
                jButton1.setToolTipText("Click to Start Search");
                jButton1.addActionListener(new ActionListener() {
                        public void actionPerformed(ActionEvent e)
                        {
                                jButton1_actionPerformed(e);
                        }

                });
                //
                // jPanel1
                //
                jPanel1.setLayout(null);
                jPanel1.setBorder(new TitledBorder("Search Query Window"));
                jPanel1.setBackground(new Color(255, 254, 254));
                jPanel1.setOpaque(false);
```

```
                jPanel1.setToolTipText("Search");
                addComponent(jPanel1, jLabel1, 5,50,100,18);
                addComponent(jPanel1, jTextField1, 5,70,180,22);
                addComponent(jPanel1, jButton1, 48,92,83,28);
                //
                // SearchGUI
                //
                this.setLocation(new Point(0, 0));
                this.setSize(new Dimension(730, 400));
                this.setResizable(false);
        }

        /** Add Component Without a Layout Manager (Absolute Positioning) */
        private void addComponent(Container container,Component c,int x,int y,int width,int height)
        {
                c.setBounds(x,y,width,height);
                container.add(c);
        }

        private void jTabbedPane1_stateChanged(ChangeEvent e)
        {
                System.out.println("NOTHING SHOULD BE HAPPENING HERE");
        }

        private void jButton1_actionPerformed(ActionEvent e)
        {
                query = jTextField1.getText();
                jTextField1.setText("");
                jLabel2.setText("Search Results for Query: "+query);
                GuiEvent ge = new GuiEvent(this,myAgent.NEW_RECON_AGENT);
                myAgent.postGuiEvent(ge);
        }

        public Object getQuery()
        {
            return (Object)query;
        }

        public void activeAgents(Vector agents)
```

```
        {
            listModel1.clear();
            for(int i=0;i<agents.size();i++)
            {
                listModel1.addElement(agents.get(i));
            }
        }

        public void setResult(MatchStore matchStore)
        {
            jTextArea1.append("Manu Manu Manu\n");
            jTextArea1.append(matchStore.nameLA+"\n");
            jTextArea1.append(matchStore.chosenDocs+"\n");
            jTextArea1.append(""+matchStore.similarityValues+"\n");
            jTextArea1.append("\n");
        }
}//end class
```

C.16 Class ReconnaissanceAgent.java

```
import jade.core.*;
import jade.core.behaviours.*;
import jade.lang.acl.*;
import jade.domain.*;
import jade.domain.mobility.*;
import jade.domain.JADEAgentManagement.WhereIsAgentAction;
import jade.domain.JADEAgentManagement.KillAgent;
import jade.content.*;
import jade.content.onto.basic.*;
import jade.content.lang.*;
import jade.content.lang.sl.*;
import java.util.*;
/**
 * ReconnaissanceAgent is a mobile agent that is created by the Interface
    Agent
 * upon user search request.
 *
 * @author M. Singh
 * @version 2.5
 */
public class ReconnaissanceAgent extends Agent
{
    private String creator="";
    private String query="";
    private String minSup="";
    private int maxHops;
    private String nameIA="";
    private String cont="";
    private Map locations=new HashMap();
    private String destName="";
    private String destLAName="";
    private int hopNumber=0;

    protected void setup()
    {
        //register language and ontology
        getContentManager().registerLanguage(new SLCodec());
        getContentManager().registerOntology(MobilityOntology.getInstance());
```

```
            System.out.println("Hi, I am Reconnaissance Agent - "+getLocalName());

            //get arguments passed while creation of reconnaissance agent
            Object[] args = getArguments();
            creator=(String)args[3];
            if(creator.startsWith("S"))
            {
                nameIA=creator;
                nameIA=nameIA.replace("S","I");
            }
            query=(String)args[1];
            minSup=(String)args[2];
            cont=query+":"+minSup;
            maxHops=Integer.parseInt((String)args[4]);

            //request location
            String nameofAgent=getNode(nameIA);
            destLAName=nameofAgent;
            System.out.println(nameofAgent);
            commForJump(nameofAgent);

        }

        protected void takeDown()
        {
            System.out.println("Terminating Myself");
        }

        protected void afterMove()
        {
            //register language and ontology
            getContentManager().registerLanguage(new SLCodec());
            getContentManager().registerOntology(MobilityOntology.getInstance());

            hopNumber++;
            //1. recon agent has to find local agent and compare the query against the catalog it is keeping
            //if any of the results are good using MinSup it Sends ACL Message to creator (Interface Agent)
```

```
        //informing about the find (possible name of file and its name of local agent keep it.

