

STATUS OF THESIS

Title of thesis:

Semantic-Based, Scalable, Decentralized and Dynamic Resource
Discovery for Internet-Based Distributed System

I

MAHAMAT ISSA HASSAN

Hereby allow my thesis to be placed at the Information Resource Center (IRC) of
Universiti Teknologi PETRONAS (UTP) with the following conditions:

1. The thesis becomes the property of UTP
2. The IRC of UTP may make copies of the thesis for academic purposes only.
3. This thesis is classified as

☐

Confidential

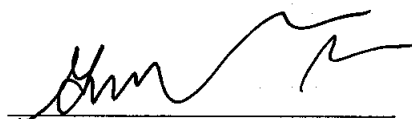
☒

Non-confidential

If this thesis is confidential, please state the reason:


The contents of the thesis will remain confidential for _____ years.

Remarks on disclosure:



Mahamat Issa Hassan
Department of Computer
Information Sciences (CIS)
Universiti Teknologi PETRONAS
Date 20/4/10

Endorsed by



Azween Bin Abdullah
Department of Computer
Information Sciences (CIS)
Universiti Teknologi PETRONAS
Date 20/9/10

UNIVERSITI TEKNOLOGI PETRONAS

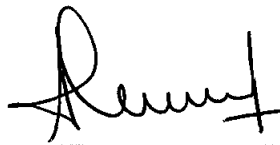
SEMANTIC-BASED, SCALABLE, DECENTRALIZED AND DYNAMIC
RESOURCE DISCOVERY FOR INTERNET-BASED DISTRIBUTED SYSTEM

By

MAHAMAT ISSA HASSAN

The undersigned certify that they have read, and recommend to the Postgraduate Studies Programme for acceptance this thesis for the fullfilment of the requirements for the degree stated.

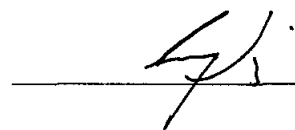
Signature:



Main Supervisor:

Assoc. Prof. Dr Azween Bin Abdullah

Signature:



Dr Mohd Fadzil B Hassan
Head
Computer & Information Sciences Department
Universiti Teknologi PETRONAS

Head of Department:

Dr. Mohd Fadzil Bin Hassan

Date:

20/9/10

SEMANTIC-BASED, SCALABLE, DECENTRALIZED AND DYNAMIC
RESOURCE DISCOVERY FOR INTERNET-BASED DISTRIBUTED SYSTEM

by

MAHAMAT ISSA HASSAN

A Thesis

Submitted to the Postgraduate Studies Programme

As a requirement for the degree of

DOCTOR OF PHILOSOPHY

DEPARTMENT OF COMPUTER AND INFORMATION SCIENCES

UNIVERSITI TEKNOLOGI PETRONAS

BANDAR SERI ISKANDAR

PERAK

SEPTEMBER 2010

DECLARATION OF THESIS

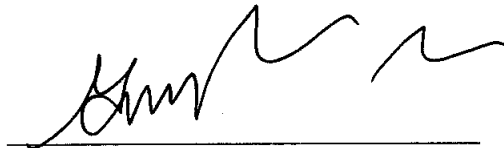
Title of thesis

Semantic-Based, Scalable, Decentralized and Dynamic Resource
Discovery for Internet-Based Distributed System

I MAHAMAT ISSA HASSAN

hereby declare that the thesis is based on my original work except for quotations and citations which have been duly acknowledged. I also declare that it has not been previously or concurrently submitted for any other degree at UTP or other institutions.

Witnessed by



Signature of Author



Signature of Supervisor

Permanent address:

Université Roi Fayçal
B.P. 582, Ndjamenà, Chad

Name of Supervisor

Azween Bin Abdullah

Date : 20/9/10

Date : 20/9/10

ACKNOWLEDGEMENT

First and famous I would like to express my gratitude to the Almighty Creator for guiding me throughout my lifetime.

My sincere appreciation goes to my entire family members for their daily prayers and love for the duration of my academic years in UTP. Their kindness made this work possible.

I also want to acknowledge the financial support I have received from UTP that made possible for me to be a graduate student and finished my work. I am profoundly thankful to my supervisor Assoc. Prof. Dr. Azween Bin Abdullah for his kindly support and intellectual guidance. He has taught me to improve and trust my research skills, as well as the intricacies of academia and how to navigate them. I am also very grateful to my other lecturers and the entire staff of my department for their support during my study period.

My research lab has been a wonderful place to work on, and I want to say ‘thank you’ to all the lab mates. I am indebted to them.

Finally may I cease this opportunity to extend my warm appreciation to my parents: Issa Hassan and my mother Radya Mahamat who have been my true parents supporting in good and bad times during the last 28 years. Their analytical mind and critical thinking has constantly challenged me to achieving better results in my career, I am proud of them.

ABSTRACT

Resource Discovery (RD) is a key issue in Internet-based distributed systems such as grid. RD is about locating an appropriate resource/service type that matches the user's application requirements. This is very important, as resource reservation and task scheduling are based on it. Unfortunately, RD in grid is very challenging as resources and users are distributed, resources are heterogeneous in their platforms, status of the resources is dynamic (resources can join or leave the system without any prior notice) and most recently the introduction of a new type of grid called intergrid (grid of grids) with the use of multi middlewares. Such situation requires an RD system that has rich interoperability, scalability, decentralization and dynamism features. However, existing grid RD systems have difficulties to attain these features. Not only that, they lack the review and evaluation studies, which may highlight the gap in achieving the required features. Therefore, this work discusses the problem associated with intergrid RD from two perspectives. First, reviewing and classifying the current grid RD systems in such a way that may be useful for discussing and comparing them. Second, propose a novel RD framework that has the aforementioned required RD features. In the former, we mainly focus on the studies that aim to achieve interoperability in the first place, which are known as RD systems that use semantic information (semantic technology). In particular, we classify such systems based on their qualitative use of the semantic information. We evaluate the classified studies based on their degree of accomplishment of interoperability and the other RD requirements, and draw the future research direction of this field. Meanwhile in the latter, we name the new framework as *semantic-based scalable decentralized dynamic RD*. The framework further contains two main components which are *service description*, and *service registration and discovery* models. The earlier consists of a set of ontologies and services. Ontologies are used as a data model for service description, whereas the services are to accomplish the description process. The service registration is also

based on ontology, where nodes of the service (service providers) are classified to some classes according to the ontology concepts, which means each class represents a concept in the ontology. Each class has a head, which is elected among its own class nodes/members. Head plays the role of a registry in its class and communicates with the other heads of the classes in a peer to peer manner during the discovery process. We further introduce two intelligent agents to automate the discovery process which are *Request Agent* (RA) and *Description Agent* (DA). Each node is supposed to have both agents. DA describes the service capabilities based on the ontology, and RA carries the service requests based on the ontology as well. We design a service search algorithm for the RA that starts the service look up from the class of request origin first, then to the other classes.

We finally evaluate the performance of our framework with extensive simulation experiments, the result of which confirms the effectiveness of the proposed system in satisfying the required RD features (interoperability, scalability, decentralization and dynamism). In short, our main contributions are outlined new key taxonomy for the semantic-based grid RD studies; an interoperable semantic description RD component model for intergrid services metadata representation; a semantic distributed registry architecture for indexing service metadata; and an agent-based service search and selection algorithm.

ABSTRAK

Penemuan Sumber atau *Resource Discovery (RD)* adalah isu utama di dalam sistem teragih berasaskan Internet. RD bertujuan untuk menempatkan jenis sumber/perkhidmatan yang sepadan dengan keperluan pengguna di lokasi yang sesuai. Ini adalah amat penting dalam penyimpanan sumber dan penjadualan tugas. Namun begitu, grid di dalam RD adalah amat mencabar kerana sumber dan pengguna adalah teragih, sumber di platform adalah heterogeneous (pelbagai), status sumber yang dinamik (sumber boleh menyambung atau meninggalkan sistem tanpa notis pemberitahuan) dan yang terkini adalah pengenalan kepada sejenis grid dipanggil intergrid (grid di dalam grid) yang menggunakan pelbagai pengantaraan. Situasi-situasi sedemikian memerlukan sistem RD yang mempunyai ciri-ciri interoperasi, berskala, tidak terpusat dan dinamik. Kajian-kajian yang lalu mendapati, grid sistem RD sedia ada mempunyai kesukaran untuk mengekalkan ciri-ciri yang dinyatakan. Selain itu, terdapat kekurangan pada sistem sedia ada dalam penyemakan semula dan penilaian yang menyebabkan kepada jurang untuk mencapai ciri-ciri yang dikehendaki.

Kajian ini membincangkan masalah-masalah integriti RD dalam dua perspektif. Pertama, penyemakan semula dan pengklasifikasi grid sistem RD sedia ada yang berguna dalam perbincangan dan perbandingan. Kedua, mencadangkan kerangka RD baru yang mempunyai maklumat sebelumnya yang diperlukan dalam RD. Sebelum ini, kami hanya menumpukan kajian untuk mencapai interoperasi yang dikenali sebagai maklumat semantik/ teknologi semantik sistem RD. Secara khususnya, kami mengklasifikasikan sistem-sistem tersebut berdasarkan fungsi kualitatif maklumat semantik. Kami menilai maklumat yang telah diklasifikasi kepada darjah pencapaian interoperasi dan keperluan-keperluan lain serta potensi RD. Pada penghujungnya, kami menamakan kerangka baru ini sebagai RD dinamik berskala tidak terpusat berasaskan semantik atau "*semantic-based scalable decentralized dynamic RD*".

Kerangka baru ini mengandung dua komponen utama iaitu penerangan perkhidmatan atau *service description* dan model pendaftaran dan penemuan perkhidmatan atau *service registration and discovery models*. Komponen pertama mengandungi urutan set-set ontologi dan perkhidmatan. Ontologi digunakan sebagai model data untuk penerangan sumber yang diperlukan ketika proses penerangan. Komponen kedua, iaitu pendaftaran perkhidmatan turut berdasarkan ontologi, yang mana nod-nod perkhidmatan (penyumbang perkhidmatan atau *service provider*) diklasifikasikan mengikut konsep ontologi iaitu setiap kelas memaparkan satu konsep ontologi. Setiap kelas mempunyai kepala atau *head* yang dipilih daripada nod kelas/ ahli sendiri. *Head* berperanan sebagai pendaftar di dalam kelas dan berkomunikasi dengan *head* kelas-kelas lain secara *peer-to-peer* semasa proses pencarian. Seterusnya, kami memperkenalkan dua agen bijak untuk mengautomasikan proses pencarian iaitu Agen Pemohon atau *Request Agent* (RA) dan Agen Penerangan atau *Description Agent* (DA). Setiap nod perlu mempunyai kedua-dua agen ini. DA menerangkan kebolehan perkhidmatan berdasarkan ontologi manakala RA membawa permohonan perkhidmatan berdasarkan ontologi. Seterusnya, algoritma pencarian perkhidmatan RA direkabentuk bermula dari kelas asal sehingga kelas-kelas yang lain.

Akhirnya, kami menilai prestasi kerangka dengan simulasi-simulasi eksperimen bagi memastikan keberkesanan sistem yang dicadangkan memenuhi ciri-ciri RD yang dikehendaki iaitu interoperasi, berskala, tidak terpusat, dan dinamik. Secara kesimpulannya, sumbangan besar kami dalam penyelidikan ini adalah kunci taksonomi baru bagi pengajian grid RD yang berdasarkan semantik; model komponen penghuraian RD semantic interoperasi bagi mewakili metadata perkhidmatan intergrid; rekabentuk daftar semantic tersebar bagi perkhidmatan indeks metadata; dan perkhidmatan ejen pencarian serta algoritma pemilihan.

In compliance with the terms of the Copyright Act 1987 and the IP Policy of the university, the copyright of this thesis has been reassigned by the author to the legal entity of the university,

Institute of Technology PETRONAS Sdn Bhd.

Due acknowledgement shall always be made of the use of any material contained in, or derived from, this thesis.

© Mahamat Issa Hassan, 2010

Institute of Technology PETRONAS Sdn Bhd

All rights reserved.

TABLE OF CONTENTS

STATUS OF THESIS.....	i
APPROVAL PAGE.....	ii
TITLE PAGE.....	iii
DECLARATION OF THESIS.....	iv
DEDICATION.....	v
ACKNOWLEDGEMENT	vi
ABSTRACT	vii
ABSTRAK.....	ix
TABLE OF CONTENTS.....	xii
LIST OF FIGURES.....	xvi
LIST OF TABLES.....	xix
LIST OF ABBREVIATIONS.....	xxi
CHAPTER 1: INTRODUCTION	1
1.1 Introduction	1
1.2 Motivation	1
1.3 Objectives.....	4
1.4 Research Questions	6
1.5 Methodology	8
1.6 Our Contributions.....	9
1.7 Organization of the Thesis	10
CHAPTER 2: BACKGROUND KNOWLEDGE.....	12
2.1 Introduction	12
2.1 Grid Computing.....	13
2.1.1 Grid Requirements.....	15

2.1.2	Grid Architecture	16
2.1.3	Grid Middleware.....	17
2.1.4	Grid Types	19
2.2	Grid RD Enabled Technologies and Related Grid Components	21
2.2.1	Peer-to-Peer Computing (P2P)	22
2.2.2	Intelligent Agent	23
2.2.3	Broker and Meta-Broker.....	23
2.3	Grid Resource Discovery: The Big Picture	24
2.3.1	Grid RD Components	26
2.4	Grid RD Requirements	29
2.5	Existing Grid RD Systems	30
2.5.1	Existing RD Systems' Description	31
2.5.2	Existing RD Systems' Registration	35
2.5.3	Existing RD Systems' Discovery	39
2.6	Assessment Summary of the Existing RD Systems	40

CHAPTER 3: THE STATE OF THE ART IN SEMANTIC-BASED GRID RD SYSTEMS..... 41

3.1	Introduction	41
3.2	Methodology	42
3.3	Semantic Technology	43
3.3.1	Ontology Languages	44
3.4	Semantic Technology and Its Use in Grid Technology.....	51
3.4.1	Semantic Grid	51
3.4.2	Grid Domain Ontologies.....	53
3.4.3	Semantic-Based Grid RD Systems' Description	57
3.4.4	Semantic-Based Grid RD Systems' Registration	66
3.4.5	Semantic-Based Grid RD Systems' Discovery	70
3.5	Discussion and Comparison Summary.....	73
3.5.1	Interoperability	74
3.5.2	Scalability, Decentralization and Dynamism	75
3.6	Semantic-Based RD Systems and Emerging Grids and Clouds.....	77
3.7	Related Work.....	78

3.7.1	A Taxonomy of Grid Monitoring Systems	78
3.7.2	Peer-to-Peer RD in Grids	79
3.7.3	Peer-to-Peer Based RD in Global Grids	80
3.7.4	Summary	81
CHAPTER 4: SEMANTIC-BASED RESOURCE DESCRIPTION MODEL....		82
4.1	Introduction	82
4.2	Methodology	83
4.3	Foundation of the Semantic Description Model	84
4.3.1	Intergrid Services	85
4.3.2	Service Grid Information Aggregation Mechanism	88
4.3.3	Service Grid Information Representation	89
4.3.4	Service Grid Request Formulation	92
4.3.5	Service Grid Information Manipulation	95
4.4	The Description of Model Building Block	96
4.4.1	Semantic Description Manager (SDM)	97
4.4.2	Service Grid Metadata Provider (SGMP)	98
4.5	The Description Process	100
4.6	Evaluation	101
CHAPTER 5: SEMANTIC REGISTRATION AND DISCOVERY MODEL..		103
5.1	Introduction	103
5.2	Methodology	104
5.3	The Model Components	104
5.3.1	The Dictionary Ontology	105
5.3.2	Intelligent Agent	106
5.4	The Model Description	107
5.4.1	The Registry Architecture	107
5.4.2	Fault Tolerance and Load Balancing Strategy	112
5.4.3	The Discovery Algorithm	118
5.4.4	Application	122
5.5	Computational Complexity of the New RD System	123
5.6	Discussion	126

CHAPTER 6: RESULTS AND DISCUSSION..... 128

6.1 Introduction 128

6.2 Methodology 128

6.3 Grid Simulation Tools 129

6.4 PeerfactSim.KOM 130

6.5 Experimental Setup 131

6.6 Performance Indices 133

6.7 Performance of the New RD Framework..... 134

6.8 Comparative Study 151

CHAPTER 7: CONCLUSION 155

7.1 Introduction 155

7.2 Contributions 156

7.3 Limitations and Future Work 157

REFERENCES 159

Appendix A 170

Appendix B..... 178

Appendix C..... 231

LIST OF FIGURES

Figure 2.1 The grid architecture	17
Figure 2.2 Grid types classified by size, source (http://www.oracle.com)	20
Figure 2.3 The autonomy of RD system with other grid components	24
Figure 2.4 Grid RD components and their interactions.....	26
Figure 2.5 The relation between Resource modeling aspects	27
Figure 2.6 Grid RD systems taxonomy based on the technologies used	31
Figure 2.7 The Condor RD centralized registration	35
Figure 2.8 The Globus MDS hierarchical registration model	36
Figure 3.1 The taxonomy of ontology languages.....	44
Figure 3.2 SAWDL overview, source (Jacek, Tomas et al. 2007).....	49
Figure 3.3 The Service Ontology Model.....	50
Figure 3.4 S-OGSA entities and their relationships	52
Figure 3.5 The main class grid resource ontology that propose by (Pernas and Dantas 2005)	53
Figure 3.6 Overview of the Core Grid Ontology classes	54
Figure 3.7 The grid knowledge architecture	56
Figure 3.8 The taxonomy of semantic-based RD system description.....	58
Figure 3.9 The grid architecture using the ontology approach as proposed by (Pernas and Dantas 2005)	59
Figure 3.10 The semantic-based RD model presented by (Somasundaram et al. 2006)	61
Figure 3.11 The S-MDS system architecture proposed by (Said and Kojima 2009)...	62
Figure 3.12 Overview of the active ontology architecture.....	63
Figure 3.13 The taxonomy of semantic-based RD systems registration	66
Figure 3.14 (A) A centralized registry on top one hierarchical information service; (B) centralized registry on top of two hierarchical information services ..	69

Figure 4.1 The relationship between providers and consumers at intergrid level.....	87
Figure 4.2 Fragment of service grid domain ontology	90
Figure 4.3 The extraction of application goals from the service grid domain ontology	93
Figure 4.4 The description of model building block	96
Figure 4.5 The data exchange between the service grid information and the semantic metadata repository.....	100
Figure 5.1 A fragment of the Dictionary Ontology	105
Figure 5.2 The proposed DA and RA agents	106
Figure 5.3 Class Formulation Algorithm	108
Figure 5.4 Head Appointment Algorithm	109
Figure 5.5 Node Subscription Algorithm	111
Figure 5.6 Head Replacement Algorithm.....	113
Figure 5.7 Member Replacement Algorithm	114
Figure 5.8 Class Management Algorithm	117
Figure 5.9 The overall intergrid system	118
Figure 5.10 The Discovery Algorithm	121
Figure 6.1 The layered architecture of PeerfactSim.KOM.....	130
Figure 6.2 Proportion of the generated service requests to intergrid size	134
Figure 6.3 Discovered Services for generated requests equivalent to 25% of the intergrid size with different TTL values	135
Figure 6.4 Discovered Services for generated requests equivalent to 50% of the intergrid size with different TTL values	137
Figure 6.5 Discovered Services for generated requests equivalent to 75% of the intergrid size with different TTL values	138
Figure 6.6 Discovered Services for generated requests equivalent to 100% of the intergrid size with different TTL values	140
Figure 6.7 Service Request Response Time for generated requests equivalent to 25% of the intergrid size with different TTL values.....	142
Figure 6.8 Service Request Response Time for generated requests that equivalent to 50% of the intergrid size with different TTL values	143

Figure 6.9 Service Request Response Time for generated requests equivalent to 75% of the intergrid size with different TTL values144

Figure 6.10 Service Request Response Time for generated requests equivalent to 100% of the intergrid size with different TTL values.....145

Figure 6.11 Average Hops for generated requests equivalent to 25% of the intergrid size with different TTL values147

Figure 6.12 Average Hops for generated requests equivalent to 50% of the intergrid size with different TTL values148

Figure 6.13 Average Hops for generated requests equivalent to 75% of the intergrid size with different TTL values149

Figure 6.14 Average Hops for generated requests equivalent to 100% of the intergrid size with different TTL values150

Figure 6.15 Discovered Services for generated requests equivalent to 100% of the intergrid obtained with the super-peer model and the semantic RD model153

Figure 6.16 Average Response Time with the super-peer model and the semantic RD model153

Figure 6.17 Average Hops obtained with the super-peer model and the semantic RD model154

LIST OF TABLES

Table 3.1 Comparison summary of semantic-based RD systems description.....	72
Table 3.2 Comparison summary of semantic-based RD systems registration	73
Table 3.3 A summary of the grid monitoring systems with their levels	79
Table 3.4 A summary of the P2P and semantic information based grid RD studies ..	80
Table 6.1 Simulation parameters.....	132
Table 6.2 Percentage of Discovered Services (SRH) for generated requests equivalent to 25% of the intergrid size with different TTL values (2-5).	135
Table 6.3 Percentage of Discovered Services (SRH) for generated requests equivalent to 50% of the intergrid size with different TTL values (2-5).	136
Table 6.4 Percentage of Discovered Services (SRH) for generated requests equivalent to 75% of the intergrid size with different TTL values (2-5).	137
Table 6.5 Percentage of Discovered Services (SRH) for generated requests equivalent to 100% of the intergrid size with different TTL values (2- 6)	138
Table 6.6 Average Response Time (RT) for generated requests equivalent to 25% of the intergrid size with different TTL values (2-5).....	141
Table 6.7 Average Response Time (RT) for generated requests equivalent to 50% of the intergrid size with different TTL values (2-5).....	142
Table 6.8 Average Response Time (RT) for generated requests equivalent to 75% of the intergrid size with different TTL values (2-5).....	143
Table 6.9 Average Response Time (RT) for generated requests equivalent to 100% of the intergrid size with different TTL values (2-5).....	144
Table 6.10 Average Hops (AH) for generated requests equivalent to 25% of the intergrid size with different TTL values (2-5)	146
Table 6.11 Average Hops (AH) for generated requests equivalent to 50% of the intergrid size with different TTL values (2-5)	147

Table 6.12 Average Hops (AH) for generated requests equivalent to 75% of the
intergrid size with different TTL values (2-5)148

Table 6.13 Average Hops (AH) for generated requests equivalent to 75% of the
intergrid size with different TTL values (2-5)149

Table 6.14 Performance of the proposed RD system in node fault condition.....151

Table 6.15 A comparison between the semantic super peer/RD and the super-
peer model.....152

LIST OF ABBREVIATIONS

CERN	The European Organization for Nuclear Research
EGEE	Enabling Grids for E-science
GLUE	Grid Laboratory Uniform Environment
LDAP	Lightweight Directory Access Protocol
MDS	Monitoring and Discovery Service
OGSA	Open Grid Services Architecture
OIL	Ontology Inference Layer
OWL	Ontology Web Language
P2P	Peer-to-Peer
RDF	Resource Description Framework
R-GMA	Relational Grid Monitoring Architecture
SAWSDL	Semantic Annotations for Web service Description Language
SDM	Semantic Description Manager
SGMP	Service Grid Metadata Provider
SPARQL	SPARQL Protocol and RDF Query Language
TTL	Time to live
VO	Virtual Organizations
WSDL	Web Service Description Language

CHAPTER 1

INTRODUCTION

1.1 Introduction

The aim of the present work is to highlight the current use of semantic technology in grid technology more specifically on the resource discovery part, and to develop a new semantic-based, scalable, decentralized and dynamic resource discovery framework in order to meet the current grid requirements that are inherited from the deployment of the intergrid systems.

To introduce into the work, this chapter describes the motivation for which the work is conducted, and the objectives that need to be achieved. The chapter thereafter, identifies the relevant research questions which should be answered and the methodology that is followed. Lastly, the chapter concludes with the contributions and the structure of the thesis.

1.2 Motivation

The last few years have seen a convergence between Internet and distributed systems. This has brought an emergence of a new generation of distributed systems known as Internet-based distributed systems such as grid computing (Berman et al. 2003) , peer-to-peer (P2P) computing (Schoder et al. 2005), and most recently cloud computing (Buyya et al. 2009). Grids enable sharing, exchange, discovery, selection, and aggregation of geographically/Internet-wide distributed heterogeneous resources such as computers, databases, visualization devices, and scientific instruments in order to achieve a common goal (Foster and Kesselman 2003) and (Asadzadeh et al. 2005).

A very basic and first step in sharing resources over grids is the detection of suitable resource for a given task/application which is commonly known as *Resource Discovery* (RD). This process is very important as resource reservation and task scheduling are based on it. RD process entails *description of the resource through its properties, registration/indexing of the described resource in common registry(s), and discovering the registered resources that match with resource request specifications*. These steps correspond to the main components of the RD system, which are *Description, Registration and Discovery* (which is composed of *search and selection*). Eventually, the performance of the grid RD system depends on how these components are modeled. For example, having an expressive resource description makes the matching process between resource requests and advertised resources easier, and hence enhances the precision.

In fact, the ultimate aim of the grid RD research is to provide a system that allows the full use of the resources which in turn fulfills the actual aim of the grid technology (Trunfio et al. 2007) and (Mastroianni et al. 2008). However, grids are normally associated with some complexities such as resources and users are distributed across different locations; resources are heterogeneous in their platforms; status of the resources is dynamic (resources can join or leave the system without any prior notice); and grids are often distributed across security domains with a large number of resources involved. Moreover, in the most recent years, the grid scope has been extended from organizational level to multi organizational and from country to cross countries, producing a new type of grid called *Intergrid/Global Grid* (a grid of grids) (Assuncao et al. 2008) with a large number of resource and service types, and multi middlewares. These complexities pose a challenge to the development of an efficient RD system to discover the resources and services. Therefore, the aim of using fully the resources on global grids creates some requirements that should be fulfilled by any developed RD. These requirements include *high searchability* in order to retrieve the relevant and precise resources and services, and *high performance* in order to make the RD system sustainable with the scale of the global grid.

The first requirement is related to the functional quality of the RD system. The RD system should be able to discover all relevant resources/services (recall) and

present only the relevant resources/services (precision). In other words, the RD should have interoperability to overcome the resource and service information heterogeneity. This may be achieved by using the semantic technology in the description, and matchmaking of the resources/services and their requests.

The second requirement is related to computational performance of the RD system which is very vital in large scale environment such as intergrid. The computational performance concerns about reducing the processing time of the discovery process, while guaranteeing the scale of the system. This requirement can be broken further into sub-requirements which are *scalability*, *decentralization*, and *dynamism*. Respectively, the RD system should perform as it supposed to, regardless of the quantitative scale of the resources and the users that use the resources, should be independent from any global control to avoid any point of failure, and should support the intermittent availability of the resources (Padmanabhan 2006).

Currently, there is a wealth of work on grid RD (e.g. Globus¹, Condor², (Lamnitchi 2003), (Mastroianni et al. 2005), and (Shen 2009)) which can be classified into two classes based on the component description of the models, which are *keyword-based* RD systems and *semantic-based* RD systems. Keyword-based system uses syntactic information and data models such as directories (Tuttle et al. 2004) and special databases to describe and discover the resources and services. Unfortunately, syntactic information and data models are not efficient in describing resources at intergrid level. This is because resources and services are initially described by using multi information services that belong to different grid middlewares. As a matter of fact, much of the efforts in keyword-based RD systems have been focused on achieving the *high performance* requirement; starting from introducing centralized registration models such as Globus MDS-1 (Fitzgerald et al. 1997), R-GMA³ (Cooke et al. 2003) and Hawakeye (Zanikolas and Sakellariou 2005); then followed by hierarchical registration models (Steven 2001), (Schopf et al. 2006) and (Ruay-Shiung and Min-Shuo 2010), and lastly peer-to-peer (P2P)

¹ <http://www.globus.org/>.

² <http://www.cs.wisc.edu/condor/>.

³ Relational Grid Monitoring Architecture: <http://www.r-gma.org/index.html>

registration models (Trunfio et al. 2007), (Marzolla et al. 2007), (Shen 2009) and (Brocco et al. 2010). Keyword-based RD systems that are based on P2P registration models have achieved high performance compared to the centralized and hierarchical models, but we cannot go far as to say that they have achieved full scalability. Moreover, their use of syntactic description, especially at the intergrid level, prevents them from fulfilling the *high searchability* requirement.

Semantic-based RD systems, on the other hand, use semantic information and data models (ontology and ontology languages) (Chandrasekaran et al. 1999) to describe and discover the resources and services. Although, there is a considerable amount of work on semantic-based RD systems (e.g. (Ludwig and Reyhani 2005), (Said and Kojima 2009)), most of the existing approaches fail to achieve *high searchability*. This is due to the lack of a proper use of semantic description mechanism as the semantic technology is initially imported from the semantic web (Berners-Lee et al. 2001). To the best of our knowledge, the main obstacle that leads to the continuous existence of this issue is the ad hoc research nature of these semantic-based RD studies (different research communities doing the same thing by different ways). As the result, there has been no systematic research trend that allows these studies to benefit from each other in terms of lesson learnt, such as the case of keyword-based RD studies (Zanikolas and Sakellariou 2005) and (Mastroianni et al. 2008). Therefore, the challenge with semantic-based RD studies is not only to achieve *high searchability*, but also to have a review or survey study to compare and evaluate them. Regarding the second requirement (*high performance*), most of the semantic-based RD studies (Ludwig and Reyhani 2006), (Said and Kojima 2009) and (Xing et al. 2010) do not address this issue as they normally rely on the registration models of the keyword-based RD systems, which some of them may have been associated with the lack of *high performance*, initially.

1.3 Objectives

Based on the importance of the RD aspect to grid technology and the issues associated with the current RD systems, this thesis aspires at contributing to the development of

an efficient grid RD system that is able to fulfill the identified requirements, and at providing scientific progress beyond the state-of-art.

In contrast to other previous works, we address the RD problem from two perspectives. First, we provide a taxonomical model to classify and discuss the current efforts on developing semantic-based RD systems. This is to highlight the research gap with regard to the latest requirements of the grid technology (intergrid), and the potential of this field. Second, we design a novel RD framework that uses the semantic technology in a way that is useful for the grid technology, and tailor the system to meet the intergrid requirements. For this, we introduce a semantic description model, semantic distributed registration model, and an agent-based optimized search and selection algorithm. The description model is grounded on abstracting the information of resources and services of intergrid. Therein, we refine the intergrid system architecture by treating each small grid within the intergrid as service (including the grid applications) that provides some functionalities. This is to reduce the amount of unnecessary information during the discovery and to prevent redundancy of the application development, and to ensure the anatomy of each small grid within the intergrid system. The model introduces ontology as an information model that can formally represent the services (the abstracted resources and services) and their relations. This is to create a meaningful naming system for the services, which will enable interoperability among the small grids. To reduce the user interaction with the system in formulating service request, the model introduces a goal-based request formulation mechanism, which is based on extracting the relations between services on the service ontology in a way that formally defines which service needs what service. Our registration model organizes the service provider nodes into set of classes which are based on the semantic relation between the services that they provide. Each class has a head in which the metadata of the class member's services is indexed. This is to ensure that our RD system can meet the requirement for high performance. To further automate the search and selection of the services, we define two kinds of agents to perform the discovery process. In this case, the first agent formulates the service request, searches for the services on the distributed registries and conducts the semantic matchmaking with the second agent that holds the capabilities of the service provider.

The overall aim of this work can be summarized in number of specific goals as follows:

- To present a taxonomy for semantic-base grid RD studies and to provide a survey and qualitative comparison of existing studies, which will be used to identify the pros and cons of these systems and to draw the future direction of this field.
- To propose a semantic description component model for intergrid RD system that is able to provide high searchability on the intergrid platforms.
- To propose a semantic, scalable, decentralized and dynamic RD system registration model that is able to achieve high performance with an unprecedented scale on the intergrid platforms.
- To propose an intelligent discovery model for the semantic RD system that will provide a high abstraction in terms of end user interaction with the system.

1.4 Research Questions

Based on the aforementioned issues that motivate us to conduct this work and the identified objectives, our work is centered on the following questions:

- What is the stage of convergence between the semantic technology and the grid technology, how this convergence impacts the grid RD systems, and has this convergence been successful at enabling the grid technology to meet its recent requirements?
- How can the semantic technology be properly implemented to describe the grid resources, services and applications?

Semantic technology has been a very promising tool in enabling the web service technology that is based on service oriented architecture(SOA) (Erl 2005) and (Michael et al. 2007). In particular, the semantic technology is used on the discovery and composition of the services. However, the applications and the focus of web services are different from the case of grid technology. As the

former is dominated with business applications, while the latter is dominated with scientific applications. Therefore, there is a need to find the ways and means of using semantic technology on the grid RD in a flexible manner. In particular, what are the RD components that should involve the semantic technology and to what extent this involvement should be?

- How the P2P network topology can be adapted to the intergrid node topology?

P2P network is a resource sharing environment in a decentralized and dynamic manner, which gives the ability to scale the system. The resources in P2P network are limited to normal files. There are several scenarios to arrange the network for the discovery of the files (e.g. structured and unstructured). Grid technology seeks to have the scale, but with different kind of resources and management from the P2P. Therefore, the thesis will find out what is the appropriate method to transfer the scalability from the P2P to the grid technology by examining the arrangement scenarios of the P2P network in their suitability to intergrid level systems.

- How can intelligent agents be adopted to the grid application development and deployment process?

Intelligent agents have some useful characteristics in dealing with complex and dynamic environments, and to a certain extent is able act on behalf of human (Perez et al. 2009). The grid technology, on the other hand, also believes in abstraction and virtualization. To that end, how we find the right places on the RD components where the abstraction is really need, and can how the intelligent agents be involved in these places and provide the needed abstraction?

- Does the adoption of three technologies (semantic technology, P2P and intelligent agents) able to effectively improve the quality of service of the RD system on the grid technology? And will this enable the grid technology to meet its current requirements, which has arisen mainly by the deployments of the intergrid level systems?

1.5 Methodology

In order to address the identified research questions in a proper manner and provide a suitable solution to each one, we divide the work into four main parts. The first part presents a survey and comparative study on the current semantic-based RD studies. Namely, we classify the semantic technology models based on their expressiveness capabilities and review them accordingly; propose a taxonomy to the semantic-based RD systems based on the qualitative implementation of semantic technology models; discuss and evaluate these studies in terms of their accomplishments of the identified RD requirements (interoperability, scalability and so on); and finally discuss the future direction of semantic-based RD systems with regard to the emerging grids and grid related technologies.

The second part addresses the resource and service metadata representation by proposing a new semantic description model. Initially, we identified some of the facts in the current grid technology that would be vital for improving the design of a semantic-based RD system for intergrid system and refine the intergrid system in such a way that makes full use of the resources and services when the semantic technology is applied. The refinement of the intergrid is based merely on the latest standard grid system requirements. We then introduce the use of common ontology to represent formally the intergrid components. More importantly, we define some set of definitions that formalize the essential requirements and guidelines that can be followed to build this ontology, and for selecting the information manipulation tools. To address the issue of resource/service request abstraction, we introduce a semantic query formulation by treating every application as a goal, which can be formally described and made reusable. Finally, our model is evaluated qualitatively by examining how it meets interoperability feature.

In the third part, we address the resource/service registration and discovery issues jointly by proposing a new semantic registration and discovery model. The model integrates super-peer architecture, ontology and intelligent agent. Super-peer architecture is used to ensure distribution of the registry into sub registries, ontology is used to manage the distribution of the registries, and intelligent agent is used to deal

with the dynamism of the service grid provider nodes' status in their respective registries, and to abstract the discovery process from the end user.

Lastly, we address the performance examination of the proposed RD framework by conducting extensive simulation and analysis. PeerfactSim.KOM simulator is used to simulate the intergrid environment with the application of the framework. The evaluation of the system is based on some common performance metrics found in the literature. This includes the percentage of the discovered services in a given goal request, and the response time for the service request to be answered. These metrics are calculated in different settings of the nodes and service requests. We analyze the results by highlighting the causes of the effects of the different settings on the results.

1.6 Our Contributions

In this work, we provide some insights on semantic-based RD systems and present the design, implementation and evaluation of an effective RD framework that enables resource sharing in the global grids. Our framework attempts to meet the recent grid technology requirements, which we have identified above. All in all, our primary contributions can be summarized as follows:

- An outlined new key taxonomy for the semantic-based grid RD studies
- A detailed discussion and analysis of the semantic-based RD studies, how they meet the current grid requirements, and what should be the future focus in this field.
- A proposed model of semantic description component for the RD system that abstracts the resources and services information, and adds semantics to the abstracted information. Hence, improving the representation of resources and services information, and provides interoperability.
- A proposed model for evaluating semantic registration architecture that organizes the service providers based on their semantic relations. Thus, reducing the search spaces and improving the scalability.

- A fault tolerance and load balancing algorithm for the registration model, which provides dynamism to the system.
- An agent-based discovery algorithm that automates the search and selection procedures, hence reducing the end user interaction with the system.

1.7 Organization of the Thesis

After having explained the motivation, objectives, research questions and contributions of the work, the rest of the thesis is organized as follows:

Chapter 2 introduces the basic ideas about grid technology and presents the related technologies that are mainly used in designing RD systems. The chapter then discusses the RD problem in grid technology, and identifies the key requirements that need to be fulfilled. The chapter thereafter examines the existing keyword-based RD systems that have been developed with some of the grid middlewares and other research oriented studies. Finally, the chapter highlights the issues of these systems and studies.

Chapter 3 discusses the semantic technology and its use in the grid technology. More importantly, on the RD part, and provides a taxonomy for the current studies that involve the use of semantic technology. The chapter then presents a deep analysis on these studies with regard to the current grid requirements. The chapter finally discusses the future of semantic technology coupled with RD systems, for future grids and clouds.

Chapter 4 presents a model of the new semantic description component for the RD system, which includes identification of the key components of the model, and explanation of the building block of the model. The chapter, thereafter, demonstrates the process of constructing the resources and services metadata, and the formulation of resource/service request based on the new model. Finally, the chapter provides a qualitative analysis to show how the model meets interoperability feature and fits the intergrid system.

Chapter 5 presents the semantic registration and discovery model for the RD system. Initially, the chapter highlights the main components that form the model, which are ontology and intelligent agents. The chapter thereafter focuses on the depiction of the model, which contains the registry architecture, fault tolerance and load balancing strategy, and the discovery algorithm. The chapter, finally, illustrates the complexity of the new RD system and how the system meets the identified intergrid RD requirements, qualitatively.

Chapter 6 presents a quantitative evaluation of the proposed RD system. This includes an extensive simulation of the proposed system and a comparative simulation study for the system against related studies. For this, the chapter identifies the performance metrics for the evaluations, and the experimental setups. The chapter, later, discusses the results and findings of the experiments.

Chapter 7 concludes the work by summarizing the main contributions and findings of the study, the limitations of the study and some possibilities for future research and development. The appendices (A and B) provide additional information about the experimental settings, data and the simulation output. It should be noted that portions of work presented in this thesis have been partially or completely derived from the set of publications, which are listed in appendix C.

CHAPTER 2

BACKGROUND KNOWLEDGE

2.1 Introduction

This chapter presents some important literatures for understanding the grid RD problem, the related technologies that have been used to build solutions and the current RD solutions. The chapter starts with an introduction on grid technology, which includes grid requirements, architecture, types and middleware. The chapter then provides a discussion on the related technologies that are partially used by some prior works in RD solutions. This includes P2P networks and intelligent agent. The chapter thereafter discusses the RD system components, issues and requirements that need to be met by any developed RD system. Next, the chapter explains the current grid RD systems, which include the middleware provided RD systems and some of the research oriented RD studies.

On the whole, the main contributions of this chapter are as follows:

- An extensive literature about grid technology and identification of its current requirements.
- A deep insight into the grid RD problem and identification of the characteristics that need to be considered in response to the latest advancement in grid technology.
- An examination of some of the current solutions and identification of those that can provide a partial solution to fulfil the identified RD characteristics.

2.1 Grid Computing

The idea of grid computing is initially borrowed from the Power Grid (PG). PG is an infrastructure that provides electric power to satisfy the power needs of our devices. Usually, when we plug these devices, we really do not concern with neither how the electricity is produced nor how it is delivered, rather we only use the power. This concept has been used recently to describe a new type of distributed computing infrastructure. In this case, users can connect heterogeneous devices to a computing grid, and then access computing and storage power and services provided by the heterogeneous sources. The connection and access processes are transparent to the user in a similar way as the usage of the PG.

Foster and Kesselman define the grid as *“a hardware and software infrastructure that provides dependable, consistent, pervasive and inexpensive access to high-end computational capabilities, allowing coordinated resource sharing and problem solving in dynamic, multi-institutional Virtual Organizations (VOs)”* (Foster and Kesselman 2003). This definition depicts the main characteristics of a grid system, which are coordinated resource sharing and problem solving in a dynamic, multi-institutional virtual organizations manner. Sharing here is not only primarily file exchange, rather a direct access to computers, software, data, and other resources, as is required by a range of collaborative problem-solving and resource brokering strategies, which are emerging in the industry, science, and engineering. The sharing also has to be highly controlled by resource providers and consumers. The control scenario should define clearly and carefully what is shared, who is allowed to share, and the conditions under which sharing occur. Virtual organization (VO) is a set of individuals and/or institutions that meet the defined sharing rules. The works of (Foster et al. 2001) and (Foster and Kesselman 2003) present four examples for a simple understanding of the concept of VO, which are:

- “A company needing to reach a decision on the placement of a new factory invokes a sophisticated financial forecasting model from an application service provider (ASP), providing it with access to appropriate proprietary historical data from a corporate database on storage systems operated by a storage service provider. During the decision making meeting, what-if scenarios are run

collaboratively and interactively, even though the division heads participating in the decision are located in different cities. The ASP itself contracts with an on-demand cycle provider for additional "oomph" during particularly demanding scenarios, requiring of course that cycles meet desired security and performance requirements".

- "Thousands of physicists at hundreds of laboratories and universities world-wide come together to design, create, operate, and analyze the products of a major detector at CERN, the European high-energy physics laboratory. During the analysis phase, they pool their computing, storage, and networking resources to create a "Data Grid" (Chervenak et al. 2000) capable of analyzing petabytes of data".
- "A large-scale Internet game consists of many virtual world, each with its own physical laws and consequences. Each world may have a large number of inhabitants that interact with one other and move from one world to another. Each virtual world may expand in an on-demand basis to accommodate population growth, new simulation technology to model the physical laws of the world will need to be added, and simulations need to be coupled to determine what happens when worlds collide".
- "A biologist wants to understand how a change in neuron synapse response induced by a drug impacts the performance of specific brain functions. To answer this question, he needs to perform low-level chemical simulations of the synapse and then map this information upward in the structural hierarchy of the brain. This analysis requires mapping simulation across many different databases, each containing information about different levels of the biological system".

It is obvious that, these examples are different from one other in many aspects such as the number and type of participants, the nature of activities, the period and scale of the interaction, and the resources being shared. At the same time, the examples share some commonalities. For example, in each case, a number of mutually distrustful participants with varying degrees of prior relationship (perhaps none at all) want to share resources for the purpose of carrying out some tasks, and

yet again, sharing here is not a simple file exchange, rather, a direct access to remote resources such as software , data, computers, and so on. This allows each case to be considered as a VO. In order to make the resource sharing a reality, there are several technical requirements that need to be considered, which will be discussed next.

2.1.1 Grid Requirements

From the above examples, we can observe that the main components that lead to a grid are the *remote resources* and *participants*. The only connection means between the two components is the Internet network. This raises a deep consideration to some requirements that related, in one hand, to the Internet network environment, and on other hand, to end user interaction with resources. A set of such requirements is defined by (Tarricone and Esposito 2004). The *Internet network environment requirements* are:

- i. *Fault tolerance*: robustness with respect to failure of network connections, machines, software components, and so on should be addressed.
- ii. *Security*: grid users must be recognizable and access to resources must be traced and controlled as the Internet is intrinsically insecure and decentralized.
- iii. *Dynamism*: grid must adapt its behavior in agreement with the Internet environment conditions, which is the fact that, resources are dynamically added or removed, and their status (load, traffic, and so on) are variable.
- iv. *Scalability*: grids performance must not be affected by the expected increase in the number of resources and users when they are operative.
- v. *Heterogeneity*: grids must define uniform and standard ways of interaction with their heterogeneous resources (network, platforms, operating systems, electronic devices, and software tools are provided by different vendors and use different architectures and paradigms), which in turn hides the heterogeneity.

- vi. *Autonomy*: grids must federate their resources as they belong to various organizations, and allow these organizations to establish and implement their own policy regarding security, scheduling, and so on.

Meanwhile, the user interaction requirements are:

- a. *Transparency*: users must access the dispersed resources while perceiving them as a whole. Location and access to a resource must be straightforward, for both local and remote resources.
- b. *Uniformity*: the interaction with a grid must happen via a uniform interface, possibly the Web browser.
- c. *Homogeneity*: grids must mask to end users their underlying heterogeneity, allowing the access to each resource regardless of its peculiar characteristics.

The above-identified requirements can be fulfilled by having some dedicated software, which will drive the distributed resources at bottom level and allow user-friendly interaction at the upper level. Consequently, the grid is composed of three kinds of entities: resources, grid software that hides the complexity of the Internet environment; and tools for the interaction of end users with the grid.

2.1.2 Grid Architecture

Baker et al. define a three-level architecture which corresponds to the three entities that the grid is composed of (Baker et al. 2000). The architecture includes *fabric level*, *middleware level*, and *application level* (see Figure 2.1). Respectively, fabric level includes everything that will be shared. This include all the distributed resources which can be *physical*, such as hardware (CPU, memory, electronic devices, network) and software (application components, databases) entities, or *logical* (clusters, distributed pools). Middleware level includes the software responsible for mediating between the resources and their higher level managers in order to hide from grid end users and application developers the complexity of the fabric level. The middleware operates on grid resources and the local managers (i.e., single domain schedulers,

allocators, and load balancers) to offer core grid services to distributed applications. Middleware level contains also the basic elements needed to develop grid-enabled applications. Application level includes both high-level services that allow software developers to implement grid-aware applications and Web tools to permit end users to work with the grid by submitting jobs, collecting and analyzing results, and cooperating with remote colleagues.

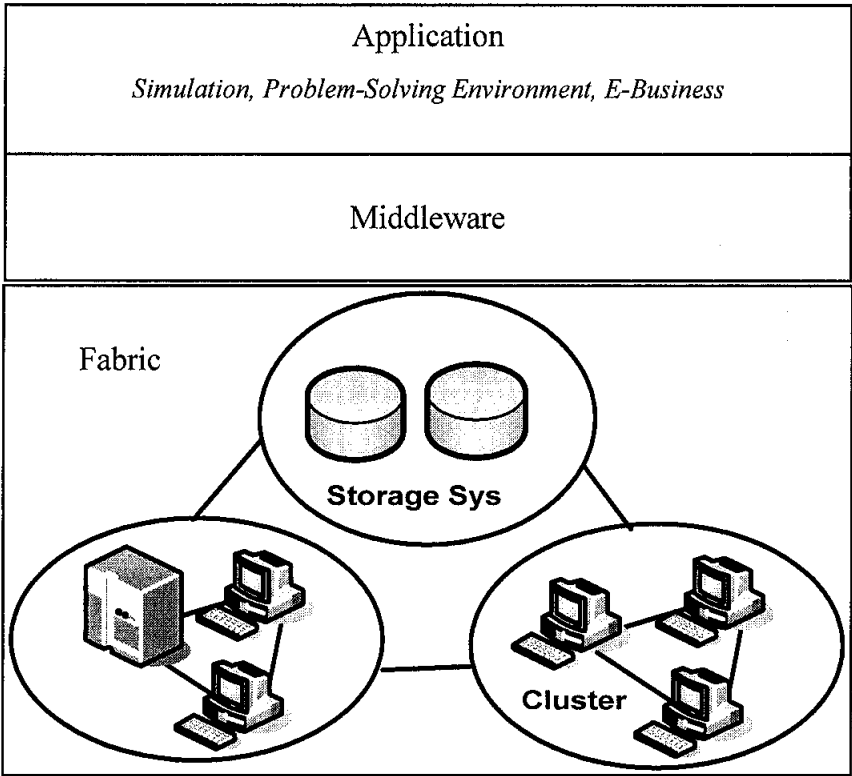


Figure 2.1 The grid architecture

2.1.3 Grid Middleware

As we have mentioned, the middleware level is supposed to mediate between the resources at bottom and applications the top. This is because resources are owned by different organizations which are initially geographically distributed. Each resource owner has its own policies with regard to security, resource allocation, platform maintenance, and so on. Therefore, the interaction between users and these resources

can take place only if there are some basic services, which are able to take out mismatches among different machines, security and scheduling policies, operating systems, platforms, file systems, and so on. Such basic services are called middleware. Middleware provides services such as discovery of new resources and reporting changes in existing ones, matching the requirements of user jobs with the characteristics of existing and available resources, verifying security rights, etc.

Currently, there are several grid middlewares which include Globus⁴, Condor⁵, gLite⁶, Legion⁷ and Unicore⁸. These middlewares differ from one another in terms of their services, which are focused on particular resources. For example, Globus and Condor focus on computing resources (e.g. CPU, Memory) whereas Unicore focuses on data resources (e.g. databases, file system) as well as the computing resources. Despite the different focus of these resources, the middleware generally provides the following basic services:

2.1.3.1 Security

Security provides resource owners the ability to define their authorization policies to monitor their resources access. These policies include what is shared, who is allowed to share, and the conditions under which sharing occur.

2.1.3.2 Information Service (IS)

IS provides a continuous monitoring of resources and status of the resources. IS contains the resource discovery system which provide two methods, registration and discovery. Registration allows the resource owners to enroll themselves as part of a resource pool. Discovery locates and accesses the resources and their attributes after their registration.

⁴ <http://www.globus.org/>

⁵ <http://www.cs.wisc.edu/condor/>

⁶ <http://glite.web.cern.ch/glite/>

⁷ <http://legion.virginia.edu/index.html>

⁸ <http://www.unicore.eu/>

2.1.3.3 Management Service (MS)

MS is responsible for scheduling and tracking the accesses to the resources in order to extract the maximum performance from them. For example, it gives to the users the ability to schedule their jobs, to track their behavior, and to analyze the status of allocated resources. RM also provides to application components the ability to change their working machines either to improve load balancing or because of a failure.

2.1.3.4 Data Management Service (DMS)

DMS provides a standardized way for accessing and transferring large amounts of data from the distributed storage systems. DMS also deals with issues that such as speed, and reliability of data transformation.

2.1.4 Grid Types

Grid systems can be classified into several types which may base on nature of emphasis, size, accessibility, and so on. Here we discuss two kinds of classifications (nature of emphasis and size) as they are related with our research context.

2.1.4.1 Grid Types Based on the Nature of Emphasis

There are six types of grids based on the nature of their emphasis: computation, data, application service, interaction, knowledge, and utility (Yeo et al. 2006) and (Ranjan et al. 2008). Respectively, *Computational grids* aggregate computational power of widely distributed computers (e.g. TeraGrid⁹ and ChinaGrid¹⁰). *Data grids* focus on a wide scale management of data in order to provide data access, integration and processing through distributed data repositories (e.g. LHCGrid¹¹ and GriPhyN¹²). *Application service grids* focus on providing remote access to applications and

⁹ <https://www.teragrid.org/>

¹⁰ <http://www.chinagrid.net/>

¹¹ <http://www.fnal.gov/gridfest/>

¹² <http://griphyn.org/>

libraries that are located on some data centers or computational grids (e.g. GridSolve/NetSolve¹³). *Interaction grids* focus on interaction and collaborative visualization between participants (e.g., AccessGrid¹⁴). *Knowledge grids* focus on knowledge applications such as acquisition, processing, management, and provide business analytic services driven by integrated data mining services (e.g. Knowledge Grid¹⁵). *Utility grids* focus on providing all the grid services IT utilities, which include computing power, data, and service to end users, on subscription basis and provides infrastructure necessary for negotiation of required quality of service, establishment and management of contracts, and allocation of resources to meet competing demands.

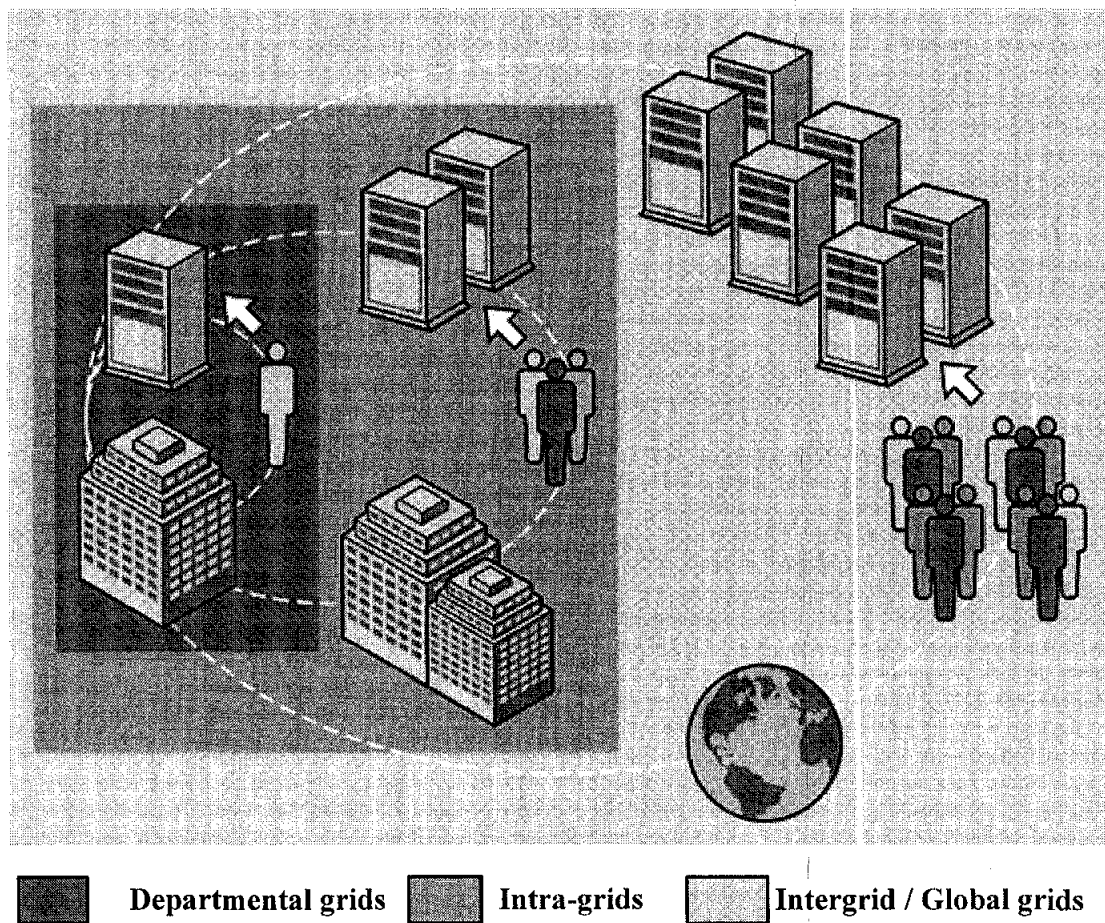


Figure 2.2 Grid types classified by size, source (<http://www.oracle.com>)

¹³ <http://icl.cs.utk.edu/netsolve/>

¹⁴ <http://www.accessgrid.org/>

¹⁵ <http://www.knowledgegrid.net/>

All these grids follow a layered design which allocates the utility grid at the top layer and the computational grid at the bottom. Each grid at a high level uses the services of the low level grids in the layered design. For example, Computational grid services are utilized by the Data grids to process huge amount of data. Therefore, Data grid is located on top of the Computational grids. Another aspect that is worth mentioning is that, higher-level grids focus more towards users and quality of service delivery, whereas lower-level grids focus heavily on infrastructure aspects.