        //Send Message to LA
        System.out.println("THE DESTINATION LOCAL AGENT IS "+destLAName);
        ACLMessage request=new ACLMessage(ACLMessage.REQUEST);
        request.addReceiver(new AID(destLAName,AID.ISGUID));
        request.setConversationId("search-request");
        request.setReplyWith("request"+System.currentTimeMillis());
        request.setContent(cont);
        send(request);
        System.out.println("Message sent to "+destLAName);

        //Prepare message receiving template from LA about the matches found
        MessageTemplate mt = MessageTemplate.and(MessageTemplate.MatchConversationId("search-request"),MessageTemplate.MatchInReplyTo(request.getReplyWith()));
        ACLMessage reply = blockingReceive(mt);
        MatchStore matchStore=null;
        if(reply!=null)
        {
            if(reply.getPerformative()==ACLMessage.INFORM)
            {
                try
                {
                    matchStore = (MatchStore)reply.getContentObject();
                }catch(Exception e)
                {
                    e.printStackTrace();
                }
            }
        }

        //prepare to send message to interface agent (home) about the matches found
        try
        {
            System.out.println("Sending Message to home");
            ACLMessage inform = new ACLMessage(ACLMessage.INFORM);
            inform.addReceiver(new AID(creator,AID.ISGUID));
```

```
            inform.setConversationId("results");
            inform.setContentObject(matchStore);
            send(inform);
        }catch(Exception e)
        {
            e.printStackTrace();
        }

        //2. recon agent checks if it has made number of jumps less than maximum number of hops allowed.
        //if it is less then it communicate to information agent here on this node and get the next jump
        //address and conatiner
        //else it kills itself.
        if(hopNumber<maxHops)
        {
            System.out.println("TIME TO JUMP TO NEXT DESTINATION");
            String nameI="";
            //request location
            if(creator.startsWith("S"))
            {
                nameI=destLAName;
                nameI=nameI.replace("L","I");
            }
            String nameofAgent=getNode(nameI);
            destLAName=nameofAgent;
            String creatorLA="";
            if(creator.startsWith("S"))
            {
                creatorLA=creator;
                creatorLA=creatorLA.replace("S","L");
            }
            if (!destLAName.equals(creatorLA))
            {
                System.out.println("NEXT JUMP IS TOWARDS CONTAINER CONTAINING AGENT NAME:- "+nameofAgent);
                //jumping time
                commForJump(nameofAgent);
            }else
            {
```

```
                    System.out.println("NO_SUITABLE_NODES_FOUND");
                    doDelete();
                }
            }else
            {
                //Preparing to die
                ACLMessage message = new ACLMessage(ACLMessage.INFORM);
                message.addReceiver(new AID(creator,AID.ISGUID));
                message.setConversationId("termination-instruction");
                send(message);
                //time to die

                System.out.println("Terminating_Myself");

                /*
                KillAgent ka=new KillAgent();
                ka.setAgent(getAID());
                sendRequest(new Action(getAID(),ka));
                */
                doWait(3000);
                doDelete();
            }
        }


        public void commForJump(String nameofAgent)
        {
            try
            {
                AID aid = new AID(nameofAgent,AID.ISGUID);
                WhereIsAgentAction where = new WhereIsAgentAction();
                where.setAgentIdentifier(aid);
                //send message to AMS
                sendRequest(new Action(getAMS(), where));

                //receiving message from AMS
                MessageTemplate mt = MessageTemplate.and(MessageTemplate.
                    MatchSender(getAMS()),MessageTemplate.MatchPerformative(
                    ACLMessage.INFORM));
                ACLMessage resp = blockingReceive(mt);
```

```
                ContentElement ce = getContentManager().extractContent(resp);
                Result result = (Result)ce;
                jade.util.leap.Iterator it = result.getItems().iterator();
                while(it.hasNext())
                {
                    Location loc=(Location)it.next();
                    locations.put(loc.getName(),loc);
                    destName=loc.getName();
                }

                doWait(5000);
                System.out.println("Wait_Finished");
                //name of agent to be tranfered that is reconnaissance agent
                    itself
                AID aidi = new AID(getLocalName(),AID.ISLOCALNAME);
                Location dest = (Location)locations.get(destName);
                MobileAgentDescription mad = new MobileAgentDescription();
                mad.setName(aid);
                mad.setDestination(dest);
                MoveAction ma = new MoveAction();
                ma.setMobileAgentDescription(mad);
                sendRequest(new Action(aid,ma));
                doMove(dest);
                System.out.println("SHOULD_HAVE_MOVED_BY_NOW");
            }catch(Exception e)
            {
                e.printStackTrace();
            }
        }

        //get node
        public String getNode(String agentName)
        {
            String nodeName="";

            //Send Message to IA
            ACLMessage request=new ACLMessage(ACLMessage.REQUEST);
            request.addReceiver(new AID(agentName,AID.ISGUID));
            request.setConversationId("node-request");
            request.setReplyWith("request"+System.currentTimeMillis());
```

```
        request.setContent(cont);
        send(request);

        //Prepare message receiving template
        MessageTemplate mt = MessageTemplate.and(MessageTemplate.MatchConversationId("node-request"),MessageTemplate.MatchInReplyTo(request.getReplyWith()));
        ACLMessage reply = blockingReceive(mt);

        if(reply!=null)
        {
            if(reply.getPerformative()==ACLMessage.INFORM)
            {
                try
                {
                    nodeName=(String)reply.getContent();
                }catch(Exception e)
                {
                    e.printStackTrace();
                }
            }
        }

        return nodeName;
    }//end get Node

    //send message to AMS for location of the named static agent
    public void sendRequest(Action action)
    {
        ACLMessage request = new ACLMessage(ACLMessage.REQUEST);
        request.setLanguage(new SLCodec().getName());
        request.setOntology(MobilityOntology.getInstance().getName());
        try
        {
            getContentManager().fillContent(request,action);
            request.addReceiver(action.getActor());
            send(request);
        }catch(Exception e)
        {
            e.printStackTrace();
```

```
            }
        }

}//end class
```