2.1.4.2 Grid Types Based on Size

Grids also have many types in terms of the scope, which include Departmental grids/Cluster grids, Intra-grids/Campus grids, and Intergrid/Global grids. Respectively, Cluster grids are the simplest that are made up of a set of computer hosts that work together and provide a single point of access to users within a departmental boundary. Intra-grids/Campus grids enable multiple departments within a single organization to share resources. Intergrid/Global grids (in some literatures is also called multi-grid (Chao-Tung et al. 2009)) are known as a grid of grids, as they are a collection of campus grids that cross organizational boundaries to create very large virtual systems that can be accessed from anywhere in the world. Intergrids are normally associated with the use of multiple middlewares as each campus grid may use a particular middleware. For this, in the rest of this dissertation, we use the term “grid level” (when we refer to system not the grid technology) to systems that normally use one single middleware (e.g. cluster grid, campus grid) and use the term “intergrid level” to denote systems that use multiple middleware. We also use the term “intergrid”, “global grid” and “multi-grid” interchangeably. More information on the classification of grids can be found in (Kurdi et al. 2008)

2.2 Grid RD Enabled Technologies and Related Grid Components

Grid technology normally is enabled with other technologies in building its middleware components. For this, we describe two technologies that have been used in the resource discovery aspects, which are peer-to-peer computing and intelligent

agents. We also discuss a grid component that usually uses RD systems, which is known as Broker at a grid level and Meta-broker in the case of the intergrid level.

2.2.1 Peer-to-Peer Computing (P2P)

A P2P system is a resource sharing environment on which participants have an equal status and equal capabilities (peers), use appropriate information and communication systems in order to establish collaboration without having a central coordination. The P2P system differs from the other network systems such as in the appointment of client/server, where in these other networks participants are initially appointed as resource providers (Server) and consumers (Client); whereas in peer to peer, participants can act both as a client and a server. Peer-to-Peer systems have three characteristics as defined in (Schoder et al. 2005):

- **Sharing:** distributed resources and services such as information, files, and storage.
- **Decentralization:** there is no central management for organizing the network (setup aspect) or the use of resources and communication between the peers in the network (sequence aspect).
- **Autonomy:** each node (peer) in the network can autonomously determine when and to what extent it makes its resources available to other entities (peers).

The first P2P network characteristic is related with resource discovery process, which raises two issues: what resources are available in the P2P network and who is having that resource; and it is the responsibility of the P2P resource discovery mechanism to handle these arising issues. Several P2P resource discovery mechanisms have been proposed since P2P emerged. These mechanisms can be categorized into several kinds, such as unstructured and structured network. Some of these mechanisms have been implemented in the grid RD, which will be discussed in the next sections. More literature about the P2P resource discovery mechanisms can be found in (Lua et al. 2004), (Edwards 2006) and (Meshkova et al. 2008).

2.2.2 Intelligent Agent

Intelligent agent is a computer system that is situated in some environment, and that is capable of autonomous actions in this environment in order to meet its design objectives (Weiss 1999). Agents have some properties such as autonomy, intelligence, social-ability, reactivity and mobility. Jennings characterizes agents as clearly identifiable problem-solving entities that have clear boundaries and interfaces; embedded in an environment where they receive inputs that allow them to act in order to control that environment; designed to accomplish some specific goals; able to control their internal state as well as their own behavior; can demonstrate flexible problem-solving behavior in achieving their design objectives; and being both reactive and proactive (Jennings 2001). Agents have two types: mobile and static agents. Mobile agents can move within a network and act on behalf of the user or another entity. Mobile agents function independently or cooperatively to solve problems, while the static agent can function only locally and acts as a host for other (mobile) agents. More details about agent technology can be found in (Wooldridge 2006).

2.2.3 Broker and Meta-Broker

Broker is a grid component that mediates between the resource provider and consumer. It works within the grid management service. Broker gets the consumer's task and uses the RD system to retrieve the relevant resources and services to which it can assign the deposited consumers' tasks. Broker, then, schedules the tasks and monitors the progress of execution until the end to provide the results to the consumers. Broker is also known as *Superscheduler*, Condor-G (James et al. 2001) Grid-Bus Broker (Srikumar et al. 2006) are example of brokers.

Normally, brokers work at grid level by having one broker at any given system. In order to provide interoperability between the middlewares at the intergrid level, another type of broker has been introduced, which is called Meta-Broker (Kertész and Kacsuk 2007). Meta-broker sits on top of the set of brokers within the intergrid level and uses metadata to assign the consumers' tasks to these brokers. The study of

(Kertész and Kacsuk 2010) and (Ivan et al. 2010) are the most recent research advancement in meta-broker.

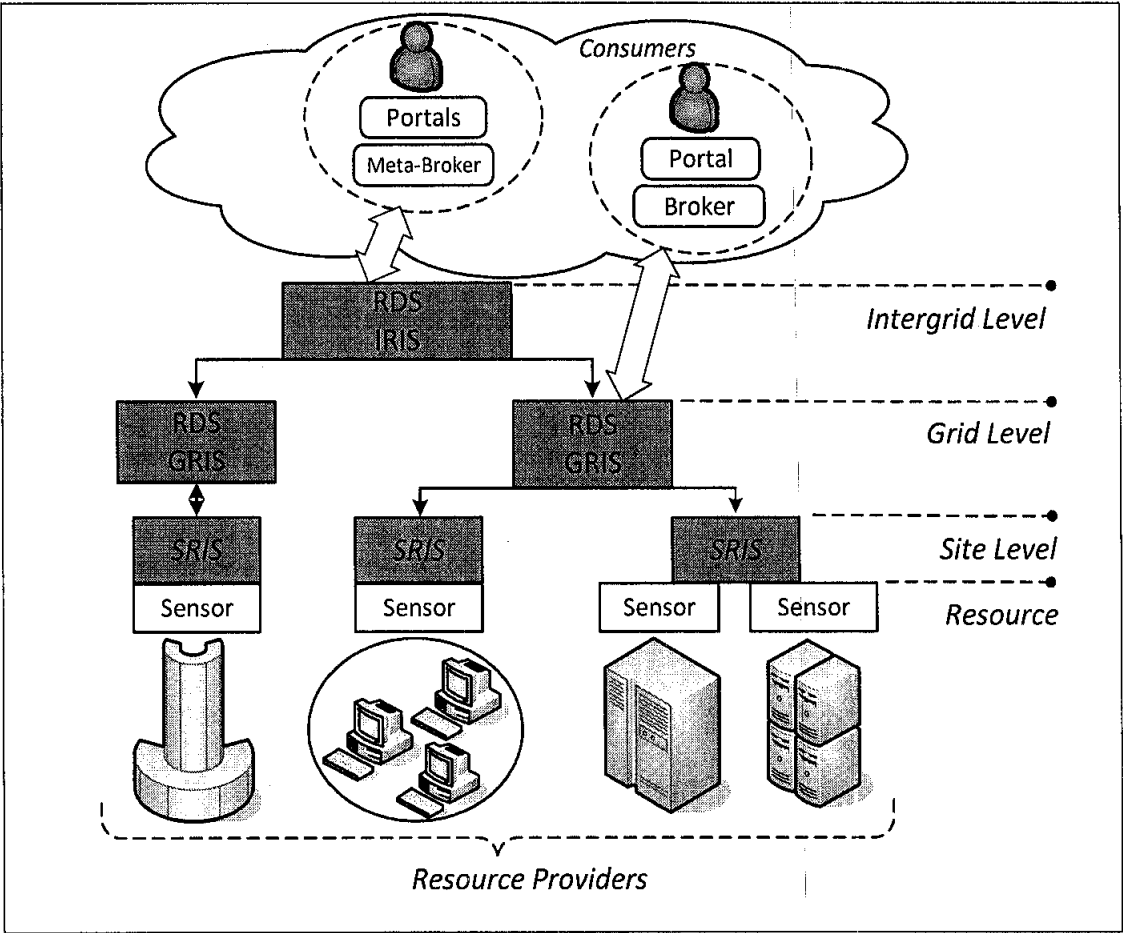


Figure 2.3 The autonomy of RD system with other grid components

2.3 Grid Resource Discovery: The Big Picture

Many definitions of RD system and its relation to other grid services can be obtained by taking a closer look at the current grid RD systems and the research studies. For example, in the course of reviewing some literatures on related works, we may come across grid information service, grid monitoring system, resource discovery, service discovery, and so on. In fact, these expressions may inherit some kind of confusion in understanding the RD problem and its requirements in grid technology. To that end, in this section we give some insights into the RD system and its relation with the other services and clarify the ambiguity among the used RD expressions. Figure 2.3

presents a generic grid system and illustrates the RD system and its related services (we ignore the other grid services such as security and management as they are out of the scope of this work), where there are four levels (*Resource, Site, Grid, Intergrid*) for the resource information flow and two sets of actors (*resource Providers and Consumers*). We elaborate each level and describe its relation with the upper and lower levels in terms of the *resource* information¹⁶ flow and RD activity.

Resource: contains the actual resources (software or hardware) with their attached sensors. Sensors measure the status of the resources and generate information about the resources. For example, CPU loads, memory size, available storage space and so on. The generated information is then sent to the upper level, which sits the grid host.

Site Resource Information System (SRIS): accommodates all resources information of the grid site coming from the sensors. Each resource is treated as a web service by using some mechanisms such as the WS-Resource Framework (WSRF) (Czajkowski 2004). SRIS provides an API so that its metadata can be quarried, registers the site resources at the grid level, and handles incoming metadata queries for the grid level.

Grid Resource Information Service (GRIS): accommodates all incoming metadata from the site level in a registry or set of registers, which depends on the registration architecture. On top of the GRIS, we have a RD system which is responsible for retrieving suitable resources among the registered resources based on the consumer's specifications. Consumers may be a grid broker or a user's portal that gets the information and informs the user to schedule his/her tasks. The collections and monitoring of the information from resource level up to GRIS is called *Grid Monitoring System* in some literature (Zanikolas and Sakellariou 2005). Meanwhile, the monitoring and discovery of the information in GRIS is called *Grid information system* in some literature (Mastroianni et al. 2008).

Intergrid Resource Information Service (IRIS): intergrid level has a higher resource information service (IRIS) that is composed of a federation of GRISs. In

¹⁶ We use the terms resource information, information of the resource(s) and metadata interchangeably.

fact, the metadata in IRIS is not only about the resources of the bottom level, but also it may be the other grid services such accounting, scheduling and brokers. The RD at this level performs as the RD at the grid level with some extra functions such as discovering the other grid services. Consumers at this level may be a meta-broker or a normal user portal that allows the user to schedule his/her task on the available resources.

Based on the above details, we can observe that the RD process involves several steps, which starts from describing the resources capabilities, registering the described resource capabilities, updating these capabilities when there is a change of their status, getting consumer resource requests, routing the query request on the registries when they are distributed, and matchmaking the query request with the available resource capabilities. All of these activities are performed by dedicated RD components, which form the RD system.

2.3.1 Grid RD Components

As we have mentioned in chapter 1, a grid RD system should contain three components, namely *Description*, *Registration* and *Discovery* (which is composed of *search* and *selection*) that correspond to the overall RD process which we have described above. Figure 2.4 illustrate these components.

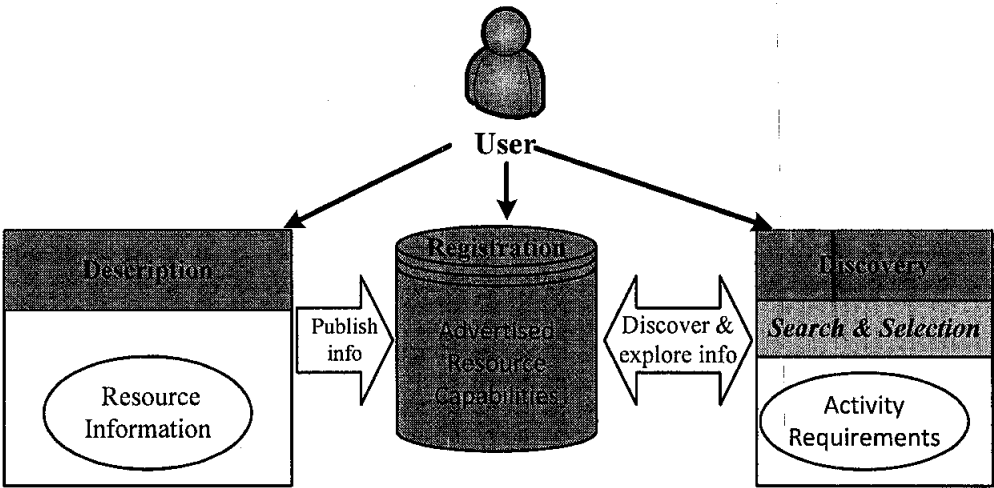


Figure 2.4 Grid RD components and their interactions

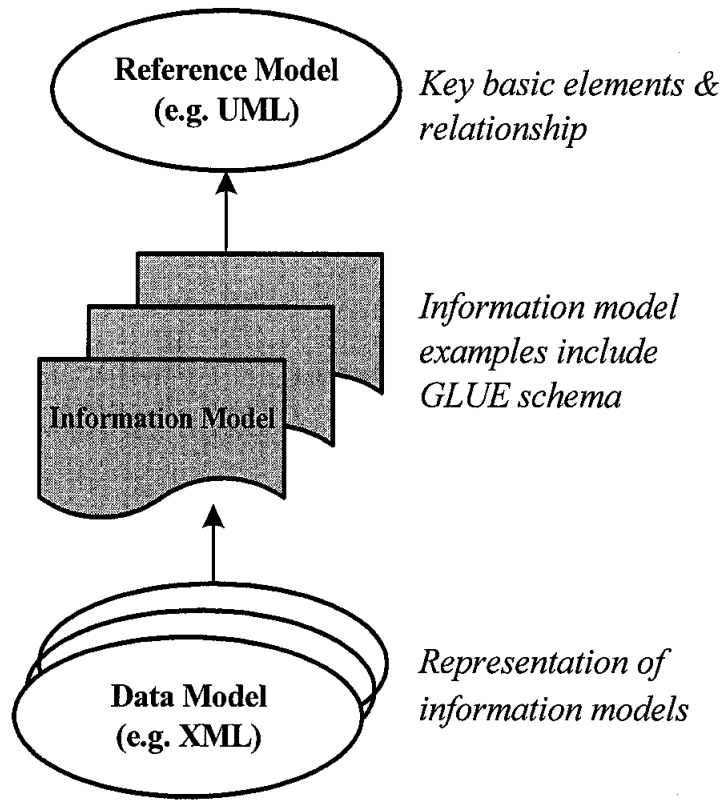


Figure 2.5 The relation between Resource modeling aspects

2.3.1.1 Description

Description refers to the abstract representation of Resource¹⁷ nature and its capabilities. This representation is done through an information system. The data at this level is called Resource Information (metadata). This metadata may be further abstracted through some algorithms that focus on showing their capabilities, and then publishing them at another component called *Registration*. The performance of RD description mainly depends on how the grid resources are modelled (Resource Modelling). In fact, Resource Modeling (RM) has three aspects: Reference Model

¹⁷ The term Resource (with capital R) includes both resource and service, whereas, resource is for hardware and software resources only, and service for middleware services (e.g. security)

(RM), Information Model (IM) and Data Model (DM). RM is a general abstract model that uses modeling notations such as UML to define some basic key grid elements and their relationships. IM is an abstraction that represents entities in a data processing environment by defining the entities, and also their properties, operations and relationships. DM is a representation of the IM in a given language. It also defines access to the IM on the wire so that the latter can be communicated. For this, a DM renders an IM according to a specific set of mechanisms for representing, organizing, and storing data. It may also define operations that can be applied to the representation, such as data retrieval and update, enumeration of entities, etc (Maciel 2008). Figure 2.5 (adopted from (Stokes et al. 2008)) depicts the relation between the Resource modeling aspects, where a DM is developed based on an IM, which is derived from a RM.

2.3.1.2 Registration

Registration is related to publishing and storing of the Resource information and how long to keep this information. It has two main aspects: registry architecture and update mechanism. The former refers to the registry location and its distance with regard to the Resource providers in the network, whereas the latter is a monitoring scenario for the status of the registered advertised Resource capabilities.

2.3.1.3 Discovery

Discovery is further composed of two subcomponents: *search* and *selection*. The former is about how to distribute a Resource request (activity requirements) from the consumer node to the registry node(s). Meanwhile, the latter is about how to evaluate the advertised capabilities that are located in the registry with regard to Resource request.

2.4 Grid RD Requirements

Having a clear understanding how grid system work and location of RD within the grid system, we now turn towards the requirements that are posed by the recent development in the grid technology, which is the intergrid system. These requirements include *high searchability and high performance* as we have mentioned in chapter 1. These requirements can be explained into more specific details as follows:

2.4.1.1 Interoperability

In intergrid level environment, each grid level participant may have its specific information service to describe its resources. This means there is heterogeneity in the information and data models that are used for the description of the resources and services. Resource consumers may not know what term to use for referring to a particular resource, and what attributers can be used to make their constraints towards wanted resources. In such situation, an RD system will not be able to discover the resources effectively unless it has interoperability. Interoperability happens with the use of scenarios that can clearly define schemes and formats of the resources and services, and request representations. For this, interoperability allows RD system to cross the resource description heterogeneity. Consequently, in order to achieve high searchability, RD system needs to have interoperability. In the rest of this thesis, we use the term “interoperability” to specify high searchability. We also break the high performance requirement into three specific requirements, which are scalability, decentralization and dynamism.

2.4.1.2 Scalability

The number of grid participants including resource and service providers, and consumers increasing continuously. Especially at the intergrid level, this number may move from thousands to millions. This causes performance degradation in discovering the resources and services with an acceptable processing time. Therefore, RD system should have scalability to discover the resources and services as efficiently as it is

supposed to, regardless of any quantitative changes in both resources/services, and the consumers that use these resources and services.

2.4.1.3 Decentralization

Grids (intergrid level and grid level) initially federate resources from multiple providers which mean, there is no common control for the entire system. This implies that RD system should work in a decentralized manner. In this case, the resources and services information (RD registration or registry) will not be put under a common control. This is to avoid the problem of bottle neck and lack of load balance, and fault tolerance features.

2.4.1.4 Dynamism

Since resource and service providers are allowed to join or leave the system without any prior notice, or even using the resources and services for their own tasks, there is a need to monitor the status of these resources and services upon their registration on the system. Therefore, RD system should be dynamic enough to track the status of the resource and service information in order to maintain reliability in the discovered resources and services.

2.5 Existing Grid RD Systems

As we have mentioned in chapter 1, there is a wealth of work on grid RD that includes the developed services within the current grid middlewares and the research oriented ones. To discuss these systems clearly, we classify these systems according to the technologies that they use in their components. Figure 2.6 illustrates the taxonomy of the existing RD systems in term of the technologies used in the components.

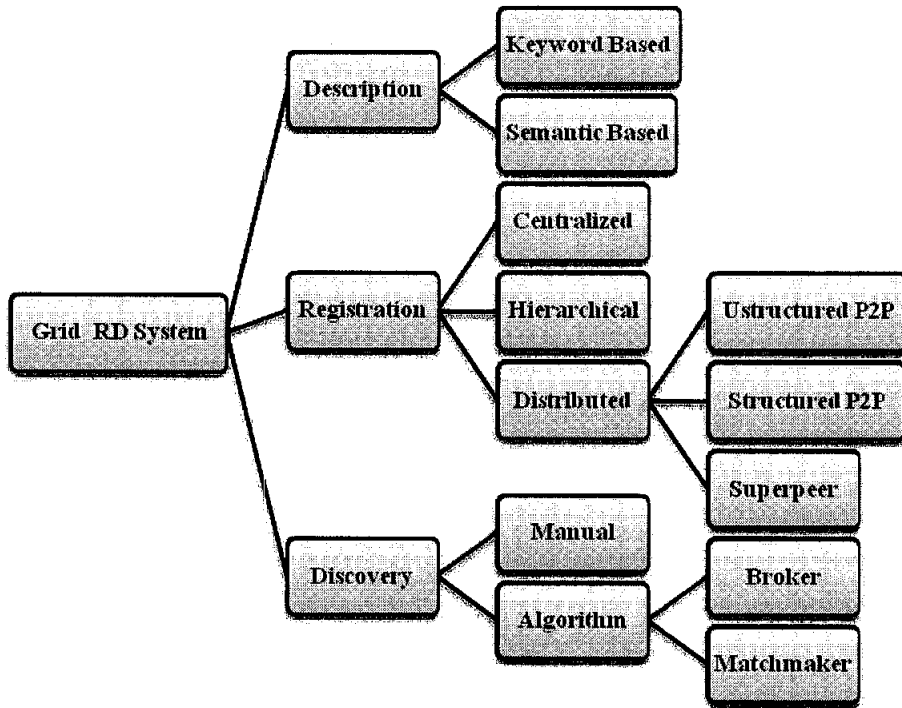


Figure 2.6 Grid RD systems taxonomy based on the technologies used

2.5.1 Existing RD Systems' Description

The description components of existing RD systems are initially divided into two categories: keyword-based description¹⁸ (non-semantic) and semantic-based description. The former uses syntactic information and data models to describe resources and service. Meanwhile, the latter uses semantic information and data models (Ontologies and Makeup languages) to describe resources and services. In the rest of this thesis, by the term “keyword description approach” we mean the syntactic information and data models, and framework that are used to describe resources or services, and use the term “keyword-based approaches/RD systems” to denote RD systems that use keyword description approaches regardless of their registration and discovery components. In this sub-section we discuss the keyword-based approaches

¹⁸ We use the term keyword based and non-semantic based interchangeably

and leave the semantic-based ones to be discussed in the next chapter, where we elaborate more on this topic.

2.5.1.1 Keyword-Based Approaches

There are three main keyword description approaches that have been used by the leading grid middlewares. These models include *Classified Advertisement language* (Solomon 2004), *Lightweight Directory Access Protocol* (LDAP) (Tuttle et al. 2004), and *Relational Grid Monitoring Architecture* (R-GMA)(Cooke et al. 2003).

Classified Advertisement language is used by Condor¹⁹ middleware. The basic representation of a resource in Classified Advertisement language is called ClassAds/advertisement. ClassAds is a set of uniquely named *expressions*. Each named expression (name, expression) is called an *attribute*. Expressions are composed of simple literals (integer, floating point, or string) and attribute references are composed with operators and functions. The attributes are organized in a tree based structure. The root of attributes is called *record*. Resource providers construct ClassAds that describes their capabilities and declares their constraints and preferences towards the jobs they are willing to run. Consumers also construct and submit ClassAds describing their jobs with their constraints and preferences with regard to execution sites. The query is then done through the evaluation of records of the resource provider and consumer. For example, if X, Y are record expressions, each of which is expected to have a "top-level" definition of the attribute *Requirements*. X 's *Requirements* attribute is evaluated in an environment in which the attribute refers *other* evaluates to Y , and Y 's *Requirements* are evaluated in an environment in which it refers *other* evaluates to X . If both *Requirements* attributes evaluate to the specific true value (not undefined, error, or a value of some non-Boolean type), the expressions A and B are said to *match* (Solomon 2004).

¹⁹ <http://www.cs.wisc.edu/condor/>

Lightweight *Directory Access Protocol* (LDAP) (Tuttle et al. 2004) is used in Globus²⁰ middleware. The basic representation of a resource in LDAP is called *entry*. An *entry* represents an object of interest in the real world, which is composed of a set of attributes, and each attribute a data type and one or more value. Entries are then organized in a tree-like structure, which is known as Directory Information Tree (DIT). DIT arranges entries based on their distinguished name (DN). A DN is a unique name that explicitly identifies a single entry. For this, DN is used as the primary key for an entry in the directory. DNs are made up of a chain of relative distinguished names (RDNs). Each RDN in a DN matches to a branch in the DIT leading from the root of the DIT to the directory entry. The common format of RDN is <attribute name>=<value>. The query process in directories, begins with the user specifying four things in the query message, which are: the starting point within a DIT, how deep within the DIT to search, what attributes an entry must have to be considered a match, and what attributes to return for matched entries. Respectively the components are formally called as: *Base*, which is a DN that defines the starting point; base object - a *Scope* that specifies how deep within the DIT to search from the base object; a *Search Filter* - a Boolean combination of attribute value assertions that identifies the criteria an entry must match in order to be returned from a search; *Attributes to Return* specifies which attributes to retrieve from entries that match the search criteria; and *Limits* specifies the time and size limits of the search (number of entries to be returned and the total time of the search).

R-GMA is implemented with gLite²¹ middleware. R-GMA is an implementation of special relational database. In this context, resources and attributes information are organized into tables, and upon that they can be inserted and queried using standard SQL constructs and views, and indices can be used. The query process in R-GMA begins with consumers using the select statement to return the relevant information about a particular resource or service.

In addition to that, traditional web services discovery framework is implemented is grid RD. For example, Pastore proposes Universal Description, Discovery and

²⁰ <http://www.globus.org/>

²¹ <http://glite.web.cern.ch/glite/>

Integration (UDDI) as RD model to Globus-based grid system. UDDI initially uses the Web Service Description Language²² (WSDL) and data model through which web services can be described. The described services are then published into a centralized registry that responds to service requests made by consumers (Pastore 2008).

Since all of these description mechanisms belong to different middlewares, initially they used different information models to represent the resources and services, which inherited some heterogeneity if a grid used two description mechanisms at the same time. To address that, a common resource information model was introduced, which is called Grid Laboratory Uniform Environment (GLUE)²³ schema. GLUE schema is an abstract modeling for grid resources and mapping to concrete schemas that can be used in grid information services. The GLUE schema defines a clear separation between the entities involved in a grid system. More precisely, it defines two main categories: *System* and *Service*. The earlier is defined as a set of connected items or devices which operate together as a functional whole. Meanwhile, the latter is an abstracted, logical view of actual software components that participate in the creation of an entity providing one or more functionalities, useful in a grid environment. The GLUE schema defines further the components of each category. For example, the system contains two main components of computational grids, which are *cluster* and *storage* systems, and the service contains computing and storage services. In addition to that, the GLUE schema defines the relations between the components. More details about the GLUE schema can be found in (Andreozzi et al. 2007). Besides that, the GLUE Schema has been mapped to several concrete data models such as the ClassAds language (Garzoglio et al. 2008) , LADP directory (Andreozzi et al. 2009 -c), Relational schema (Andreozzi et al. 2009 -a), and XML schema (Andreozzi et al. 2009 -b).

²² www.w3.org/TR/wsdl

²³ <http://forge.ogf.org/sf/projects/glue-wg>

2.5.2 Existing RD Systems' Registration

Existing RD systems that use keyword description approaches, which have been described above, use in their registration components either *centralized* or *hierarchical* architecture. However, there are other mainly research oriented RD systems that use *distributed* registration architectures. In this sub-section we discuss briefly these registration models.

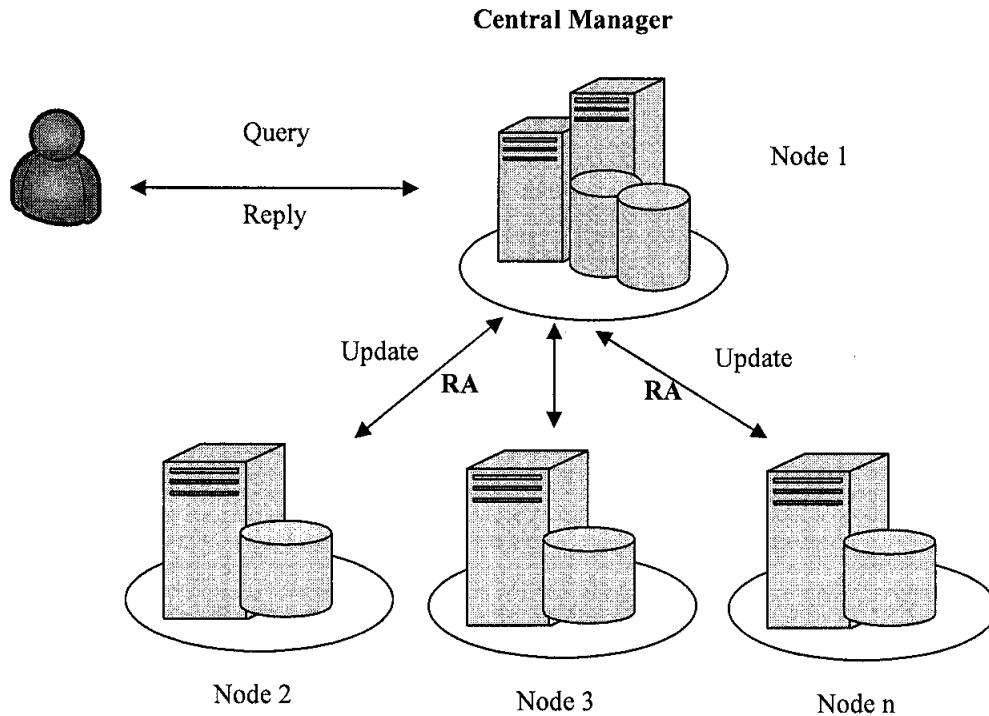


Figure 2.7 The Condor RD centralized registration

2.5.2.1 Centralized Models

In centralized RD systems, resources information is indexed under a common centralized server/node. Consumers send their resource queries to that server, and the common server/node will match the queries. Resource providers update their resource status at periodic intervals using resource update messages. The Condor system is an example of centralized registration model. Condor contains a central scheduler called Central Manager (CM) which performs the scheduling task. In this case, CM

gathers the information of the resources and receives users' requests as ClassAds. The CM then find matches between these ClassAds, and finally decides where to schedule the jobs (see Figure 2.7). Condor uses intelligent agent to represent the resource providers, which are called Resource owner Agents (RA). Each RA periodically checks the state of its resource and then constructs a ClassAds of the resource. A Condor scheduler can thus, discover new resources and update the state of existing ones, only when they advertise their state by sending ClassAds. The validity of the information contained in the ClassAds is also related to its rate of update. The Condor scheduling process includes the possibility that a resource may reject an assigned job due to changes incurred in its state since the last ad was sent.

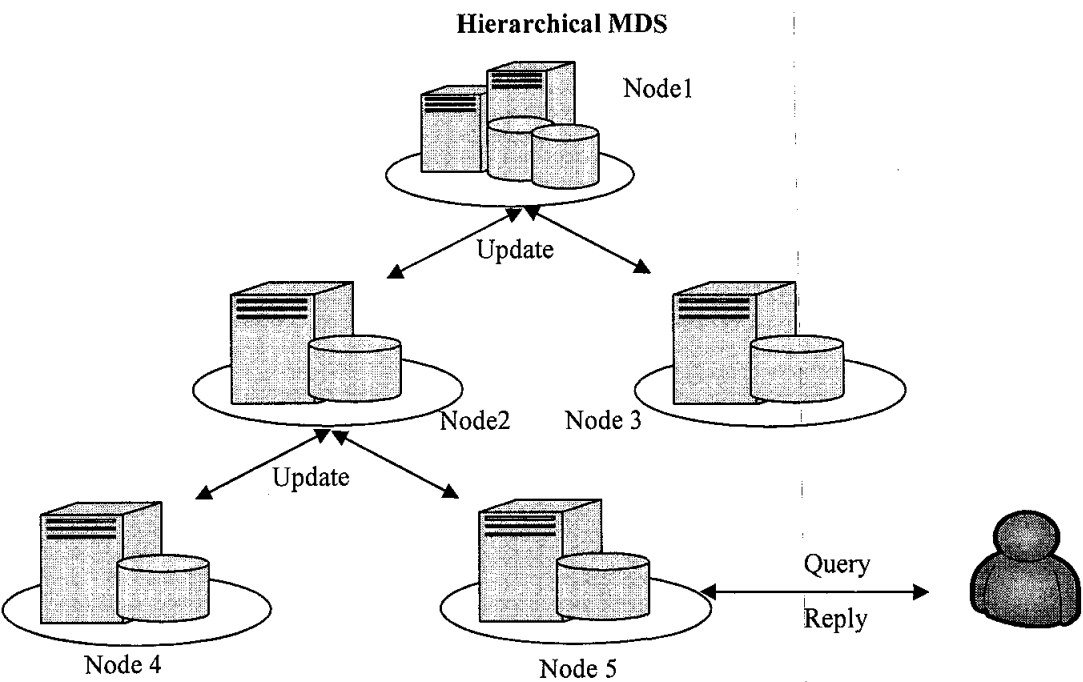


Figure 2.8 The Globus MDS hierarchical registration model

2.5.2.2 Hierarchical Models

In hierarchical RD systems, the resource information is indexed under a set of server nodes in a hierarchical manner. Each parent nodes can answer queries/requests about its child nodes. The monitoring and discovery service (MDS) of Globus implements this model. It uses two services: a configurable information provider called Grid

Resource Information Service (GRIS) and a configurable aggregate directory service called Grid Index Information Service (GIIS). A GRIS answers queries about the resources of a particular grid node. A GIIS combines the information provided by a set of GRIS services managed by a given Virtual Organization (VO). Figure 2.8 describes the hierarchical Globus MDS model, where Node 4 and Node 5 run the GRIS that connects to the GIIS hosted at Node 2. It should be noted that, Node 2 hosts both GIIS and GRIS, and updates the information about its local resources along with the child GRISs with the root GIIS service hosted at Node 1.

Both of the described models (centralized & hierarchical) have some issues with regard to the RD requirements. For example, in Condor system, the Central Manager that matches the resources with the users' tasks may be a point of the failure. In Globus MDS, the updates on GRISs at the lowest levels do not automatically propagate up to the top of the hierarchy, which means the available resource information may not be completely up-to-date. This has motivated researchers recently to focus on distributed peer-to-peer based registration models.

2.5.2.3 Distributed Models

Distributed RD registration models are mainly research oriented studies. These studies use P2P resource discovery protocols and architectures to build their registration. In fact, P2P resource discovery mechanisms can be classified into three classes which are unstructured systems, structured systems and super system. All three models are adopted on grid RD registrations.

2.5.2.4 Unstructured P2P

In unstructured P2P systems, each peer maintains a consistent number of connections to other peers, called its neighbors, by doing so a network of peers is formed. This network has no underline structure, therefore there is no information about the

location of files or resource. Systems like Gnutella²⁴ and Routing Indices (Crespo and Garcia-Molina 2002) are examples of unstructured P2P network. The discovery process is based on broadcast-like process called “flooding”. A peer looking for a resource issues a query message and broadcasts it in the network. Upon receiving a query, each peer broadcasts it to all of its neighbors except the upstream one, and sends all matching query responses to the originating peer through the reverse path. Grid studies such as (Iamnitchi and Foster 2004) and (Talia and Trunfio 2005) followed the unstructured P2P models.

2.5.2.5 Structured P2P

Structured P2P systems are introduced to enhance the resource discovery performance of unstructured systems by using distributed indexing service, which is based on hashing, and is known as Distributed Hash Table (DHT). In these systems, Peers and files are mapped through a hash function to a key space. Peers and file indices are organized in a rigid structure according to their keys, which facilitates the location of files. Examples of these systems are Chord (Ratnasamy et al. 2001), CAN (Content Addressable Network) (Rowstron and Druschel 2001) and Pastry (Rowstron and Druschel 2001). Grid RD studies such as (Bharambe et al. 2004) and (Shen 2009) followed the structured P2P models.

2.5.2.6 Super-peer

Although, structured P2P aims to improve the performance of the unstructured one, the maintenance of the tables DHTs limit their scalability of the network. Therefore, a new model known a super-peer network has been proposed (Nejdl et al. 2003) and (Yang and Garcia-Molina 2003). A super-peer is a node in a P2P that performs as a server on a set of clients and as an equal with regard to other super-peers. Together, a super-peer and its clients is called a *cluster*, and the number of nodes (clients and super-peer) is known as *cluster size*. Each super-peer keeps and maintains an index

²⁴ <http://wiki.limewire.org>

over its clients' data, which contains the resource information. The look up for resources is as follows: a client submits a query to its super-peer only. The super-peer will then submit the query to its neighbors as if it was its own query, and forwards any *response messages* it receives back to the client. Grid RD studies (Mastroianni et al. 2005) and (Puppini et al. 2005) have adopted the super-peer architecture.

2.5.3 Existing RD Systems' Discovery

Discovery involves search and selection. Search initially depends on the registration architecture as it is supposed to route a consumer resource or service request into registration component. Generally all, but the existing RD systems including middleware RDs and the research oriented ones, implement or extend one or some of the known *Packet Propagation* algorithms such as *Unicast* , *Multicast* , and *Anycast* for their searches. For example, in Condor system the search is unicast one-to-one communication that involves the CM and consumer. Meanwhile, unstructured P2P based RD systems use as multicast one-to-many communication. More details about the *Packet Propagation* algorithms related RD can be found in (Meshkova et al. 2008).

Selection is the act of deciding which resource to select from the set of resources or services that match with consumer's request. Selection in the RD systems is either done manually by the user or some algorithm. In the manual case, users use their browser and ports to select any resource from the retrieved list based on their own preferences. Some Globus users use this method. Algorithm is further divided into two, namely broking and matchmaking. In broking, grid broker such as Condor-G is used to perform the selection by using criteria based algorithms. In matchmaking case, Matchmakers are used to perform the selection, which are similar to brokers, but matchmakers do not interfere in the next step after the matchmaking, which is the interaction between matched request and selected resource, as is the case of brokers.

2.6 Assessment Summary of the Existing RD Systems

We conclude this chapter by providing an assessment on the existing RD systems with regard to the identified RD requirements, to see if they provide a partial solution to intergrid RD problem. Existing grid middleware RD that use keyword description approaches work fine at grid level but they are not efficient at intergrid level due to the information and data models heterogeneity. We understand that GLUE schema is a good effort to provide a standard information model among these information services, but at present GLUE schema is mainly focusing on the computational resources (CPU and Memories). However, intergrid level resources go beyond the basic computational resources. To sum up, keyword description approaches are associated with a lack of interoperability.

Centralized and Hierarchical are acceptable at the grid level as the number of participants is relatively small. However at the intergrid level, centralized model will be a point of failure with the addition of lack of load balancing. Hierarchical models are associated with a delay during the update from the bottom nodes to the upper level nodes. This means there is a lack of dynamism. Therefore, we can say that centralized models have no scalability and decentralization, and hierarchal models lack dynamism although they are more scalable than centralized ones. Regarding the distributed registration, thanks to the study by (Mastroianni et al. 2008) that has provided a broad simulation based comparative study on unstructured P2P based model, Hierarchical model and super-peer model. The authors concluded, based on the results, that hierarchical model is more scalable than unstructured P2P model, and super-peer model is more scalable than hierarchical. Based on these comparisons and considering other facts related to fault-tolerance, load balancing, and administrative features, super-peer is the good candidate for intergrid registration component compared to the other models, but we cannot go far as to say that it achieves a full scalability as it uses blind distribution of the resources and service requests.

CHAPTER 3

THE STATE OF THE ART IN SEMANTIC-BASED GRID RD SYSTEMS

3.1 Introduction

In the previous chapter, we have concluded that keyword description approaches that are used by the current grid middleware RD systems are not able to provide interoperability at the intergrid level. A potential candidate to provide interoperability in the RD system description is the semantic technology. This is because semantic technology was initially introduced to make the current web meaningful (Sheila et al. 2001), and has been implemented in some grid RD studies (e.g. (Pernas and Dantas 2005), (Said and Kojima 2009), and (Xing et al. 2010)). However, the lack of a proper implementation of semantic technology that is suitable with grid technology in terms of resources, services and behaviors is preventing most of these studies to achieve interoperability. In fact, the main drawback that has contributed indirectly to that is the lack of review and survey studies to review and compare these studies so that they can benefit each other from the lesson learnt. On the contrary, in the case of keyword-based RD systems, there have been several review studies (Zanikolas and Sakellariou 2005), (Trunfioa et al. 2007), (Mastroianni et al. 2008) and (Ranjan et al. 2008), who have evaluated these studies and identified the focus of future directions. As the results, keyword-based RD systems have been gradually heading towards achieving scalability and decentralization features. All in all, providing a review or survey study to compare and evaluate studies that have introduced semantic technology and drawing future research directions in this field remains a challenge (Trunfioa et al. 2007). This is equally challenging as providing interoperable RD system through the use of semantic technology.

To that end, this issue is being addressed in this chapter. Initially, we will discuss semantic technology and its use in grid technology. Then we will focus on the use of semantic technology in grid RD system components and present the current works in this perspective. In addition, we will present a detailed analysis of these systems and the future research directions in this field.

Overall, the main contributions of this chapter are as follows:

- A comprehensive review on the available semantic technologies with a focus on their information expressiveness capabilities, and classify them in that regard.
- A discussion on the use of semantic technology in grid technology with the focus on current efforts in providing grid domain ontologies and semantic description services.
- A taxonomy for semantic-based grid RD systems based on their qualitative use of semantic technology, a review of these studies with regard to the identified grid RD requirements and the future direction of this field.

The rest of this chapter is organized as follows. Section 3.2 discusses the semantic technology. The use of semantic technology in grid and semantic-based RD systems are presented in Section 3.3 In Section 3.4 we discuss the reviewed RD systems and provide a comparative summary. Section 3.5 tells the future use of semantic-based RD systems in emerging technologies. Section 3.6 presents related work in this area. Finally, section 3.7 concludes the chapter.

3.2 Methodology

We address the surveying and comparison issue on semantic-based RD studies in four parts which are: reviewing the semantic technology models, reviewing the RD studies that implement the semantic models, evaluating the reviewed studies in terms of their accomplishments of the identified RD requirements, and discussing the future of the RD system usage with emerging grids and grid related technologies.

In the first part, we review only the semantic technology models that are standard of the WC3²⁵ or have been used by any semantic-based RD study. The reason behind that is, as in the case of the standard models, it is to highlight the competencies of these models for future research use if they are not used at this time around. Meanwhile in the case of models that are used by some semantic-based RD studies and are not standard, it is to grant a deeper understanding on how these models are implemented in the grid technology.

In the second part, we review the studies that have been presented from 2005 until the present date when this thesis is written. The reviewed studies are selected based on the scientific maturity. Particularly, we mainly select studies that appear in the most reputed journals of the well known publishing houses such as Elsevier and Springer. The study review covers all aspects of the use of semantic technology in the studies.

In the third part, we discuss and evaluate the reviewed studies with regard to the RD identified requirements. The evaluation is based on a deep analysis and observations of the studies capabilities, and mapping these capabilities to the required capabilities; which then results the ability of the studies to fit into the required capabilities. This method has been followed by some of the related works in the case of keyword-based RD studies (Trunfioa et al. 2007) and (Ranjan et al. 2008).

In the last part, we discuss the future of semantic-based RD with the ongoing advancement in grid technology and the most related technology to the grid, which is the Cloud technology. The discussion is based on what the semantic-base RD systems can offer to these technologies in achieving their goals.

3.3 Semantic Technology

Semantic technology is a type of information and data models, and mechanisms that are used in resource and service description, and information integration. Information

²⁵ <http://www.w3.org/>

model in semantic technology is called *ontology* and data model is called *markup language/ontology language*. Ontology primarily is a formal, explicit specification of a shared conceptualization (Gruber 1995) and (Chandrasekaran et al. 1999). Conceptualization here is an abstract, simplified view of a domain, which identifies the relevant concepts of that domain. For this, ontology consists of *concepts* and *relationship* between these concepts. It should be noted that, establishing relationships between domain concepts allows us to understand the concept not merely by its properties, but by its presence in relation to other concepts within the ontology (Flahive et al. 2009).

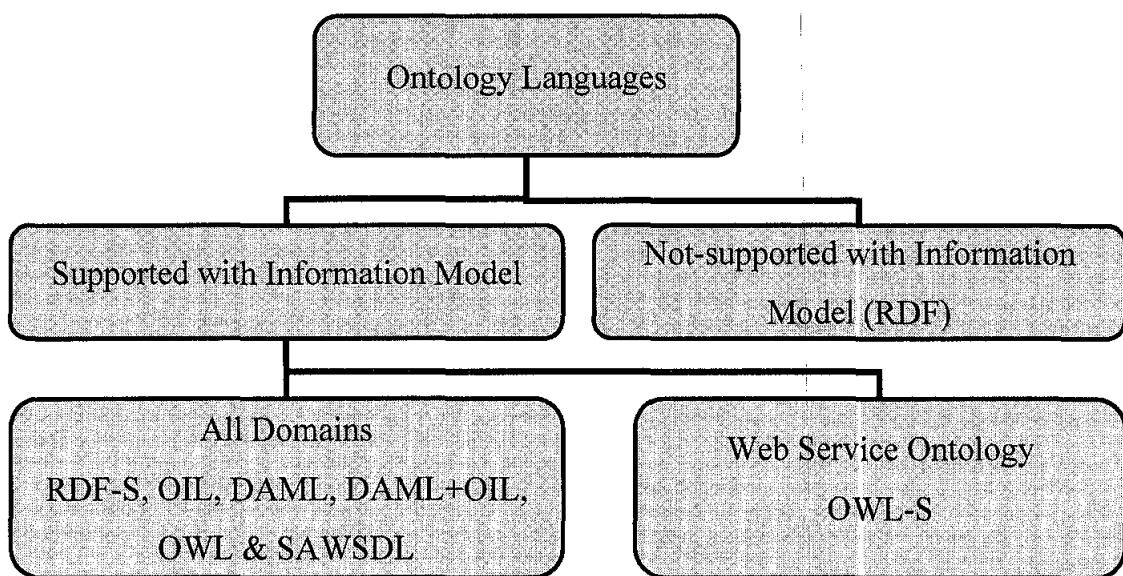


Figure 3.1 The taxonomy of ontology languages

3.3.1 Ontology Languages

Currently, there are several languages available to encode ontologies. The difference between these languages is mainly on their expressiveness capabilities to represent a given domain of interest. Therefore, they can be classified based on their expressiveness capability. Accordingly, they initially can be divided into two classes. The first class includes languages that have their basic information model for a domain description. In other words, the languages are themselves ontologies. This class can be further divided into those that provide the basic ontology components,

which are useful for constructing ontologies of any domain of interests such as biology, chemical and physics, and others that provide a basic ontology for the web services (see Figure. 3.1).

3.3.1.1 Not-supported with Information Model

Resource Description Framework²⁶ (RDF) is the only language that represents this class. RDF provides a simple way to express resources. A RDF expression is basically a collection of triples, each consisting of a *subject*, a *predicate* and an *object*. A set of such triples is called an RDF graph. Each triple corresponds to a declaration of a relationship between the things denoted by the nodes that it links. The collection of the triples in the same document is called RDF document.

3.3.1.2 Supported with Information Model

Ontology languages that are supported with information models comprise of *all domains* and *web service ontology* languages. The earlier includes Resource Description Framework Schema²⁷, Ontology Inference Layer (OIL), DARPA Agent Markup Language + Ontology Inference Layer²⁸ (DAML+OIL), Ontology Web Language²⁹ (OWL), and Semantic Annotations for WSDL³⁰ (SAWSDL). Meanwhile, the former is represented by OWL-S³¹.

- Resource Description Framework Schema³² (RDF-S)

RDF is extended with an information model that is called RDF Schema (RDFS) . RDFS provides methods that are able to describe groups of related resources and their relationships to each other. The basic elements of the schema are *classes* and *properties*. Classes describe the kinds of resources and properties characterize these

²⁶ <http://www.w3.org/RDF/>

²⁷ <http://www.w3.org/TR/rdf-schema/>

²⁸ <http://www.w3.org/TR/daml+oil-reference>

²⁹ <http://www.w3.org/TR/owl-features/>

³⁰ <http://www.w3.org/2002/ws/sawSDL/>

³¹ <http://www.w3.org/Submission/OWL-S/>

³² <http://www.w3.org/TR/rdf-schema/>

resources. RDFS class and property system is similar to the type systems of object-oriented programming languages such as Java. Therefore, resources can be defined as *instances* of one or more *classes* (`rdfs:Resource`), and classes can be organized in a hierarchical fashion so that a class may have a subclass (`rdfs:subClassOf`). Each class is often identified by RDF *Uniform Resource Identifier* (URI) and may be described using RDF properties. A property is the relation between subject resources and object resources. For example, the `rdf:type` property may be used to state that a resource is an instance of a class. Each property has *range* (`rdfs:range`) and *domain* (`rdfs:domain`) attributes. Range is used to state that the values of a property are instances of one or more classes; meanwhile, domain is to state that any resource that has a given property is an instance of one or more classes.

- **Ontology Inference Layer (OIL)**

OIL has emerged to further provide a more expressive power than RDFS. OIL is based on three components, namely, *frame-based system*, *description logic* and *web standard*. Frame-based systems have rich modeling primitives (Horrocks et al. 2000). The central modeling primitive in frame-based systems are classes (*Frames*), which have some properties called *Attributes*. These attributes have a local scope that bounds them to be applicable to the frames for which they are defined. A frame also provides a certain context for modeling one aspect of a domain. OIL is based on the idea of a class and the definition of its super-classes and attributes. Relations can also be defined as independent entities with a certain domain and range. *Description logic* provides formal semantics and efficient reasoning support. In description logic knowledge is represented as *concepts* and *roles*, and translated into mathematical format, which is used to automatically derive classification taxonomies (reasoning). *Web standard* includes XML and RDF. OIL has well defined syntax in XML. It is also defined as an extension of the RDF and its schema.

- DARPA Agent Markup Language + Ontology Inference Layer³³ (DAML+OIL)

DAML was initially proposed by the US Defense Advanced Research Projects Agency. The aim was to introduce a simple language for expressing more sophisticated RDF class definitions than permitted by RDFS. However, to improve the language standardization on the semantic web, the DAML group pooled its efforts with the Ontology Inference Layer (OIL) to produce what is known as DAML+OIL (McGuinness et al. 2002). DAML+OIL was initially built on RDF and RDFS, and these models have been extended with richer modeling primitives. DAML+OIL provides modeling primitives commonly found in frame-based languages. It has three characteristics: first, an *underlying mapping* to an expressive Description Logic (DL) that provides a well defined semantics and clear understanding of the formal properties of the languages. Thus, using DL allows DAML+OIL to be flexible in composing classes and slots to form new expressions, unlimited nesting of class elements, transitive and inverse slots, general axioms, etc. The second characteristic is *a machine-readable syntactic encoding in the languages of the web*. As RDF has gained a wide use in the metadata deployments, DAML+OIL ontologies are accessible by any agent written by RDF. Third, *a layered architecture*, avoiding the temptation to throw everything into the core language, mixed up features that cannot be reasoned over with those that can be. Therefore, the limits are clear and explicit (Bechhofer and Goble 2001).

- Ontology Web Language³⁴ (OWL)

OWL is the first ever standard language for ontologies. It is compatible with the early ontology languages that have been described above, and provides us more power to express semantics. This includes *existentially, conjunction, disjunction, and universally quantified variable*. This allows reasoners to take the advantages of these capabilities (Pulido et al. 2006). The OWL language has three sublanguages, which are designed for supporting some specific domains. First, *OWL Lite* is designed for users primarily needing a classification hierarchy and simple constraint features. Second, *OWL*

³³ <http://www.w3.org/TR/daml+oil-reference>

³⁴ <http://www.w3.org/TR/owl-features/>

DL (short for *Description Logic*) is designed for users who want the maximum expressiveness with a reasonable time-complexity. It allows efficient reasoning and inferencing. Last, *OWL Full* is meant for users who want maximum expressiveness and the syntactic freedom of RDF with no computational guarantees. For one thing, it is possible to treat a class simultaneously as a collection of individuals and as an individual in its own right. The basic elements in OWL are *classes*, *instances of classes* (individual), *subclass* and *properties*. Classes are the roots of various taxonomic trees, and they should correspond to the most basic concepts in a domain. Every individual in the OWL world is a member of the class `owl:Thing`. Subclass (`rdfs:subClassOf`) relates a more specific class to a more general class. If A is a subclass of B, then every A's instance is also an instance of B. The `rdfs:subClassOf` relation is transitive. If A is a subclass of B and B is a subclass of C, subsequently A is a subclass of C. Therefore, each user-defined class is implicitly a subclass of `owl:Thing`. A *property* is a binary relation that allows the assertion of general facts about the classes and specific facts about instances. It has two types: *datatype* (`DatatypeProperty`) and *object* (`ObjectProperty`). The former relates instances of classes and RDF literals, and XML Schema data types, meanwhile the latter relates instances of two classes. OWL provides an exceptional property which is the import of external ontology to an existing one. The `owl:imports` provides an include-style mechanism that imports another ontology, which means bringing the entire set of assertions provided by that ontology into the current ontology.

- Semantic Annotations for WSDL³⁵ (SAWSDL)

SAWSDL (Jacek et al. 2007) initially is not an ontology languages for representing a given domain, rather it is a mechanism that allows the WSDL³⁶ and XML schema³⁷ to have additional tags that refer to a domain ontology. The domain ontology can be in OWL. SAWSDL consists of two parts as shown in Figure 3.2. First is schema mappings, which specifies the data transformations between messages of XML data structure and the correspondence ontology/semantic model; and second is model

³⁵ <http://www.w3.org/2002/ws/sawSDL/>

³⁶ <http://www.w3.org/TR/wsdl20/>

³⁷ <http://www.w3.org/TR/xmlschema-1/>

reference which points the XML Schema element to one or more semantic concepts in the ontology/semantic model.

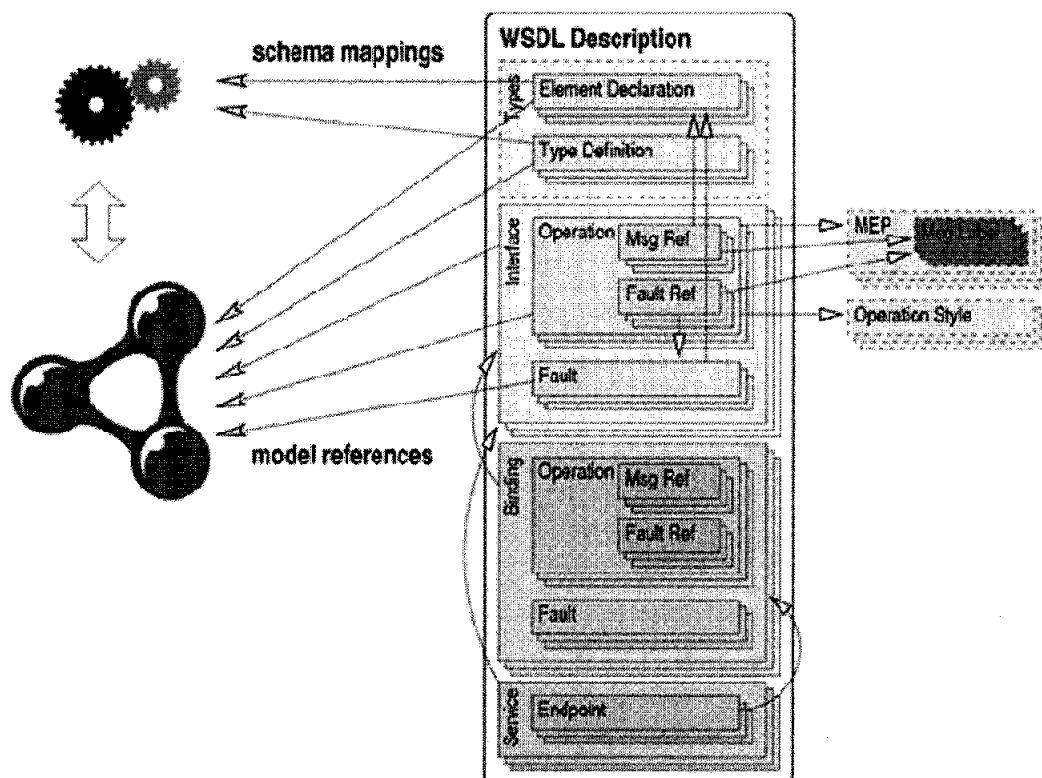


Figure 3.2 SAWDL overview, source (Jacek, Tomas et al. 2007)

- OWL-S³⁸

In order to improve the description and discovery of the semantic web services, there have been some efforts to provide ontology for the semantic web services. This ontology represents an upper layer ontology that is made specially to describe the web service using the ontology languages. OWL-S (formally called OWL-S) has emerged in that context. OWL-S defines a set of *classes* and *properties*, specific to the description of services, within OWL-S. The class *Service* is at the top of the OWL-S ontology. The class *Service* is characterized by three components, which are *service profile*, *service model* and *service grounding* (see Figure 3.3 taken from: <http://www.w3.org/Submission/OWL-S/>). *Service profile* is a class that describes the

³⁸ <http://www.w3.org/Submission/OWL-S/>

capabilities and parameters of a service. Therefore, the class *Service* presents a *ServiceProfile*. *Service profile* answers to the question of ‘*what does the service require of agents, and provide for them*’. *Service model* is a class that describes the workflow and possible execution paths of a service. Accordingly, the class *Service* is described by a *ServiceModel*. *Service model* answers to the question of ‘*how does the service work*’. *Service grounding* is a class that provides information about a service that can be used by an agent to determine if the service meets its requirements. It answers to the question of ‘*how to communicate with the service*’. Hence, *Service* supports *ServiceGrounding*. In short, the *service profile* is used for the discovery, whereas, the *service model* and *service grounding* are used for communication between the service requesters and providers.

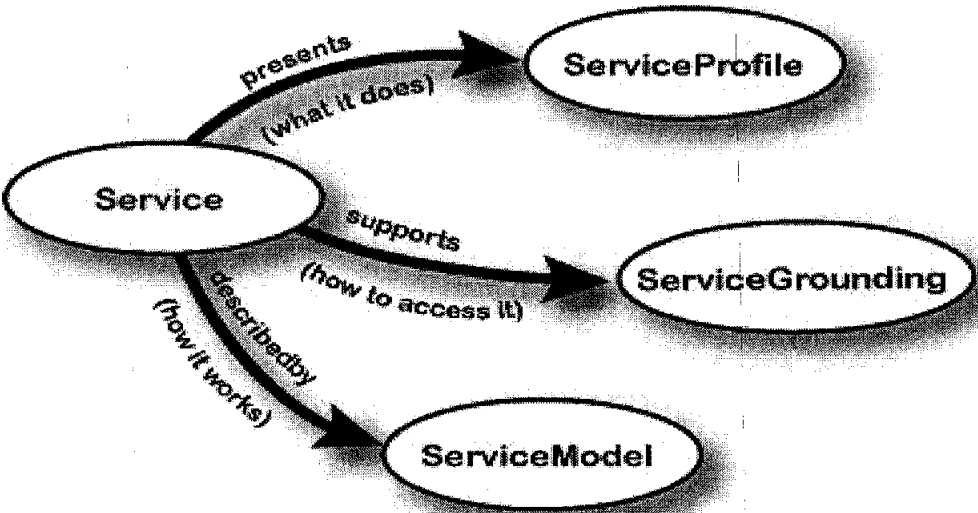


Figure 3.3 The Service Ontology Model

3.3.1.3 Ontology Query Language

In order to manipulate the semantic information and reasoning with them, a query language is most needed in ontology languages. In this regard, in the recent years the W3C has recommended the use of SPARQL as the query language RDF. SPARQL has the capability to query required and optional graph patterns, along with their conjunctions and disjunctions. A SPARQL query contains a set of triple patterns called a *basic graph pattern*. The triple patterns are similar to the RDF triples; the

only difference is that each of the subject, predicate and object may be a variable. A basic graph pattern *matches* a sub-graph of the RDF data when RDF terms from that sub-graph may be substituted for the variables, and the result is a RDF graph equivalent to the sub-graph. It should be noted that SPARQL is data-oriented as it only queries the information held in the models with no inference in the query language itself. Therefore, ontology language such as RDFS and OWL may use their inference engine to produce some entailments against which SPARQL queries are executed.

3.4 Semantic Technology and Its Use in Grid Technology

Initially, the use of semantic technology in grid technology aims at adding well defined meaning to the grid resources, services and other entities so that we can cope with grid heterogeneity and provide self-management to the system. For this, the idea of the *Semantic Grid (SG)* is introduced (Zhuge 2005). In SG, resource and service metadata are exposed and handled explicitly, so that it can be shared and managed by the grid protocols. Several studies have been conducted in transforming the conventional grid to SG (Corcho et al. 2006). This includes providing ontologies to describe the grid domain semantically, and services and mechanisms to accomplish the semantic description and management. In fact, providing grid domain ontology is the most important part in this case.

3.4.1 Semantic Grid

SG has two important aspects, namely ontologies to describe the grid domain semantically, and services and mechanisms to accomplish the semantic description and management. In this line, a SG reference architecture has been proposed by (Corcho et al. 2006) , where the authors extended the *Open Grid Services Architecture* (OGSA) (Foster et al. 2005) to support the explicit handling of semantics, and defined the associated knowledge services to support a spectrum of service capabilities. The architecture is known as the *Semantic-OGSA (S-OGSA)*. S-OGSA defines a *model*, *capabilities* and *mechanisms* for SG.

The model is the elements that S-OGSA is composed of and their interrelationships. It consists of three main components namely: *Grid Entities* (G-Entities), *Knowledge Entities* (K-Entities) and *Semantic Binding* (S-Binding). G-Entities are anything that carries an identity on the grid, including resources and services. K-Entities are special types of G-Entities that represent some form of knowledge. This includes ontologies, rules, knowledge bases, and so on. K-Entity has two types: *knowledge services* and *knowledge resources*. S-Binding are the entities that come into existence to represent the association of a G-Entity with one or more Knowledge Entities (see Figure 3.4 taken from source (Corcho et al. 2006)).

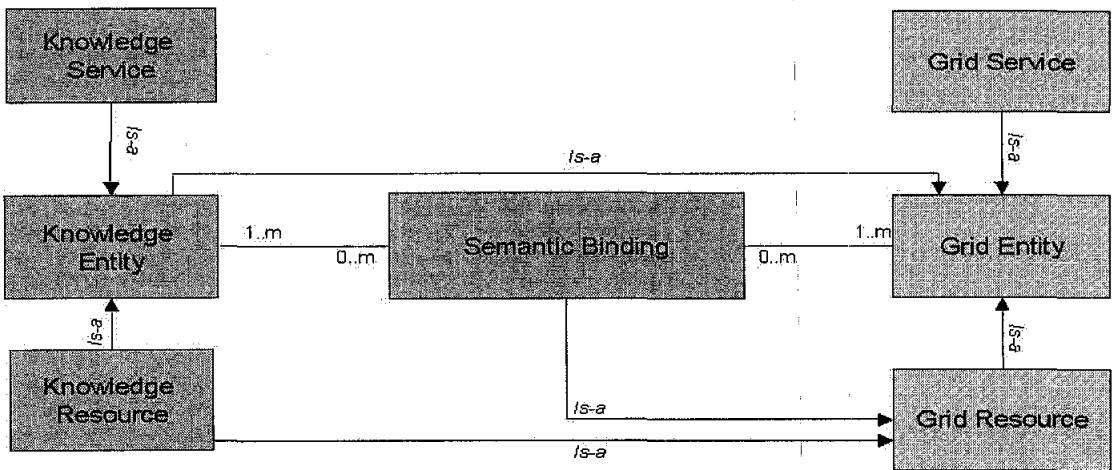


Figure 3.4 S-OGSA entities and their relationships

S-OGSA capabilities are the services needed to deal with the model components. These services should be provided by the grid middleware to include *Semantic Provisioning Services* (SPS) and *Semantically Aware Grid Services* (SAGS). SPS supports the provision of semantics, by allowing the creation, storage, update, removal and access of different forms of knowledge and metadata (i.e. Knowledge Entities and Semantic Bindings of the S-OGSA model). SPS are further divided into two: *Knowledge Provisioning Services* and *Semantic Binding Provisioning Service*. The former include ontology services, which are in charge of the storage and access to the conceptual models of representing knowledge, and reasoning services, in charge of computational reasoning with those conceptual models. Meanwhile, the latter includes metadata services, in charge of storage and access to semantic binding, normally considered as sets of ontology instances, and annotation services, in charge

of generating metadata from different types of information sources. SAGS are those enhanced grid services that deliver OGSA enumerated capabilities semantically.

OGSA mechanisms are to ground the conceptual definitions regarding the use of metadata in the grid into concrete grid modeling element. The first mechanism is that knowledge entities and semantic binding are treated as *grid resources*. The second mechanism states that semantic bindings are delivered by the *grid services*.

3.4.2 Grid Domain Ontologies

As ontologies are the key aspect in SG for modeling resources and services, from the literature survey, there are three studies that have proposed grid domain ontologies. We briefly discuss these studies in this section with aim of highlighting the current efforts in providing grid domain ontology.

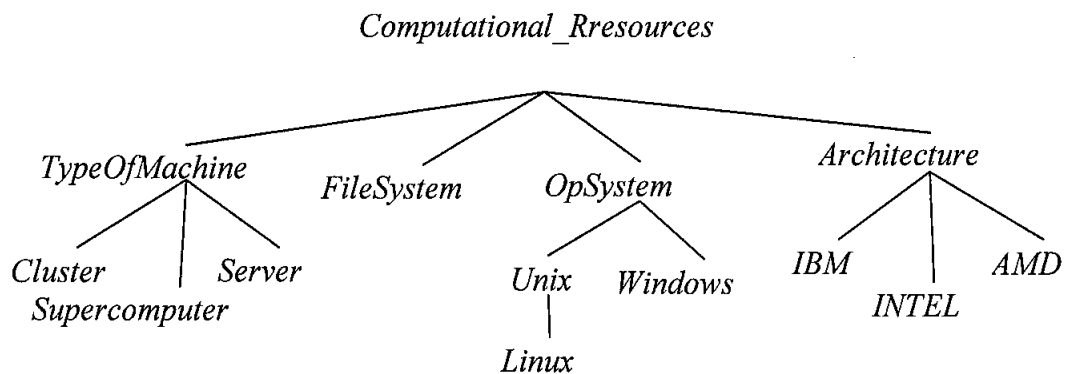


Figure 3.5 The main class grid resource ontology that propose by (Pernas and Dantas 2005)

Pernas and Dantas introduced ontology for resource description to improve the search for resources and their selection. The authors defined a common ontology for the grid environment (Pernas and Dantas 2005). In designing the reference model for the ontology, the authors searched for the most utilized vocabulary by the community, and which resources were commonly employed in grid configurations. The search was realized by considering the NPACI , ESG (Earth System Grid), NASA

Information Power Grid (IPG) and the Distributed ASCI Supercomputer Project 2 (DAS- 2) (Henri et al. 2000). The search was then documented for designing the ontology components, which are *Data Dictionary*, *Concepts Classification Tree*, *Table of Classes Attributes and Instances*, *Table of Instances* and *Tables of Attributes Classification*. Data Dictionary gathers all the classes and instances from the ontology together with their meanings. It has 14 classes and the first class created is *Computational_Resources* (see Figure 3.5). *Concepts Classification Tree* is comprised of all the classes and subclasses of the ontology. *Table of Classes Attributes and Instance* presents to each class and instance all their attributes (e.g. the attribute related to *Cluster* is *TypeOfMachine*). *Table of Instances* accommodates attributes and value of each instance of the ontology. *Tables of Attributes Classification* graphically illustrates attributes, which are deduced upon the existence of other attributes from higher hierarchy. The documentation is then reproduced to OWL language, using the Protégé-2000 editor.

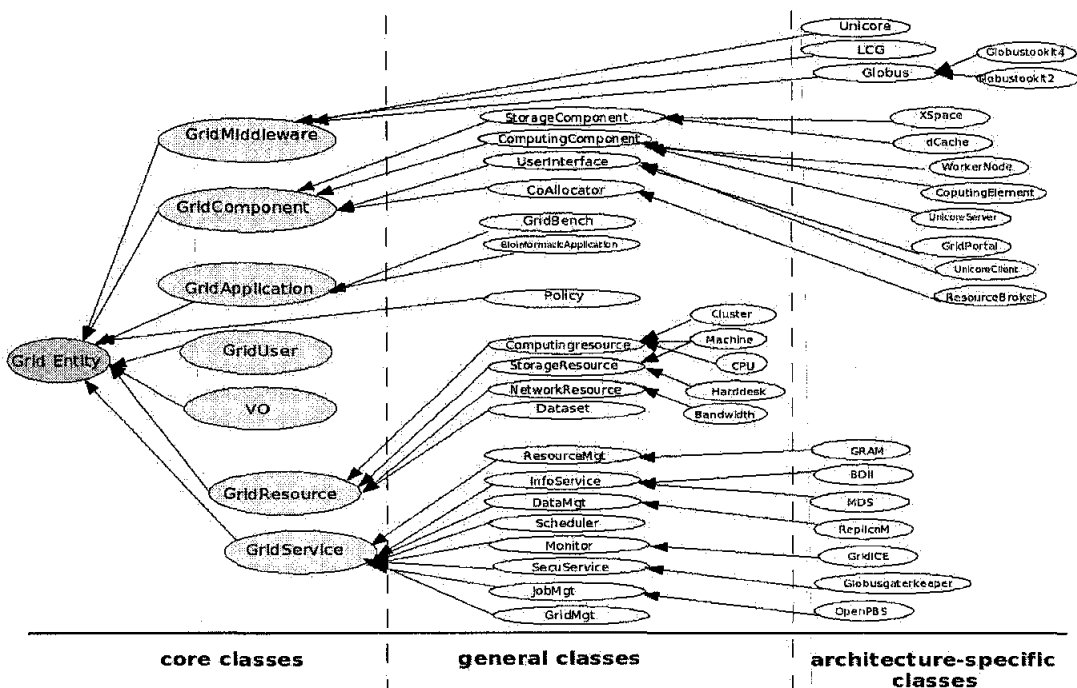


Figure 3.6 Overview of the Core Grid Ontology classes

Xing et al. proposed a core grid ontology that is general enough to capture the grid system, and easily extensible to be used by different grid middleware or grid architectures (Xing et al. 2006). The ontology is built on an abstract, generic model,

which is a layered-structure and designed on three layers scheme. The top layer includes grid VOs, users and application. Grid services and grid middleware lay in the middle layer. The bottom layer contains the grid resources. The basic concepts of the ontology are defined according to the model structure. These concepts correspond to the classes that are fundamental elements or the very important aspects of a grid system. The classes are organized into *core*, *general* and *platform-specific*. Seven core classes of a grid system from the abstract grid model are defined. They are: VO, GridResource, GridMiddleware, GridComponent, GridUser, GridApplication, and GridService (see Figure 3.6). Each of these classes has a description and constraints. For example a GridUser is described as “ a person who can access to a grid ”; while its constraints are: (1) has an ID, (2) registered VO, (3) gridEntry. In order to describe a grid system, the 7 core classes are divided into two parts. First are VO, *GridMiddleware*, and *GridResource* as the three vital and crucial aspects that define distinct features of a grid. The second part consists of *GridUser*, *GridApplication*, *GridComponents*, and *Grid-Service* as the associated concepts of basic grid entities. Subsequently, the authors defined 24 general classes that correspond to the general grid entities referring to *VO*, *Grid middleware*, and *Grid Resources*. These general classes (e.g. *InfoService*, *storageComponent*, *DataMgt*) can be used to describe the Grid in further details. Lastly, the authors introduced the grid platform specific classes to represent the entities of specific grid architecture. For instance, the MDS information service of Globus-2 can be represented by a class MDS, which is a subclass of *InfoService*. To represent the relationships and constraints among the ontology classes, properties are defined to provide a semantic meaning for the Core Grid Ontology. They are defined according to the constraints of the classes. To provide flexibility and extensibility to the CGO, the authors suggested that users can add their classes and properties on “required-to-have” basis. The COG uses the Web Ontology Languge OWL as makeup language.

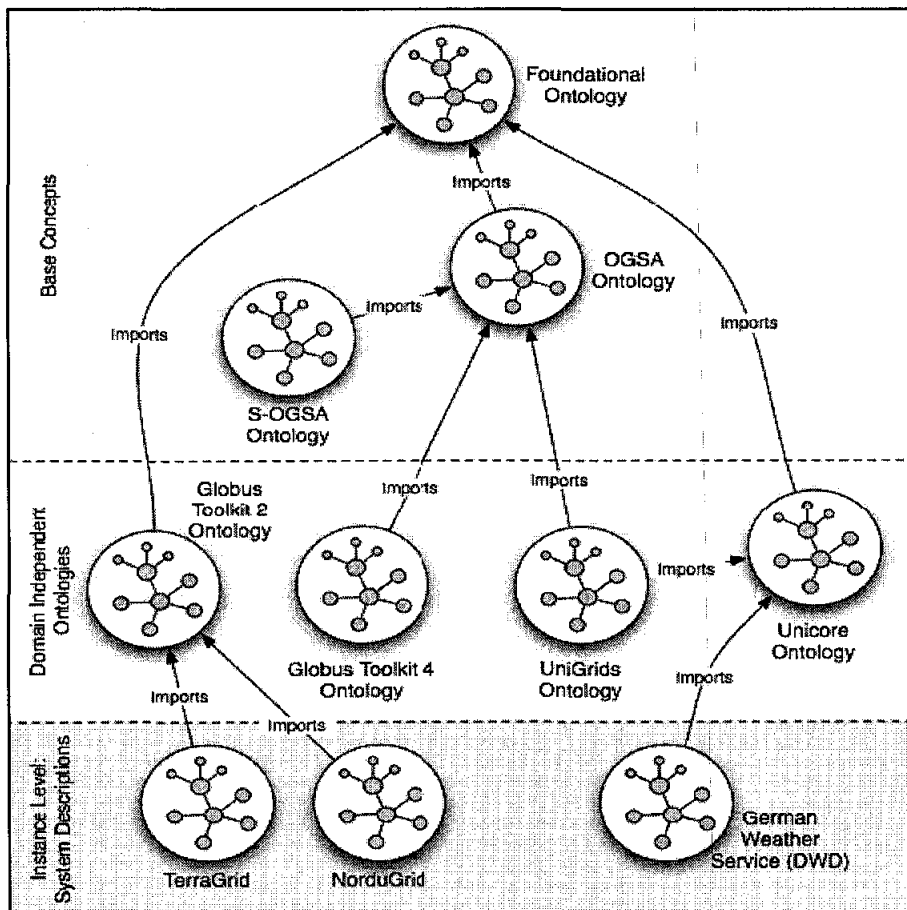


Figure 3.7 The grid knowledge architecture

Parkin et al. also introduced a grid ontology to provide interoperability between Globus and Unicore grid middlewares. Whereby, a resource broker can use resource information that are described by either the Monitoring Discovery Service (MDS) of the Globus or the Unicore Incarnation DataBase(IDB) (Parkin et al. 2006). The authors defined two requirements for the proposed ontology: (i) the ontology must allow consumer /resource provider to express resource requirements in an abstract, resource and middleware independent form; and (ii) the ontology must express again, in an abstract manner, both the actions requested and the resources that enable these actions. To fulfill the requirements, they defined three layers grid knowledge architecture: *Basic Concept*, *Domain Independent Ontologies* and *Instant level* layers. Ontologies in these layers can import concept each other hierarchically (a lower layer ontology can import from its upper layer ontology) (see Figure 3.7 taken from (Parkin et al. 2006)). The *Basic Concept* layer includes *foundational ontology* that defines the

high-level, common, general-purpose grid ontology concepts that can be reused in the description of any grid middleware or application, protocols, services, resources and Virtual Organizations. The foundational ontology includes 104 classes and 154 object properties. In the domain of independent ontologies layer, there are middleware-specific ontologies that describe the instances of implementations of grid middleware. Examples of these middleware are Globus, Unicore , UniGridS³⁹ and so on. All of these ontologies import and extend the basic concepts from the upper layer. The final layer is the instant level, this contains the actual grid deployments ontologies. They hold the details of the grid systems such as The UK National Grid Service⁴⁰ and US TeraGrid⁴¹. This ontology is implemented using the OWL.

From RD system perspective, the use of semantic technology means the involvement of semantic technology in the RD components mainly in description. This means, resources and services are semantically described, which is known as *semantic information*. However, some studies have gone beyond using semantic information for description to other RD components such as registration. In this case, the resource and service registries are distributed semantically so that the distribution of the resource and service queries during the discovery process will be based on semantic sub-registries. To this end, in the rest of this thesis we mean the term “semantic-based RD system” to any RD system that involves the use of semantic information in its components. We also use the term “semantic-based RD system” and “RD based on semantic information” interchangeably.

3.4.3 Semantic-Based Grid RD Systems’ Description

The use of semantic information in description components of grid RD systems is generally the application of the available ontology languages or grid semantic information, and data models. For this, we classify the descriptions of the RD systems based on their use of which semantic description, and what that semantic description supports. The taxonomy initially classifies RD system description into *using semantic*

³⁹ <http://www.unigrids.org/>

⁴⁰ <http://www.ngs.ac.uk/>

⁴¹ <https://www.teragrid.org/>

information (semantic description) and *non-semantic information* (keyword-based description). Semantic description is further divided into *using the same ontology* and *using different ontologies*. The former is further divided into *supporting one middleware information service* and *supporting different middleware information services* (see Figure 3.8).

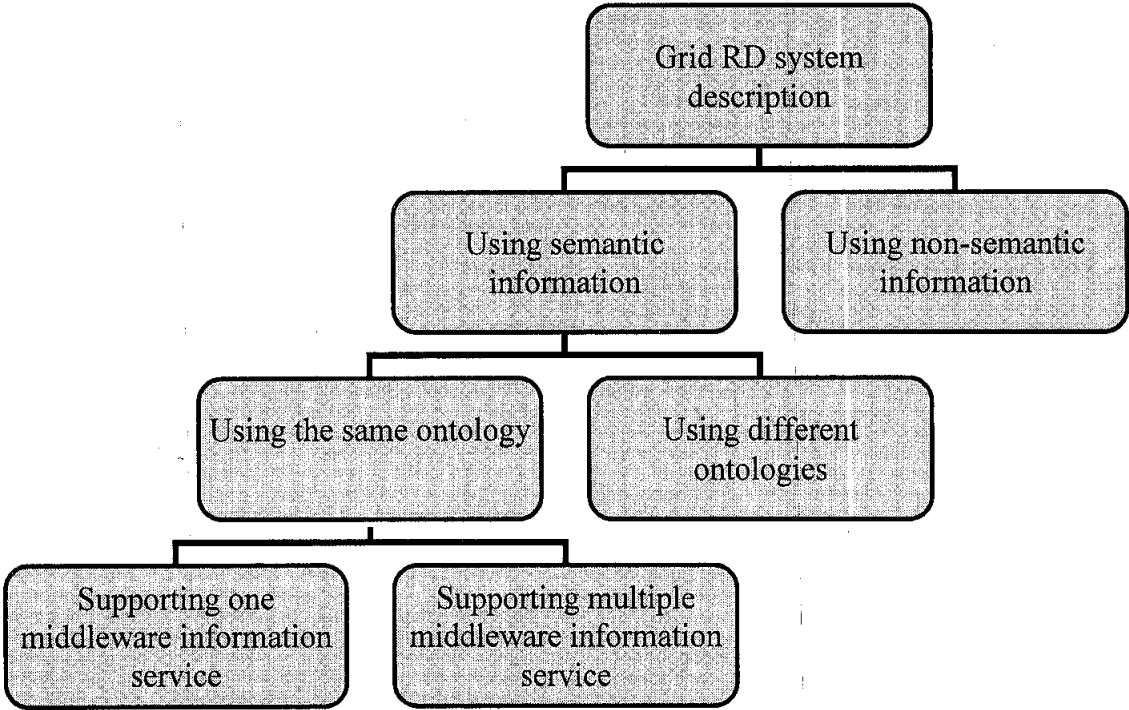


Figure 3.8 The taxonomy of semantic-based RD system description

3.4.3.1 Using the Same Ontology

Using the same ontology is a kind of grid RD system that uses homogeneous information and data models. The RD systems that fall in this class normally use the existing grid domain ontologies that we have discussed earlier to build a semantic grid metadata service. This semantic metadata service can integrate metadata sources that belong to a particular middleware information service or multiple grid middleware information services.

- *Supporting One Grid Middleware Information Service*

Pernas and Dantas used their own ontology, that we have mentioned above, with an interaction service to build a semantic grid information service on top of the Globus Monitoring and Discovery Service (MDS) (*Pernas and Dantas 2005*). The main elements of the system are *ontology*, *metadata*, *semantic view* and *MDS* (see Figure 3.9). Metadata stores all the information about existing resources, and semantic view gets the status of the resources from the MDS. Ontology utilizes metadata and semantic view to obtain information about any computational resource in order to answer any user resource queries. The interaction between the ontology and consumers is made through a Java based application service. The service has three modules. The first module provides a list of all classes and instances defined in the ontology. The names of these classes and instances are used by a consumer to process queries into the metadata and computational resources of the remaining modules. In the second module, consumers can search for metadata from any class listed by the first module. The third module allows a search of any existing computational resource, where a consumer can visualize the entire configuration. The system was tested at the Federal University of Pelotas, and has shown a clearer description of the computational resources.

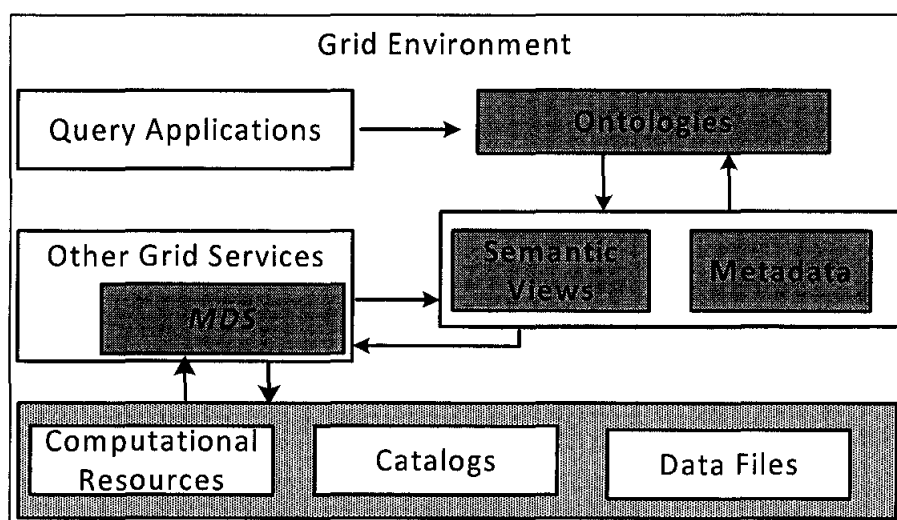


Figure 3.9 The grid architecture using the ontology approach as proposed by (*Pernas and Dantas 2005*)

Somasundaram, Balachandar et al. proposed a knowledge layer on top of the Gridbus broker architecture for semantic description and discovery of resources (Somasundaram et al. 2006). This yields five layers of grid architecture, which are *fabric*, *core middleware*, *high level middleware*, *knowledge* and *application* layer. The knowledge layer provides knowledge discovery from a huge amount of data aggregated from underlying information services layer. The knowledge layer consists of three main components, and works with other two additional components. The main components are *resource description*, *semantic repository*, and *resource discovery*. Meanwhile, the additional components are *monitoring and discovery service* (MDS), and *job description* (see Figure 3.10). The resource description defines resource ontology template, and provides necessary concepts and properties with which a resource can be described. Different possible computing resources are considered for creating ontology template (although the authors did not provide any details about the ontology structure, it is obvious that they used one grid domain ontology). Semantic repository is made up from the ontology and knowledge base. The knowledge base is built with the instances and specific property instantiations of the ontology of the resource description. Resource discovery allows users to submit their queries. It then generates appropriate Algernon query depending on the requirements specified by the user, and executes these queries over the ontology knowledge base to obtain the best possible resources that closely match the request. The MDS provides the value of the properties concepts for the ontology. The discovery process is done as follows: user forms a query with the format *label: label_value* in which the properties of the resource are denoted as *label* and requested value as *label_value*. Query generator converts the query into an Algernon query. The discovery executes the queries over the knowledge base of the semantic repository and finds the resource that corresponds to the user's request.

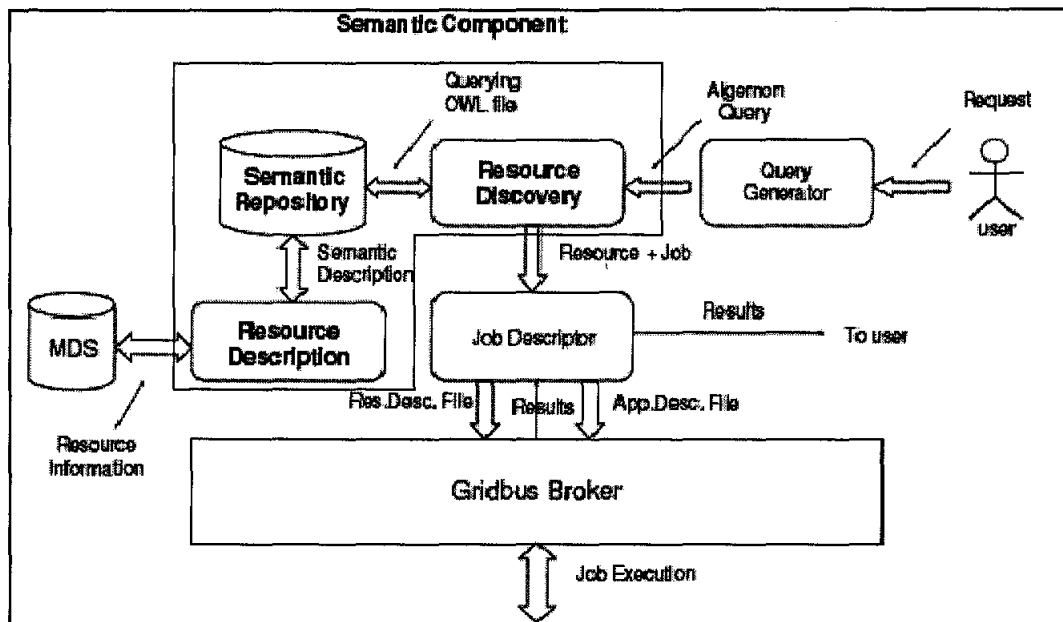


Figure 3.10 The semantic-based RD model presented by (Somasundaram et al. 2006)

Said and Kojima work proposed a RD system known as *Semantic Monitoring and Discovery System (S-MDS)*, which is built on top of the Globus Toolkit 4 (Said and Kojima 2009). The system uses ontology, OWL and other services to create the semantic resource metadata. The ontology does not have a predefined full structure, as the non-semantic resource metadata that are presented as XML resource properties (RP) documents are mapped into OWL classes. These classes construct an ontology called *domain specific ontology (DSO)*. This is done through three services and a semantic repository, which are *semantic metadata manager (SMM)*, *semantic metadata provider service (SMP)*, *semantic metadata index service (SMI)*, and a *RDF repository* (see Figure 3.11). Respectively, SMM reads any RP, maps it into the ontology (DSO), and instantiates the ontology using the defined values in the RP. The created ontology instance (semantic metadata (SMD)) contains all the values specified in the mapped RP. The SMM also allows users to enrich any SMD further by associating it with other relevant ontologies, and then instantiating these ontologies. The enriched SMD is then published at SMP through a registration process. SMP service stores SMDs in their RPs and monitors SMDs changes by tracking their mapped RPs. Therefore, when a RP is updated, the SMP updates the corresponding

SMD to reflect the change. The SMP finally registers the SMDs into one or more SMIs. SMI service uses a RDF framework to store and maintain the SMDs that are registered by SMPs. SMDs are stored in a local or remote RDF repository and can optionally be placed in the SMI's RP. SMI is similar to SMP in a sense that it monitors the changes of the stored SMDs in SMPs. When SMD is updated, SMI updates the corresponding stored metadata. S-MDS uses SPARQL query language for querying the RDF repository. The system also provides some GUI-based tools for creating, enriching, and registering the semantic metadata, and constructing SPARQL queries. S-MDS has been implemented and its result is promising.

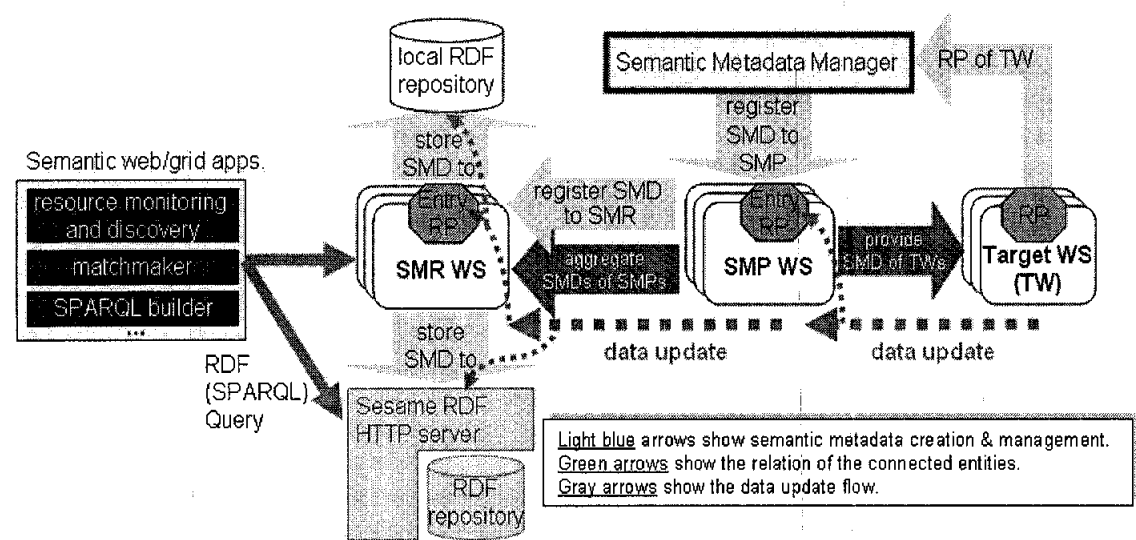


Figure 3.11 The S-MDS system architecture proposed by (Said and Kojima 2009)

- *Support Multiple Grid Information Services*

RD systems fall in this class that uses ontology to integrate multiple grid information services to provide a semantic grid information services on top of these services. Normally, this kind of RD is related to intergrid level.

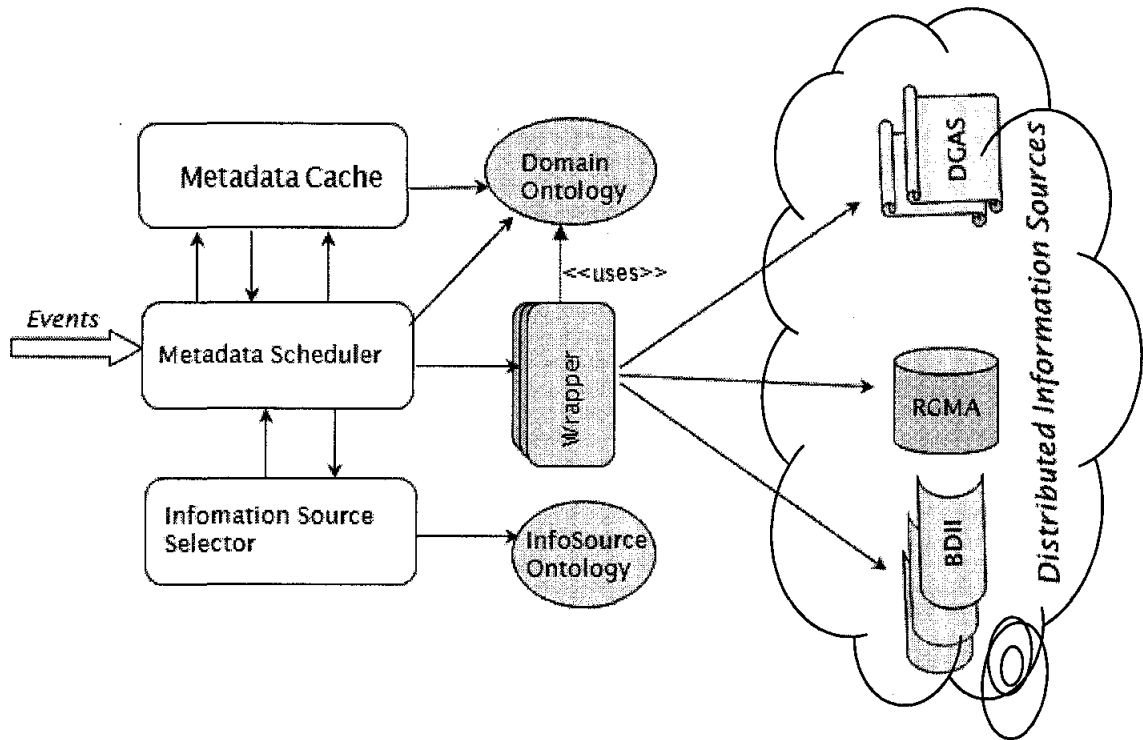


Figure 3.12 Overview of the active ontology architecture

Xing et al. proposed an ontology based information services integration for grid infrastructures such as Enabling Grids for E-science⁴² (EGEE). The model allows grid distributed metadata services such as BDII⁴³ and Globus MDS to be integrated into a common pool (Xing et al. 2010). Active Ontology (ActOn) (Xing et al. 2007) is used for this integration. ActOn initially is an information integration approach that is able to generate and maintain up-to-date metadata in a dynamic, large-scale distributed system. It consists of a set of knowledge components and software components. In the proposed model, knowledge components comprise *domains* and *information sources* ontologies. *Domain ontologies* describe information and data models for the resources, components, services, and applications of the EGEE (see Figure 3.12). For this, the author uses the ontologies of (Xing et al. 2006) and (Parkin et al. 2006) as domain ontologies. *Information sources ontology* provides information

⁴² <http://public.eu-egee.org/>

⁴³ Berkeley Database Information Index (BDII): <http://lfield.home.cern.ch/lfield/cgi-bin/wiki.cgi?area=bdiipage=documentation>.

about the information services that are deployed in EGEE. The two ontologies are related by means of *mappings* that identify which domain concepts, and which of their properties can be generated by which information sources. Software components include *metadata cache* (MC), *metadata scheduler* (MSch), *information source selector* (ISS), and a set of *information wrappers*. MC stores and manages the metadata that are obtained from the information sources. The metadata use the domain ontology as a data model for the stored metadata. The MSch updates the information in the *metadata cache* using on-demand based policy. The demand takes place when there is an event such as queries. *Information source selector* is used to select the most suitable information sources among the available sources, which are described in the information sources ontology. The selection is based mainly on the actual information needed and the geographical proximity. *Information wrapper* is used to retrieve the up-to-date information from the sources. Wrapper is called by the metadata cache as soon as the selection of the information resources is done. Each type of information source has a special wrapper. The system uses SPARQL as the query language. The proposed system has been implemented and compared to other existing grid information services; it shows promising results.

3.4.3.2 Using Different Ontologies

RD systems are those which do not restrict resource and service providers to use common domain ontology, rather they propose a specific ontology language and/or a set of rules upon which each resource or service provider site can build its own ontology. The sites' ontologies will then be federated to have a virtual ontology or remain as they are. Several studies have proposed the idea of using different ontologies as we see next:

Li and Vuong proposed RDF as a data model for grid resource providers to encode their resources' metadata without any defined specific grid domain ontology (Li and Vuong 2005). Each grid node stores its resource metadata in a local database which belongs to the node itself. The stored metadata is then summarized using the Bloom filter, which will make the query process easier. A hash function is used to map the resources with their attributes. For example, if x is a resource and (H_1, H_2) are

functions, so $H(2, 3)$ means x is located in the second and the third bit in the Bloom filter bitmap. The query matching has two phases. First, a matching between the query and the resource summary, if there is a complete or partial matching based on the resource attributes combination. For example, if R is a resource with two attributes $\{a, b\}$ it can satisfy 3 queries, $\{b\}$, (Padmanabhan). If the first matching exists then it goes for the second phase, which involves the database of the resource provider.

Ludwig and Reyhani proposed DAML-S to describe grid services such as authentication, authorization, job submission and applications (Ludwig and Reyhani 2005). In this case, each grid service will have the DAML-S's description components, which are *profile*, *model* and *grounding*. All of the information related to these components for each grid service is stored locally (service provider). As service profiles contain the functional capabilities, the profiles are then federated and stored in a registry for matchmaking with the service requesters. The authors extend the work in (Ludwig and Reyhani 2006), and propose DAML language instead of DAML-S. For this, each grid service is treated as a DAML class and the request of a service is also considered as DAML classes. The set of service classes from different sites construct an ontology named *grid service ontology*. Likewise, the set of application classes from different consumers create another ontology known as *application ontology*. The query system is done through the similarity calculation between an application class and service class using the DAML parser, which parses the grid services ontology.

Groleau et al. proposed OWL-S for both resource description and request, where each resource and request is presented as an OWL-S class through its elements, which are profile, model and grounding (Groleau et al. 2007). The selection of a resource takes place by a matchmaking process between the profiles of resource and the resource request using Pellet OWL Reasoner⁴⁴.

Han and Berry assumed the availability of ontology to describe the resources in each grid resource provider (Han and Berry 2008). However, this ontology should fulfill the definition of (Gruber 1995), which is about ontology components (concepts,

⁴⁴ <http://www.mindswap.org/2003/pellet/index.shtml>

relations, axioms), the hierarchy of concepts and the equivalence between concepts. The query system is based on the calculation of semantic similarity between the concepts. The similarity of concepts represents the degree of commonality between concepts, which are the requested resource and the advertised resource. The similarity function of (Andreasen et al. 2003) is used to perform the similarity degree calculation, which can result a range of values between “0” and “1”.

3.4.4 Semantic-Based Grid RD Systems’ Registration

As we have mentioned, RD registration involves two aspects: *registry architecture* and *update mechanism*. In this section we discuss semantic-based RD systems’ registration components. Our discussion covers three kinds of RD systems: (i) RD systems that use semantic information in their description and registration components, (ii) RD systems that use semantic information in their registration components only, and (iii) RD systems that use semantic information in their description only. Accordingly the registration components of the first two types of RD systems are called semantic registrations as semantic information is involved in both. Meanwhile, the registration components of the last type of RD systems are called non-semantic registration as they do not involve semantic information. We further classify each class based on registry architecture as shown in Figure 3.13.

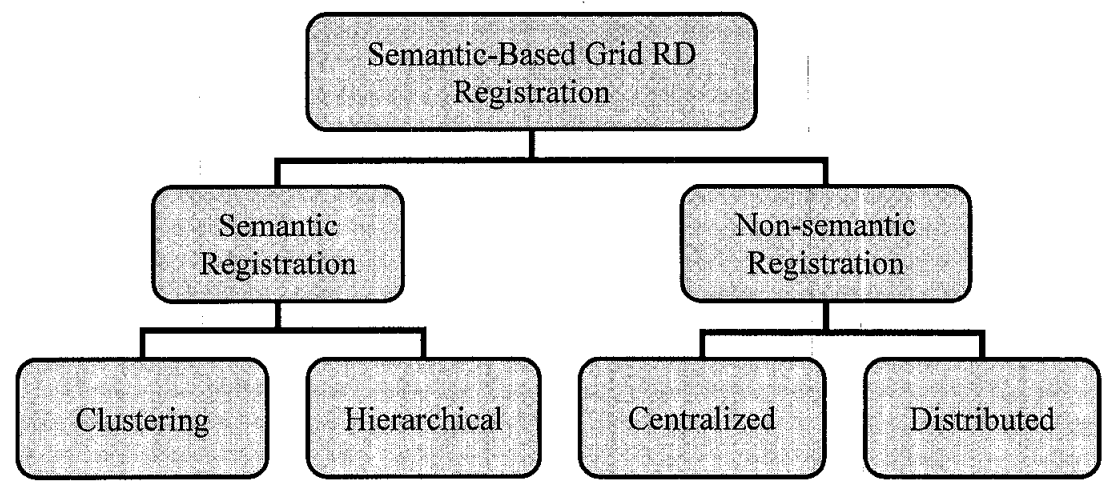


Figure 3.13 The taxonomy of semantic-based RD systems registration

3.4.4.1 Semantic Registration

RD systems that use semantic registrations keep their resources and services metadata into distributed sub-registries. The distribution of the sub-registry nodes is based on some well defined concepts such as resource interest, geographical locations and application type. In turn, resource and service queries are distributed to the sub-registries according to the defined concepts. Such systems sometimes use ontology to drive concepts to build their registrations. Semantic registration models are further classified architecturally into *clustering* and *hierarchical*.

- *Clustering*

In clustering architecture, nodes are classified according to some groups; each group represents a well defined concept related to a type of applications or resources. Each group will have a registry (sub-registry). The sub-registries are connected with each other to construct a network in top of the groups.

Li and Vuong implemented a P2P architecture to construct the registry. Nodes are grouped into some clusters according to their resource interests (Li and Vuong 2005). Each cluster is a tree structure. Every node in the tree has a local resource summary as well as aggregated summaries from the children nodes. Consequently, the top root node will have the entire knowledge about the cluster. To share the resources among the clusters, root nodes are connected with each other to form an overlay network on top of the clusters. In addition to that, each root node may have some knowledge about its neighboring root nodes. The update mechanism is through the propagation of the update messages from the resource provider nodes to the root nodes.

Li and Vuong again proposed a semantic community-based P2P approach but, in this time, the authors assume the availability of an ontology in which domain interest of the grid nodes is defined (Li and Vuong 2006). The domain interest is used to categorize the nodes to communities (groups). The SkipNet P2P network is used to organize the node communities. It has two overlay layers. The first contains all grid nodes, in which nodes are organized in multi rings for each defined domain interest.

Each node has a numerical ID and name ID. Numerical ID is obtained by hashing the node IP address, and the name ID is obtained by concatenating node interest and its identification. The second layer is the category overlay, which accommodates representative nodes for the different domains. Therefore, each domain interest has one node to represent it. When a node wants to join the system, it registers its interest in the category overlay. The category overlay returns the new node all the related interest responsible nodes. The update mechanism is such that the system imposes on each node to update timely its own resources information, and upon the domain representative nodes to update the existence of the domain nodes.

- *Hierarchical*

In hierarchical registration model, the sub-registries are organized in hierarchical manner which means that each semantic defined parent sub-registry accommodates the related information of its child sub-registries. The work of (Kou et al. 2007) is example of this model as it proposes a registration that is based on classifying the nodes to some groups named *Personalized Grid Information System* (PGIS). Grid resources are considered as services, and personalization is done on these services through a rank *model*. The model ranks nodes into three ranks, which are *resource service* (RS), *virtual organization cluster point* (VCP) and *domain cluster point* (DCP). RS represents service providers, requesters or the base services with simple operations. Services and users with similar characteristics are grouped, and the group is treated as a virtual organization (VO), which provides a centralized management. This management includes metadata access, retrieval and storage. VCP is used to describe the VO it represents and the sub-registry for the VO, which is in charge of recording the information and characters of RSs in the corresponding *VO autonomy region* (VAR) to provide uniform access to dynamic and static information. DCP represents a domain registry, which is responsible for managing the information of all VCPs registered in its *domain autonomy region* (DAR), query forwarding and information accessing, retrieving and storing. The update mechanism is that, each RS updates its VCP about the status in each period of time.

3.4.4.2 Non-semantic Registration

RD systems that use non-semantic registrations are those that do not involve semantic information in their registration models. Such RD systems include those that use semantic information in their description components and those that do not. However, the latter systems we have already discussed them in the previous chapter a more about the can be found some of the related work (Zanikolas and Sakellariou 2005) and (Mastroianni et al. 2008). We classify the former systems based on their registries' architectures, which are centralized and distributed models.

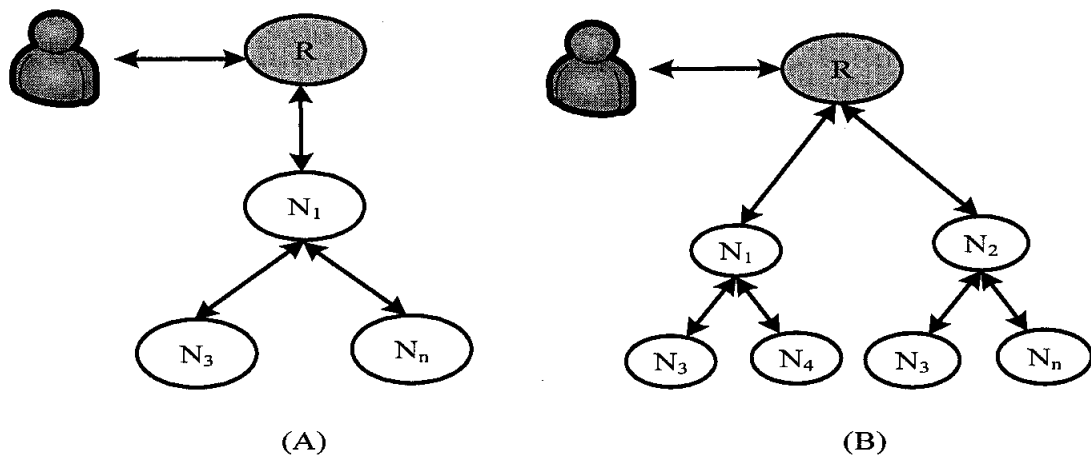


Figure 3.14 (A) A centralized registry on top one hierarchical information service; (B) centralized registry on top of two hierarchical information services

- *Centralized*

In centralized registration models, the semantic resources or services metadata of an entire system is indexed under a centralized registry and users would send their resource and service queries to that registry. It should be noted, centralization of registration in semantic-based RD systems is different from the key-word based RD systems as normally the earlier is built on top of the former, which itself can be centralized or hierarchical. Figure 3.14 describes the centralizations of the semantic-based RD systems, where (A) represents a registry (node R) that is located on one hierarchical grid information service, and (B) represents a registry (R) that is located

on top of the set of hierarchical grid information services. Most of the semantic-based RD systems use one hierarchical grid information service (case (A)) as their information sources. Systems such as (Pernas and Dantas 2005), (Somasundaram et al. 2006), (Ludwig and Reyhani 2005), (Ludwig and Reyhani 2006) and (Groleau et al. 2007) are included in this class. Meanwhile only one study in the literature uses multi hierarchal grid information, which is (Xing et al. 2010).

- *Distributed*

Semantic-based RD systems that fall in this class use a flat P2P network as registry architecture. In this case, each resource provider node registers its resources and maintains their metadata in its own registry, and it may inform its neighboring nodes about its resources. The work of (Han and Berry 2008) is an example of this class, where each node has its own registry whereby it accepts the resource queries from the other nodes and solves them. The node also provides a summary of its resource information to its neighbors. As the registry is local, the update mechanism is done directly on the registry from the resource sensors. In turn, the node updates its neighbor about the changes.

3.4.5 Semantic-Based Grid RD Systems' Discovery

As we have described that RD system discovery includes *search* and *selection* of the registered resources in the registry. In fact, is quit hard sometimes to differentiate between the two sub-components as they may be independent or integrated as one component. In This section, we discuss the search and selection in semantic-based RD systems.

3.4.5.1 Search

Search initially depends on the registration architecture as it supposed to search the resources that are registered in the registry(s). Therefore, the involvement of semantic information in registration has a clear impact on the search performances. However,

generally all but the discussed Semantic-based RD systems do not focus on the search algorithm as they relay on the keyword-based RD systems search algorithm. The latter as we have mentioned in the previous chapter that they implement or extend one or more of the known *Packet Propagation* algorithms such as *Unicast* , *Multicast* , and *Anycast*. This implies that RD system such as (Ludwig and Reyhani 2005) ,(Pernas and Dantas 2005), (Somasundaram et al. 2006), (Said and Kojima 2009), (Xing et al. 2010), implement *unicast* search algorithm. Where, the query is sent directly from resource requester node to a centralized registry. The work of (Kou et al. 2007) implements both unicast and multicast in different levels. The system initially has two types of registry which are cluster registry and domain registry, so between the resource requester node and cluster registry a unicast is used. If the cluster registry cannot solve the query, another unicast will take place between the cluster registry and domain registry. If again the domain registry cannot solve the query a multicast will take place between the current domain registry and the other domain registries.

Some RD systems extend the anycast algorithm, as the case of (Li and Vuong 2005), in which each node checks its local cache where it has some information about its neighbors. If it cannot get an answer to its query, it will select one of its neighbors and forwards the query. The query will be forwarded with this manner until an answer is got or the TTL expires. In (Han and Berry 2008), the request node sends queries based on the information exchange with its neighbors. Therefore, it sends to neighbor that most likely may have the resources. The neighbor in turn forwards the query using the same manner when it cannot solve the query until an answer is found.

3.4.5.2 Selection

Semantic based RD systems such as the one proposed by (Pernas and Dantas 2005), (Said and Kojima 2009), (Xing et al. 2010), and (Kou et al. 2007) use manual selection. On the other hand, studies such as (Ludwig and Reyhani 2005), (Ludwig and Reyhani 2006) and (Groleau et al. 2007) use matchmaker (algorithm). The work of (Han and Berry 2008) also implement matchmaking but using agent as the agents themselves are matchmakers. In such situation, the resource requester and the

resource provider are presented as agents. Each of them describes its capabilities. Requester agent interacts with the provider, performs the matchmaking process. If the capabilities of the provider meet the requirements, the requester agent selects that provider. Lastly, the study of (Somasundaram et al. 2006) uses broker.

Table 3.1 Comparison summary of semantic-based RD systems description

Author(s)	Information Model	Data Model
Authir(s)	Information Model	Data Model
Pernas and Dantas 2005	One common ontology with a project scope	OWL
Li and Vuong 2005	Each resource provider may have its own ontology	RDF
Ludwig and Reyhani 2005	Each grid service provider uses its own ontology	DAML-S
Li and Vuong 2006	Hash tables	XML
Ludwig and Reyhani 2006	Each grid service provider may have its own and the applications also have their own ontology, the ontology will then construct to ontologies, one for application and another for services	DAML
Somasundaram et al. 2006	One common ontology with a project scope	OWL
Groleau et al. 2007	Each grid resource provider uses the OWL-S framework	OWL-S
Kou et al. 2007	Extended UDDI	XML
Han and Berry 2008	Each grid resource provider has its own ontology	Not defined
Said and Kojima 2009	One common ontology that is constructed by all the resource providers	OWL
Xing et al. 2010	One common ontology that is supported by information sources ontology and integration tools	OWL

3.5 Discussion and Comparison Summary

In this section, we present a qualitative comparison between the presented semantic-based RD systems. The comparison is based on the effectiveness of the systems with regard to current the identified RD requirements which include interoperability, scalability, decentralization, and dynamism. We discuss how these requirements are met by the respective semantic-based RD systems. In fact, each requirement is related to one or more RD components. For example, interoperability is related to description, whereas scalability, decentralization and dynamism are related to registration. Note that, this relation does not mean we can meet a requirement fully through the performance of the corresponding component(s), rather, they may be dependent upon each other in meeting the requirements. For example, it does not make sense if we have scalable registration without an expressive description, then there is no sense for the scalability. Table 3.1 and 3.2 illustrate a qualitative summary of description and registration mechanisms of the discussed systems.

Table 3.2 Comparison summary of semantic-based RD systems registration

Author(s)	Registry type and architecture	Update mechanism
Pernas and Dantas 2005	Non-semantic centralized registry	Timestamp
Li and Vuong 2005	Semantic clustering registers	Timestamp
Ludwig and Reyhani 2005	Non-semantic, centralized registry	Timestamp
Li and Vuong 2006	Semantic , clustering registers	Timestamp
Ludwig and Reyhani 2006	Non-semantic centralized registry	Timestamp
Somasundaram et al. 2006	Non-semantic centralized registry	Timestamp
Groleau et al. 2007	Non-semantic centralized registry	Not defined
Kou et al. 2007	Semantic , Hierarchical registers	Timestamp
Han and Berry 2008	Non-semantic, distributed registries	Timestamp
Said and Kojima 2009	Non-semantic centralized registry	Timestamp
Xing et al. 2010	Non-semantic centralized registry	Timestamp

3.5.1 Interoperability

Interoperability entails a meaningful description for resources and services that can cross the different middlewares, languages, and programming environments. Meaningful description relies on well accepted ontology that is able to be extended with the required time. Using common ontology seems to be more close to meeting interoperability if it is able to provide two aspects. First, the common ontology should be able to integrate all grid information services that may belong to different middlewares; for example, the Globus MDS and gLite BDII or allow each participant to describe its resources and services. Second, the defined concepts of the ontology should be acceptable to all the parties (e.g. inspiring the concepts from the standard recommendation provided by OGF⁴⁵). In fact, most of the semantic-based RD systems that use common ontology integrate only resource information that comes from one grid middleware information service. Initially, semantic technology is associated with a high computational cost compared to key-word based models. Therefore, the trade-off of using semantic technology is to provide users with an easy view of the shared resources and flexible resource/service query matching, which is supported by key-word approaches, rather than achieving interoperability. In such situation, it would be better to have a key-word based common information model between all participants especially if the type of shared resources is dominated by the hardware resources. However, the work of (Xing et al. 2010) uses a common ontology that integrates more than one grid information service that belong to different middleware. This study can be considered as the most interoperable compared to other RD systems in this context.

On the other hand, we have some RD systems which use different ontologies. They seem to be more flexible for management by local grid nodes as there is no need to describe the resources in syntactic manner, and later they can be integrated into semantic pool as the case of some that use common ontology systems such as (Xing et al. 2010). However, they raise another issue, which is semantic interoperability or in other words, how to ensure that two concepts from two different ontologies are referring to the same resource that they represent. Therefore, using different ontologies do not bring the system closer to achieving interoperability. In fact, there

⁴⁵ Open Grid Forum <http://www.ogf.org/>

are some ontology tailoring mechanisms (Flahive et al. 2009) that work on a grid environment for adopting an ontology from the existing ones. This may be used to adopt existing common ontologies among the grid participants instead of having different ontologies.

3.5.2 Scalability, Decentralization and Dynamism

Scalability, decentralization and dynamism requirement are much related to registration than the other RD components. To fulfill them, there is a need for a scalable decentralized dynamic registration component model on top of an interoperable semantic description. Scalable decentralized dynamic registration requires a well established distributed registry architecture that can provide both a low latency during the discovery process, and a dynamic update mechanism that makes metadata in the registry reliable.

Most of the semantic-based RD systems use centralized registration model as they are initially built on top of the existing grid information services. Centralized registration would be effective in small grids, but it may suffer in intergrid level due to the large number of resources, services and users. Most of the centralized registration models also use a time defined update mechanism, which causes high traffic messages due to the dynamic nature of the grid in addition to the expected traffic associated with centralization of the registry. Time defined update mechanism would be effective in decentralized registry. Therefore, the work of (Xing et al. 2010) introduces on demand update mechanism which is indeed suitable for the centralized registry.

On the other hand, we have other semantic-based RD systems that have distributed registry architecture, most of which use semantic information to distribute the sub-registries across the network, while few do not. When semantic information is used to distribute the sub-registries, the resource request will be pointed only to the most reliable sub-registry, and failure of a sub-registry may not affect the whole grid system. In addition to that, the management of the resource information in terms of update and query processing will be much easier. This idea is promising for achieving

scalability, decentralization, and dynamism compared to the canalized registry. However, there are some factors that need to be taken into consideration such as, providing a systematic distribution to ensure some balance in both, the number of resources and the type of resources that are assigned to each sub-registry. Apart from that, the description of sub-registry concepts should be expressive. These factors have not been considered by most of the semantic RD semantic registrations that we have reviewed here.

It would be very difficult for semantic-based RD system that use non-semantic, distributed registration to have scalability as the resource request will be forwarded from node to node, and controlled by some mechanism such as TTL, which may cause low precision. They may be effective in small grids as they have low cost in their update mechanism since each node updates only its neighbors at far.

All in all, the main issue in semantic-based RD systems is that each work focuses either on the description or registration. This means a system may meet only one or two of the RD requirements. Therefore, most of the discussed studies have not been able to provide interoperability, scalability, decentralization and dynamism at one time to fulfill the intergrid level requirements. The reason behind that maybe from the ad-hoc nature of the studies and/or the ultimate aim of each project. Hence, in order to have an RD system that meets the defined requirements, research communities should work in how to synthesize the good features of the discussed studies. For example, the expressive description of (Xing et al. 2010) with centralized registration may benefit from the idea of the semantic distributed registration model of Li and Vuong (Li and Vuong 2005). Another aspect we urge to be considered is that, the scope of description and discovery should be extended to services as most of the discussed systems are restricted to hardware resources. In fact, this is very vital in the current intergrid level systems, in which we need to discover not only resources but also services such as job management and security. Furthermore, it is also to make the current developed grid applications (e.g. medical imaging) more reusable. They may be treated as services; therewith they can be described and discovered.

Another issue with the current semantic-based RD systems but is related to the semantic technology itself, is the high computation cost. Semantic technology has

been initially used in web service applications which are dominated by sharing services (e.g. online reservation), however it is not the case in grid systems. Grids are dominated by hardware resources, and for this reason the leading grid middleware Globus does not involve semantic technology. Therefore, full use of semantic technology in the grid is not worthy compared to the web services applications. One way to overcome this issue, is to have semantic data models that can support both key-word based and semantic matching. For example, CPUs and memory resources can be described and discovered syntactically, while Clustering Algorithms for dataset may be described and discovered semantically.

3.6 Semantic-Based RD Systems and Emerging Grids and Clouds

In the most recent years, there has been a wide agreement that the current grids have not been able to deliver the promise of better applications and usage scenarios (Jha et al. 2009). This may be mainly due to high programming details when a user wants to describe, discover or use the resources and services. Several ongoing research projects have focussed on overcoming this matter. Some of them focus on how to provide virtualization to the grid systems in a way that the complexity can be wholly hidden from the user. These include the emergence of *meta-brokers* (Kertész and Kacsuk 2010) and (Ivan et al. 2010), new type of grids that focus on pervasiveness and the ability to self-manage which are known as *Emerging Grids* (Kurdi et al. 2008), and a new area called *Cloud Computing* (Buyya et al. 2009). For this, we describe some of the application opportunities of the semantic-based RD systems on these new systems as well as the potentials of these systems to achieve their goals through the use of semantic information.

Since meta-broker schedules the user tasks to sub-brokers in intergrid level. A key issue here is to discover the capabilities of the brokers or the other peer meta-brokers (in case of P2P meta-brokers). Therefore, using the semantic technology to describe and discover these meta-brokers and brokers may help to automate the task scheduling process, which is what the meta-broker looks for.

Emerging grids that are looking for pervasiveness, such as ad hoc, mobile and wireless grids, may not have worthy use of semantic RD system, as their devices are associated with low energy. Meanwhile those looking for manageability such as autonomic grids may benefit from the use of semantic RD systems, as they can detect suitable services for self-composition, which in turn can provide self- manageability.

Although, there is no standard definition for cloud technology, there is wide acceptance of cloud feature that computing resources (e.g. CPUs and storage) are provided as services (Zhang et al. 2010). In fact, an obvious question here is how to move the current grid infrastructures to the cloud technology. One way to do that is to treat each grid (e.g. for organization) as a service. This means we will not deal with neither a single resource capabilities description nor how to program, and use it through its middleware. Rather, we just need to describe abstract information of the overall system that can show what the system is able to provide. This information can then be federated into registries. Thereafter, we can discover these service grids and invoke them. Of course in this situation, the description is not only limited to functional capabilities but also the pre-conditions and post-conditions for invocation as in the case of web services. The use of semantic RD systems in this situation, when it exists, will allow for service grids to be described and discovered for applications.

3.7 Related Work

Several works have been conducted with regard to grid RD systems' taxonomies and surveys. In this section, we describe such works.

3.7.1 A Taxonomy of Grid Monitoring Systems

The work of (Zanikolas and Sakellariou 2005) introduces a broad taxonomy to the grid monitoring systems (GMS). The aim is to provide an advanced understanding of GMS. For this, the work classifies GMSs based on the mapping of Grid Monitoring Architecture (GMA) components (*Producer, Directory Service (Registry), Consumer, and Republisher*) that have been presented by (Tierney et al. 2002) to the grid

monitoring phases (GMP). GMP includes *generation of metadata, processing metadata, distribution of metadata, and presentation/computation of the metadata*. This mapping produces four phases, which are then leveled from zero to three, and the proposed GMSs are classified according to the levels that they fall into (see table 3.4 the summary the classified studies).

Although, the taxonomy classifies the GMSs with respect to their compliance to the core GMA components: main target of monitored entities, and the dependency to each other, it does not discuss the issues that are related to metadata modeling, selection process, and management of the overall GMS.

Table 3.3 A summary of the grid monitoring systems with their levels

Level 0	Level 1	Level 2	Level 3
<p>MapCenter (Bonnassieux et al. 2002).</p> <p>GridICE (Andreozzi et al. 2005).</p>	<p>Autopilot (Ribler et al. 1998)</p>	<p>CODE (Smith 2002).</p> <p>GridRM (Baker and Smith 2003).</p> <p>Hawkeye⁴⁶.</p> <p>HBM (Stelling et al. 1999).</p> <p>JAMM (Tierney et al. 2001).</p> <p>Mercury (Balaton and Gombás 2004).</p> <p>NetLooger (Tierney and Gunter 2003).</p> <p>NWS (Wolski et al. 1999).</p> <p>OCM-G (Balis et al. 2004).</p> <p>Remos (Dinda et al. 2001)</p> <p>SCALEA-G (Truong and Fahringer 2004)</p>	<p>Ganglia (Massie et al. 2004).</p> <p>Globus MDS (Foster and Kesselman 1997).</p> <p>MonALISA (Newman et al. 2003).</p> <p>Paradyn (Miller et al. 1995).</p> <p>RGMA (Cooke et al. 2003).</p>

3.7.2 Peer-to-Peer RD in Grids

The work of (Trunfioa et al. 2007) reviews the grid RD systems that adopt peer-to-peer RD models and protocols in their building blocks. It presents qualitative comparison of the existing approaches, describes their advantages and disadvantages,

⁴⁶ <http://www.cs.wisc.edu/condor/hawkeye/>

and finally discusses the future research directions of grid RD systems. Initially, P2P based grid RD systems are classified into two groups which are: RDs based on unstructured P2P and RDs based on structured P2P systems. Both of the classes are qualitatively compared. Based on these comparisons the study concludes that, grid RD systems based on structured P2P perform better than unstructured systems but associated with high cost of network maintenance. The work also discusses briefly grid RD systems that are based on semantic information (see table 3.5 is the summary of the discussed systems). This includes the potential of semantic technology in this filed, the current efforts on building semantic information and P2P based Grid RD systems. The work finally raises the need for studies to discuss and compare the growing grid RD based on semantic information approaches. In response to this need, this chapter mainly focused on the RD systems based on semantic information.

Table 3.4 A summary of the P2P and semantic information based grid RD studies

Grid RD based on unstructured P2P systems	Grid RD based on structured P2P systems	Grid RD based on semantic information
(Iamnitchi and Foster 2004); (Talia and Trunfio 2005); (Mastroianni et al. 2005); (Puppin et al. 2005); and (Marzolla and Mordacchini 2005).	MAAN (Cai et al. 2003); (Andrzejak and Xu 2002); SWORD (Oppenheimer et al. 2004); XenoSearch (Spence and Harris 2003); Mercury (Bharambe et al. 2004); (Schmidt and Parashar 2003); and (Ratnasamy et al. 2003)	(Li and Vuong 2005) and (Kashani et al. 2004).

3.7.3 Peer-to-Peer Based RD in Global Grids

Another study on P2P based grid RD systems has been conducted recently by (Ranjan et al. 2008). The work has focused mainly on structured P2P systems (e.g. distributed hash tables) and how they can be extended to indexing *d*-dimensional grid resource queries. Towards the end, the authors classify P2P systems based on their

supportiveness to d -dimensional query routing, review the existing work that can support d -dimensional grid resource queries, and classify the reviewed approaches based on the proposed P2P classification. The use of semantic information is out of the scope of the paper.

3.7.4 Summary

The recent years have seen a convergence between grid and semantic technology. The most concerned grid part in this convergence is the RD system. Thus, we reviewed the current RD systems that use semantic technology as information and data models for their Resource description or other aspects. First, we discussed both grid RD system and the semantic technology. We then presented a taxonomy for the RD systems, which is based on how they integrate the semantic technology. We then compared and analyzed these systems in terms of how they fulfill the grid RD requirements. We highlighted the pros and cons of each system, and what the research community in this field should be focusing on in the future. We believe this chapter can be profitably used by grid and cloud resource discovery designers, and developers as it provides some hints on how to select an appropriate semantic-based RD system.

CHAPTER 4

SEMANTIC-BASED RESOURCE DESCRIPTION MODEL

4.1 Introduction

This chapter presents the proposed RD semantic description model for the intergrid system. This appears to be essential in the RD system as it is related to the representation of the services and resources for the discovery process. For this, the chapter starts by identifying the methodology and the key components of the model that represent the foundation, which forms the basis for building the model. The foundation components include some set of definitions that are related to intergrid system modeling, the resources and services representation and the resource/service request formulation. The chapter then gives an explanation of the building block, which is about the assembly of the identified foundation components to form the new model, the available data models and tools for editing and exporting the semantic information that can be implemented in the model components, and collaboration method between the components. The chapter, thereafter, depicts the process for the construction of resources and services metadata, and the request formulation of resource/services based on the new model. Finally, the chapter presents an evaluation of the model as to how the model meets the interoperability feature, and its suitability to the intergrid system.

In short, the main contributions of this chapter can be summarized as follows:

- A proposal for a new architecture for the intergrid system based on the refinement of the latest grid system standard requirements.
- A proposal for a novel semantic representation mechanism for the resources and services of the refined intergrid system.

- A demonstration on the applicability of the semantic representation mechanism with the current grid information services without posing an overhead on the services.

4.2 Methodology

As we have described in the previous chapters, the use of semantic information on RD system description is one of the solutions to the existing problem of metadata heterogeneity at the intergrid level. However, the problem here is how to use a proper semantic technology mechanism that is useful to intergrid systems, due to high computational cost of the semantic technology. One way of addressing that is by using the semantic description only when it is needed. The second issue is the interaction between the end user and RD system in discovering the resources and services. In addressing this issue, we have identified some of the facts in the current grid technology that would be vital for improving the designing of a semantic-based RD system for intergrid system. This is based on our experience with the grid technology. We use the identified facts to refine the idea of the intergrid system in such a way that makes full use of the resources and services when the semantic technology is applied. The refinement of the intergrid is based merely on the latest standard grid system requirements. The problem of information heterogeneity is addressed by introducing ontology as an information model for the refined intergrid system. We use one common ontology that is able to formally represent the intergrid components. It should be noted here that we build neither the intergrid domain ontology nor the tools of the information manipulation, as they are out of the scope of the present work. Instead, we define some set of definitions that formalize the essential requirements and guidelines that can be followed to build the respective ontology, and for selecting the information manipulation tools. To address the issue of abstraction, we introduce the semantic query formulation by treating every application as a goal, which can be formally described and made reusable. We illustrate the standard ontology languages and the editing tools for the model implementation to ensure the applicability of the model. The model is evaluated qualitatively by

examining how the model meets interoperability as it is the requirement that is very much related to the description component in the RD system.

4.3 Foundation of the Semantic Description Model

A description model for an RD system should be based on a deep consideration of everything related to the RD system, either directly or indirectly. This is to ensure that the system can effectively work with related entities in the entire system. To that end, we present some observations from our experience with the grid technology, which could be useful for consideration in our new description model.

First, existing grid systems are associated with many redundant developed applications. For example, a grid user may develop a parallel clustering algorithm to cluster some datasets. In this case, the required resources are CPUs and memories, where the clustering will take place, and the dataset. If the dataset is not available in his/her side, there is nothing to do with finding clustering algorithm. It may happen that another user from a different grid level system wants to do the same clustering application, which requires the development of the application from the beginning since the clustering algorithm is not sharable. Obviously, this is an application redundancy, which wastes time and efforts. In contrast, if there is a mechanism that allows the reuse of such applications at the intergrid level, the redundancy can be overcome.

Secondly, most of the existing grid projects have been focused on a particular type of grid based on the nature of the emphasized perspective, which we have mentioned in chapter two. For example, data grid project where consumers can access pooled data and store their data into distributed storage. Normally, such systems provide some tools for easy access and hide the complexity from their users, and so on. In addition to that, these kinds of systems work fine at their scope level. A need for resources may rise in these systems only in two situations. First, when they need the same type of resources, but beyond their scope. For example, a bioinformatics research center has data grid that provides bioinformatics data only; then one of the user wants to access some medical imaging data which is a data resource but out of

the scope of that bioinformatics data grid. The second situation is when the need of resources is out of the focus of the current grid. Back to the bioinformatics data grid project as an example, for instance a user wants to run a large application that requires a huge amount of computing power but it is not supported by the current bioinformatics data grid system.

Third, the current intergrid level architecture that is supported by broker and meta-broker are mainly focused on computing resources and data resources. For example, in the case of broker, the broker is provided with the information about the computing/data resources of different level grid nodes, and the broker will have the ability to assign the jobs to these resources. In the case of meta-broker, the meta-broker is provided with information about the brokers and the meta-broker then assigns the task based on their capabilities. In either situations, grid level systems are provided with the facility of having computing/data resources beyond their scope, or those with computing resources can access data resources and vice versa. However, the sharing of resources beyond the computing and data resource is not supported. This implies that the redundancy of the developed applications still exists.

From these observations, it is obvious that there is a need to share resources beyond computing and data resources such as application software, but the current architecture does not support that entirely. Therefore, a solution is needed to make the intergrid level system able to share beyond these resources. In the next subsection, we will refine the idea of the intergrid level in order to make the sharing of resources possible beyond the current implementation.

4.3.1 Intergrid Services

We merely rely on the latest grid system requirements that have been presented by the OGF⁴⁷ (Subramaniam et al. 2009) in defining the grid level and intergrid level. As a result, we treat grid level system as a *service grid* that is provided by a *provider* to *consumers*, and this service grid is assumed to be among the grid types (e.g.

⁴⁷ Open Grid Forum: <http://www.ogf.org/>

computing, data and application) that we have emphasized in chapter two. For this, the three new terms (service grid, provider and consumer) are formally defined as follows:

Definition 4.1: A service grid is a software/set of softwares that abstracts the entire grid level system and provides some functionality that is based on the focus of the system such as data, application service, interaction, knowledge, and utility to consumers.

Definition 4.2: A provider is an entity who provides service grid to consumers. A provider may provide one or more service grids such as data service, application service, interaction service, knowledge service, and utility service.

Definition 4.3: A consumer is an entity who makes use of one or more of the service grids upon some agreement with the providers.

Definition 4.4: An intergrid level system is a collection of service grids that have agreed to work cooperatively as consumers and providers. Consequently, a service grid may be a consumer as well as a provider. It becomes a consumer when it uses other service grids without providing any service to them, whereas it functions as a consumer/provider when other services are added to its own and at the same time it also provides a complete service to the end user (see Figure 4.1).

Initially, intergrid level systems are distinguished by the use of multi-middlewares in their grid level systems. The above definitions ensure that the use of multi-middleware still exists. Thereupon, each service grid has its own local information service that describes its internal components, which incorporate both resources and services. The local information service contents represent the overall capabilities of a given service grid. In the rest of this thesis, we mainly replace the term “resource and service” with “service grid” since we treat the grid level system as service grid. However, this does not affect the overall system name which is resource discovery system (RD). We also mainly replace the “intergrid level” system with “intergrid service” to be consistent with definition 4.4.

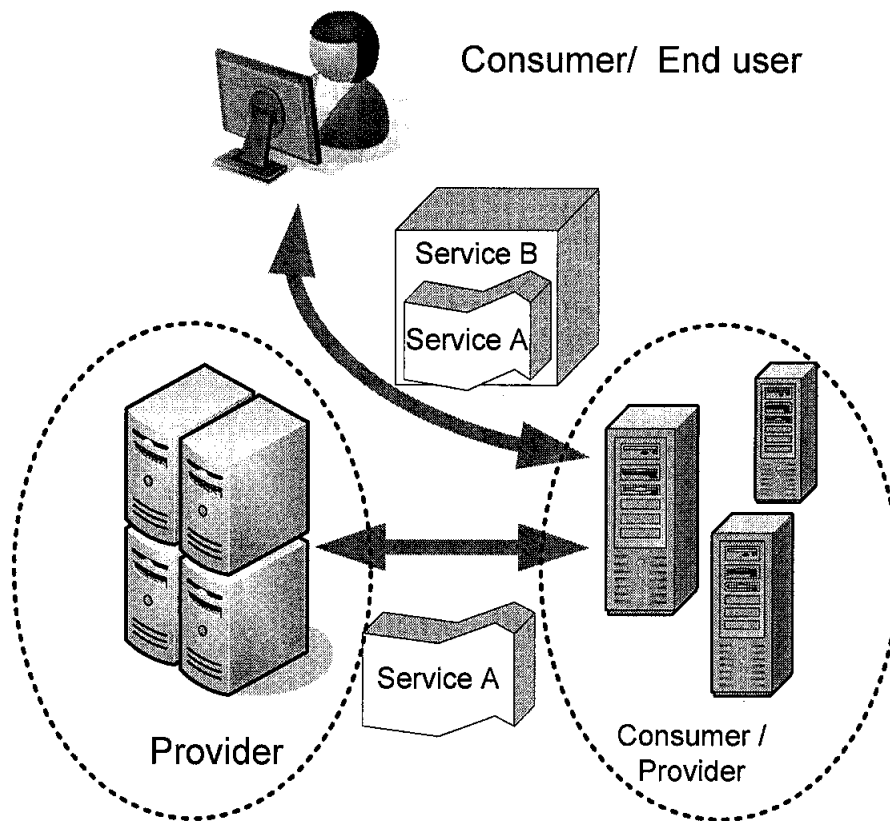


Figure 4.1 The relationship between providers and consumers at intergrid level

In order for service grids to exchange their services with each other, they need a description mechanism that will allow them to reveal their capabilities for discovery. This description must be based on information model that is able to represent service grids, their properties and the inter-relationship between the service grids, and a data model that is acceptable to every service grid. The normal way to have such a description mechanism is to integrate all internal local information services of the service grids into a common metadata pool, which is based on a common information and data model. This is quite costly as we need space in the metadata pool(s) that may be as much as the sum of the spaces of each service grid level in local metadata space. Therefore, we introduce a new scenario for integrating the information services by aggregating the local metadata content and then integrating them into a common information model.

4.3.2 Service Grid Information Aggregation Mechanism

Each service grid uses any other service grid only when it needs services that are either of the same type of its service, or service(s) that is different from its service. The internal management services (e.g. scheduling), that are provided by the middleware of each service grid, is strong enough to get service request and provides the service as it does with its local end users. For that reason, the only information needed here is the one that tells the capabilities of a given service grid so that we can invoke it through its defined invocation method. Accordingly, the entire internal information of each service grid components has no importance to be integrated and known.

Capabilities of service grid can be either quantitative or qualitative. Quantitative capabilities are the features of service components that can be measured; for example, the number of computing elements in computing service grid. Further, the quantitative capabilities can either be dynamic or static. Static capabilities are those that stay for a period of time unless the service components have been changed (e.g. the maximum number of CPUs), whereas dynamic quantitative capabilities are those that change from time to time due to local usage circumstances (e.g. the actual load of the CPUs at a given time). Qualitative capabilities is related to features of the service components that can make the component identifiable in terms of what it offers, for example a domain name of a given data, and the type of events that a given simulation tool can support.

Our aggregation scenario here is to sum up the static quantitative capabilities of the service components to maximum and minimum, and the dynamic capabilities to the total of the current status. Meanwhile, qualitative capabilities can be abstracted from the most important feature that is important to differentiate the overall service from the others. For instance, operating system, scheduling mechanism, CPU model, CPU vendor of a service grid can be abstracted from the name of the service middleware.

Definition 4.5 Let S_G be a service grid, R, S be the sets of the local resources (hardware or software) and services respectively, hence, $r_i \in R, s_j \in S$. Let c denotes

capabilities of a given r_i or s_j . The overall capability of the service grid C_{sg} can be represented as:

$$C_{sg} = \{C_R, C_S\}, \forall c_r \in C_R \text{ and } \forall c_s \in C_S \quad (4.1)$$

Where C_R , C_S are the aggregation and abstraction of c_r , and c_s .

4.3.3 Service Grid Information Representation

The service grid metadata aggregation mechanism assists us to highlight the capabilities of the service grids. To formalize these capabilities and make it reusable for every appearing service grid, there is a need for information model. In fact, we have discussed in the previous chapter that using common ontology is much closer toward achieving interoperability compared to using different ontologies. Therefore, we introduce the use of common ontology that can be used as an intergrid service grids information model. We call this ontology as *service grid domain ontology*. Service grid domain ontology defines all the service grid types, the attributes that are needed for each service grid, the relationships between all the services, the structure of the values of each attributes and so on.

Figure 4.2 shows a fragment of example of service grid domain ontology. The rounded rectangles represent the concepts and the arrows represent the relation (subclass of) between the concepts. The dashed arrows denote the continuation of the subconcepts and their relations. The super concept is the intergrid system. The intergrid system treats every grid level system as service, hence the subconcept of intergrid is service. The subconcepts of service are the six grid types (computing, data, application, interaction, knowledge, and utility) based on the current grid technology. The subconcepts and properties under each of the six services concepts represent the detailed subservices of each service concept. For example, a file system is the property of data concept and data is service grid. File can also be a concept for other properties. Based on the above details, we can formally define our ontology as follows:

Definition 4.6 Let S_{GO} denotes service grid domain ontology. Then S_{GO} consists of three entities: set of *concepts* (C), *properties* (P), and *relationship* between those concepts (R).

$$\therefore S_{GO} = \{C, R, P\} \tag{4.2}$$

Where C is the set of service grid concepts, P is the set of properties for the concepts, and R is the relations between the service grids which can produce the concept hierarchy.

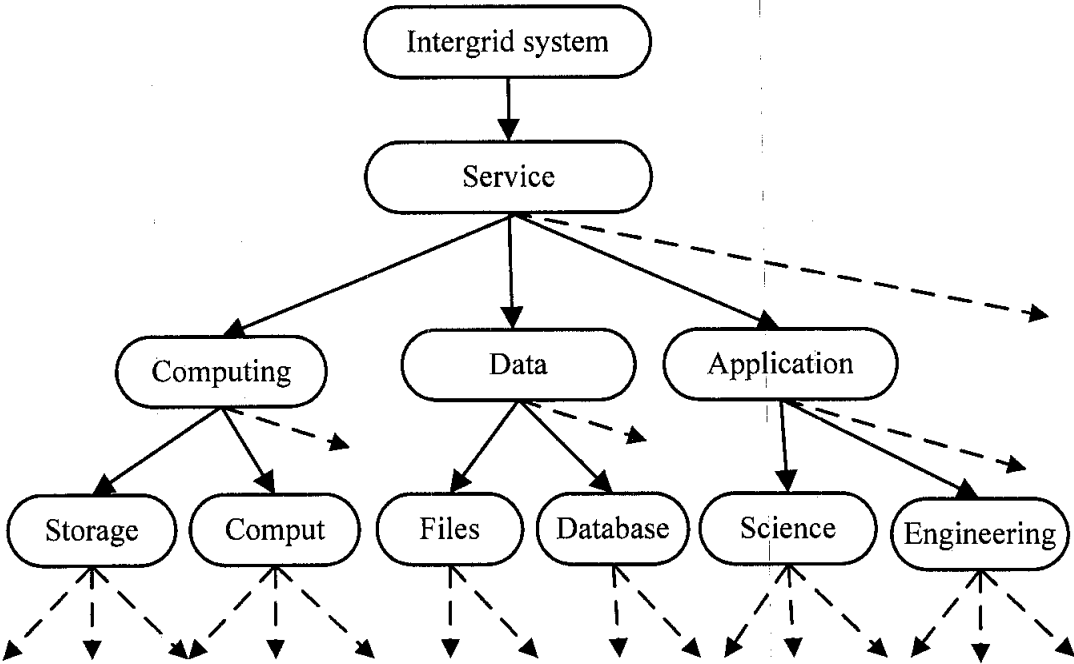


Figure 4.2 Fragment of service grid domain ontology

In order to ensure that S_{GO} can effectively serve as information model for service grids, it must meet two requirements, which are completeness and expressiveness.

Definition 4.7 S_{GO} must be *Complete* and *Expressive*. Complete means all the given intergrid services are covered. Meanwhile, expressive refers to the situation where S_{GO} is semantically perfect with no ambiguity on its terms and notations of the services. This indicates that S_{GO} is accepted not based on its interconnectivity, but on the meaning of the contents.

Once definition 4.7 is met we can have the following definition regarding the concepts relations.

Definition 4.8 Let C_i and C_j be sets of concepts within S_{GO} , where $c_i \in C_i$ and $c_j \in C_j$, thus $C_i, C_j \in S_{GO}$, δ be a denotation that is given to a concept c , β denotes c_i and c_j that are synonyms, and ρ be the set of properties of a given concept, either c_i or c_j .

$$\therefore \exists (c_i, c_j) \left((c_i \approx c_j) \rightarrow \left((c_{i\delta} = c_{j\delta}) \vee (\beta) \vee (c_{i\rho} = c_{j\rho}) \right) \right) \quad (4.3)$$

This means that two different concepts (c_i, c_j) are semantically equivalent only when they have the same denotation names, are synonyms or their properties are same.

Definition 4.9 Let C_i and C_j be sets of concepts within S_{GO} , where $c_i \in C_i$ and $c_j \in C_j$, thus $C_i, C_j \in S_{GO}$.

$$\therefore \exists (c_i \in C_i, c_j \in C_j) \left((c_i \approx c_j) \rightarrow (C_i \approx C_j) \right) \quad (4.4)$$

This means that two sets of concepts (C_i, C_j) are semantically equivalent when there are at least two subconcepts that are approximately equivalent.

Definition 4.10 Let c_i and c_j be different concepts within S_{GO} , and ρ be the property set of a given concept c_i or c_j .

$$\therefore \exists (c_i, c_j) \left((c_i \subseteq c_j) \vee (c_j \supseteq c_i) \right) \rightarrow ((c_{i\rho} \subseteq c_{j\rho}) \vee (c_{j\rho} \supseteq c_{i\rho})) \quad (4.5)$$

This means that the two concepts (c_i, c_j) are semantically inclusive when there is at least one property set of a concept that is a subset of property set of another concept, OR when there is at least one property set of concept that is a superset of property set of another concept.

Definition 4.11 Let C_i and C_j be sets of concepts within S_{GO} , where $c_i \in C_i$ and $c_j \in C_j$, thus $C_i, C_j \in S_{GO}$.

$$\therefore \exists (c_i \in C_i, c_j \in C_j) \left((C_i \subseteq C_j) \vee (C_j \supseteq C_i) \right) \rightarrow ((c_i \subseteq c_j) \vee (c_j \supseteq c_i)) \quad (4.6)$$

This means that the two sets of concepts (C_i, C_j) are semantically inclusive when there is at least one subconcept of a set of concepts that is a subset of the subconcept of another set of concepts, OR when there is at least one subconcept of a set of concepts that is a superset of the subconcept of another set of concepts.

4.3.4 Service Grid Request Formulation

As we have described above that a mechanism to reduce the user interaction with programming details in using the grid technology is highly needed. From the RD perspective, it would be more useful to reduce the user interaction with the RD system in building the service grid requests. For this, we introduce a kind of mechanism which is called Goal-based Service Grid Request Description (GSGR). A goal is “something you want to do successfully in the future” according to the Cambridge dictionary⁴⁸. In the context of the service grids here, a goal is referring to what a given consumer/end user wants to achieve by using the service grids. For example, if a user wants to simulate the weather condition of the earth so the simulation is his/her goal. Obviously, a goal requires a set of the grid services in order to be accomplished. For example, the simulation of weather condition of the earth requires computing service grid, satellite images data and temperature dataset which can be under the data service grid and so on. To accurately describe goals we define some of the requirements that should be considered in the definition of goal:

- Goals should describe the required service grids as clear as possible so that service grids can be reasoned.
- Goals must have two parts, which are information model (schema) and the instance of the schema. The former is a generic goal format that describes a general class of service grids, whereas the latter is instantiated from the goal format with its concrete information.

⁴⁸ <http://dictionaries.cambridge.org>

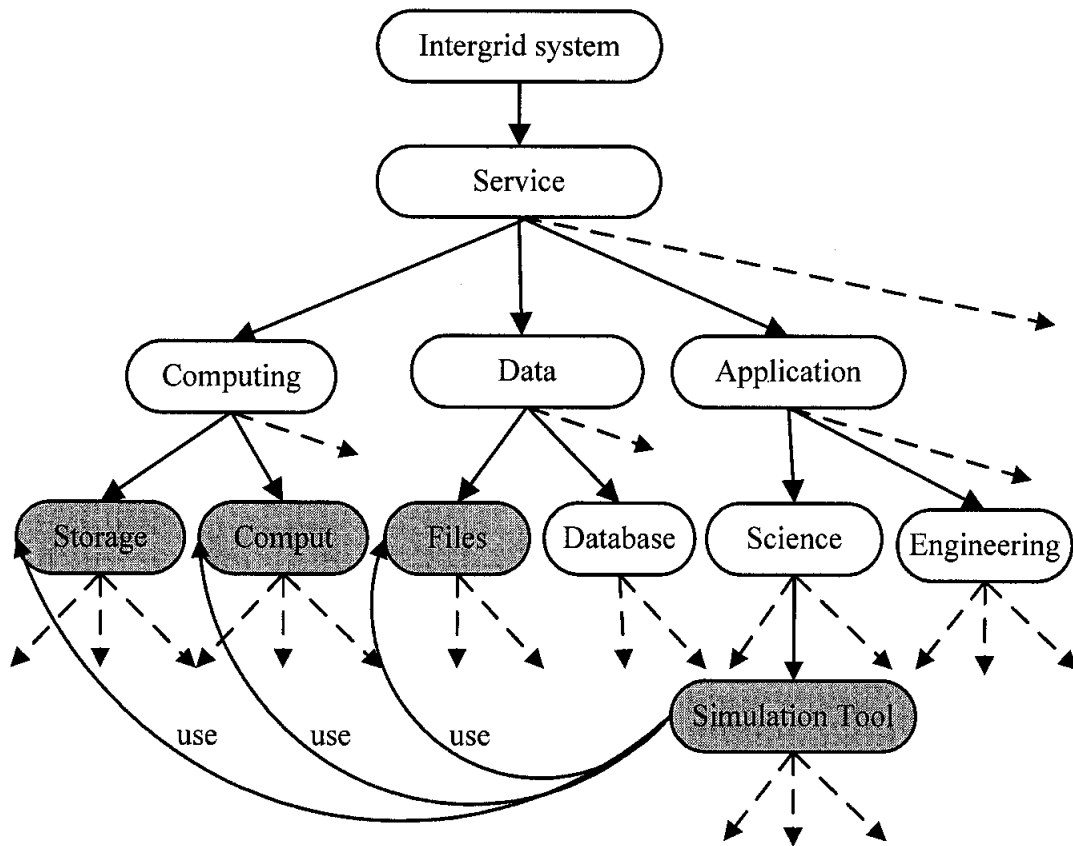


Figure 4.3 The extraction of application goals from the service grid domain ontology

In brief, goals should be expressive, allow less human intervention and provide format and instances. The service grid domain ontology, among other concepts of application service grid, includes all software applications that are available on the intergrid level system. In fact, these applications represent the goals that a user may want to achieve because application service grids are the only services that need one or more service grids to work on, as they cannot stand alone. A goal that is not covered in a given service grid domain ontology may be added into the service application concepts. Thus, we can extract the goals from the service grid domain ontology. However, this needs a definition of a clear relation between the concepts of the application service grids and the concepts of the other service grids. For this, we introduce a relation so called “*use*” between the application service concepts and the other service grid concepts. The “*use*” relation is a binary relation between a particular application service concept and another service grid (e.g. data service). Denoting this application service requires the second service grid with which the

relation is established. Therefore, we can formally define the goals as follows:

Definition 4.12 Let $G = (V, E)$ be a graph, then $V = \{v_1, v_2, \dots, v_i, \dots, v_n\}$, $E = \{e_1, e_2, \dots, e_i, \dots, e_x\}$, where V represents the set of nodes and E the set of connections between the nodes. We can treat our service grid domain ontology as graph $S_{GO} = (C, R)$ where $C = \{c_1, c_2, \dots, c_i, \dots, c_n\}$ as the set of concepts and $R = \{r_1, r_2, \dots, r_i, \dots, r_x\}$ as the relations between the concepts. Let $U = \{u_1, \dots, u_i, \dots, u_n\}$ be a set of the “use” relation between the concepts $U \in R$ and φ denotes a concept c as the goal.

$$\therefore \exists c_i (c_i = \varphi) \rightarrow (\exists (f: c_i \rightarrow \exists c_j) \in U) \quad (4.7)$$

This means that a concept is a goal if there is at least one “use” relation between two concepts.

Based on definition 4.12, we can derive the overall goals with their respective required service grids in a given service grid domain ontology. We call the overall set of goals with their required service grids as the Goal Template Matrix.

Definition 4.13 Let $A = [a_1, a_2, \dots, a_i, \dots, a_x]$ be a set of goals in $S_{GO} = (C, R)$, where $A \in C$ and $\bar{A} = C - A = \{\bar{a}_1, \bar{a}_2, \dots, \bar{a}_i, \dots, \bar{a}_k\}$ represents the set of concepts that are not goals. Let ω denotes the Goal Template Matrix. Using the adjacency matrix of the graph then:

$$\therefore \omega = A \times \bar{A} = \begin{pmatrix} a_1 \bar{a}_1 & \dots & a_1 \bar{a}_k \\ \vdots & a_i \bar{a}_j & \vdots \\ a_x \bar{a}_1 & \dots & a_x \bar{a}_k \end{pmatrix} \quad (4.8)$$

Where the elements of ω , $a_i \bar{a}_j \in [0, 1]$, $a_i \bar{a}_j = 1$, when a goal a_i requires \bar{a}_j service grid and 0 if otherwise.

Figure 4.3 illustrates the use relation between a simulation tool application service and three other service concepts, namely storage, compute (computing power), and files. The arrows represent the use relationship. Simulation tool is a descendent concept of the application service grid and the other services are descendents of their respective super concepts. In fact, any simulation tool requires a data entry, a

computing platform, and storage to store the result, and probably analysis software to study the results. Therefore, this relation can be extracted to describe the goal format of simulation. However, the simulation goal format at this stage is too general, so during the service request process the format will be instantiated with concrete values.

4.3.5 Service Grid Information Manipulation

As soon as service grids are semantically described in the ontology, their request can be formulated through the goals. There is a need for a method that is able to process the semantic information when a concrete goal instance is introduced against the available service grid information. The method is known as similarity function. Several similarity functions have been proposed (Resnik 1999) and (Rodr et al. 2003) and embedded in the ontology query languages, but for the purpose of giving an insight into the idea of similarity in the ontology in the context of this thesis, we formally define the similarity function as follows:

Definition 4.14 A similarity function is a real valued function that computes the similarity degree between two concepts based on their properties.

$$sim(a, b): C \times C \rightarrow [0 - 1] \quad (4.9)$$

Where a and b are concepts, the value $sim(a, b)$ ranges between 0 and 1; $sim(a, b) = 1$ means that they have exactly the same properties; $sim(a, b) = 0$ means that there are no common properties between the concepts. We compute the similarity using the Dice distance fraction as follows:

$$Sim(a, b) = \frac{2|a \cap b|}{|a| + |b|} \quad (4.10)$$

Where $(a \cap b)$ is the set of common properties of the concepts, and $|a| + |b|$ is the sum of property sizes of the two concepts.

It should be noted that in the rest of the thesis, we use the term " $Sim(variable_1, variable_2)$ " to refer to equation 4.10.

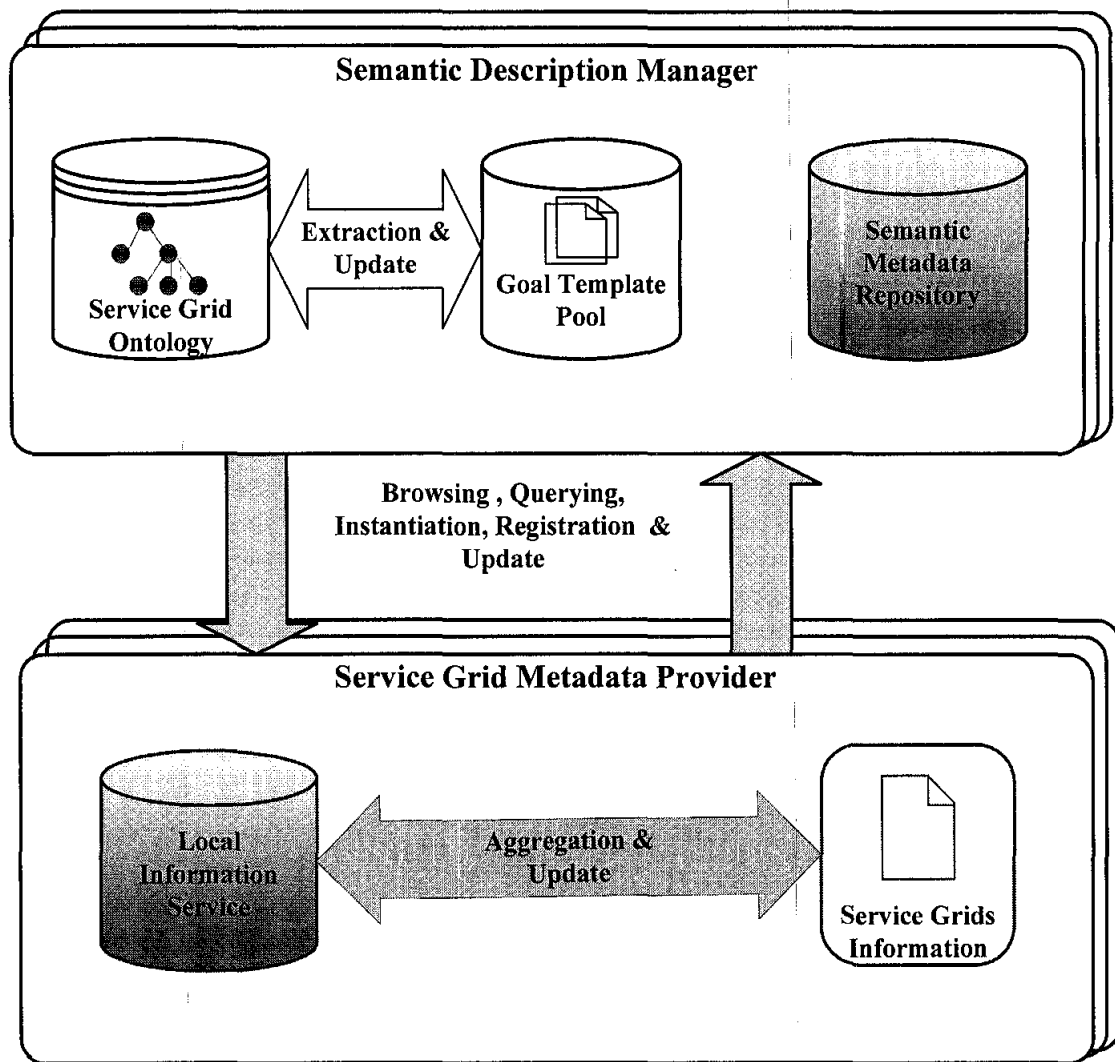


Figure 4.4 The description of model building block

4.4 The Description of Model Building Block

Having described the fundamental components of the description model, in this section we illustrate the building block of the description model. Figure 4.4 shows the components of the model and their related subcomponents. The model is initially composed of Semantic Description Manager (SDM) and Service Grid Metadata Provider (SGMP). SDM generally is responsible for the global service grid description in the intergrid system, and a pool that can be used to accommodate the service grid metadata coming from SGMPs. Meanwhile, SGMP is responsible for managing local service grid metadata that belongs to a service grid provider. The reason for having the SDM and SGMP in such architecture is that, SDM will provide

all the needed information and data model management for a set of intergrid members. Therefore, interoperability can be ensured. In the meantime, SGMP provides autonomy for each service grid member as it concerns about the local information of the service grid.

4.4.1 Semantic Description Manager (SDM)

SDM consists of Service Grid Ontology, Goal Template Pool and Semantic Metadata Repository. Service Grid Ontology is where the service grid domain ontology is accommodated. SGO is supposed to provide browsing and querying tools for viewing contents of the ontology. SGO should allow the addition of new concepts, properties and relations to the ontology in order to provide adaptability with time change. The reason behind having the service grid ontology in such position is for easy maintenance of ontology, without involving all the participants in the intergrid system since there may be only some dedicated nodes that are responsible for this management. SGO can be implemented by using some of the available tools, which provide the functionalities that SGO looks for. First, these tools include ontology language to encode the ontology and browsing, and the second is an extraction and manipulation tool. The Web Ontology Language⁴⁹ (OWL) can be used as an ontology language for service grid ontology since OWL is the standard that has been recommended by W3C⁵⁰ for semantic web, and is suitable for semantic data storing. Protégé⁵¹ is an ontology editing, browsing and exporting tool, which is very relevant to SGO.

Goal Template Pool (GTP) stores the application services with their required services in terms of goals. This is done by extracting the information from the application services and the other services that have the “use” relations between them (the goal template matrix). GTP should provide a browsing and instantiation functions, so that the consumers can understand the available templates and get instances of these templates for use in their query formulation. In addition to that, goal

⁴⁹ <http://www.w3.org/TR/owl-features/>

⁵⁰ <http://www.w3.org/>

⁵¹ <http://protege.stanford.edu/>

template pool is updated timely when there are new concepts or relations added to the service grid domain ontology. The reason for separating the goals from the SGO is that we will have two kinds of interests among the participants of the intergrid system, which are the provider and the consumer. Providers are very interested on knowing how to describe their services, whereas the consumers are very concerned on how to prepare a proper service request. For that, the consumers do not need to know about the other services concepts or navigate the whole service grid domain ontology.

Semantic Metadata Pool contains the actual semantic information about a given set of service grids. The information is based on the service grid domain ontology. In such situation, the service grid domain ontology is populated with some actual values of the semantic service grids and stored in a repository that belongs to the semantic metadata pool. Semantic metadata pool provides a query tool to solve queries or requests that come to the semantic metadata pool. It also accepts registration of the service grid metadata providers according to prior agreements. The Sesame⁵² open source framework for storage, inferencing and querying of RDF data is a very useful tool to be used in the semantic metadata pool.

In order to support decentralization in the registry, we may have a set of semantic description managers that can work in a distributed manner. Where each SDM can be responsible for managing some parts of the service grid domain ontology and some set of service grid metadata providers. When there is an update in the service grid domain ontology, the SDMs can notify each other about the update.

4.4.2 Service Grid Metadata Provider (SGMP)

SGMP consists of local information service and service grid information. Local information service is the actual information service that are embedded in any given grid middleware such as the Globus Monitoring and Discovery Service⁵³, which are supposed to manage the service grid internal information, initially. Local information service also interacts with SDM to get the necessary service grid attributes/properties

⁵² <http://www.openrdf.org/>

⁵³ <http://www.globus.org/toolkit/mds/>

from the service grid ontology, and creates a service grid information file that has the actual values of the service grid attributes of a given provider. Upon that, the local information service registers its service grid into the SMR in SDM, and keeps a link between the service grid information file and semantic service grid information in the SMR. An obvious question here is the type of data model that belongs to the service grid information file. It is possible to have a semantic file for service grid information same as the SDM with values of the service grid information of different properties. However, we do not have many elements in the service grid information that should be represented in a subontology in the service grid information file. Therefore, a normal XML file that is based on the XML schema can be used to accommodate the service grid information. Once the description of the service grid information in an XML file is done, we need to ensure that communication between the SMR and the XML file in terms of data exchange can take place. The data exchange includes the registration of service grid information and the service query that may come from SMR. Thus, we need a tool that has the ability to wrap the XML information to the semantic information in the SMR during the registration, and to wrap the semantic query to XML query during a service request. The SAWSDL model (Jacek et al. 2007) that we have discussed in chapter 3 provides such facilities. SAWSDL has, in its schema mapping component, two attributes for attaching schema mappings:

- *sawSDL: liftingSchemaMapping*, and
- *sawSDL: loweringSchemaMapping*.

Lifting mappings transforms XML data into a semantic model, and lowering mappings transforms data from a semantic model into an XML message. The implementation of the SAWSDL in our model is illustrated in Figure 4.5. The SMR leaves the XML data containing the service grid information, in order to be stored as semantic information. SMR lowers the semantic request into an XML format when there is a request for the service grid information.

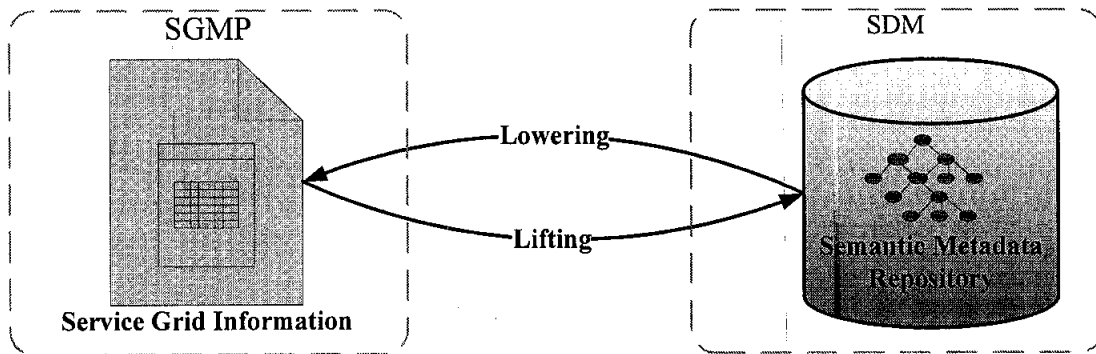


Figure 4.5 The data exchange between the service grid information and the semantic metadata repository

4.5 The Description Process

The description process includes the description of a service that will be advertised, and the service request formulation. The steps of the first case are as follows:

- The user invokes the Service Metadata Provider system.
- The user browses/queries the service grid ontology through which all the available content of the ontology can be manipulated (e.g. using add and drag menu), and selects the concept that is relevant to his/her service grids.
- The user gets an instance of the selected service grid concept, which is in the form of XML elements that is based on XML schema, using the service grid ontology tools.
- The user can adjust the XML file in a way that incorporates all of the most needed attributes with which the service can be described, and populates the file with the actual service grid information, which is an aggregated summary of the overall service grid information.
- Finally, the user sends the service grids information to the respective SMR of the SDM node that is responsible of holding the metadata of the current service provider, and in turn, the SMR stores the semantic information about the service grid for the discovery process.

Meanwhile for the service request formulation:

- The user invokes the Semantic Description Manager system.
- The user browses/queries the available goals in the goal template manager, and selects the relevant goal.
- The user then gets an instance of the selected goal and adds it to his/her local information system.
- Since each goal requires one or a set of service grids, the user adds the concrete values of attributes of each service. For example, if among the required service is computing service, and one of the attributes of the service is *maximum number of computing nodes*, the user may add a concrete number for such requirements.
- Finally, the user sends the service grid request to the respective SDM, and the SMR of that SDM will generate a proper query statement for each service among the required service grids.

4.6 Evaluation

The first aim of the description model is to provide some interoperability in the intergrid system. In this section, we will discuss how the model meets this feature and other features that are convenient to the intergrid system, such as applicability with middleware information service, reducing the cost of using semantic information, and so on.

From the building block, it is clear that the model has introduced the use of semantic information in way that does not require the use of local information service, which exists currently in grid middleware. For example, the intergrid participants (small grids) are able to use their local discovery system that would normally be possible through a keyword-based RD system. They just need to have one file that accommodates the summary of the overall capabilities of the service. As a result, this provides interoperability among the participants in the intergrid system. The model also reduces the cost of using semantic information in terms of processing time, as

well as storage of the semantic information, since the semantic information is used at the intergrid level, and not at the grid level. Therefore, during the discovery we look up for a complete service grid, not just for components of the service grid. This is in contrast to the other semantic-based RD studies (Said and Kojima 2009) and (Xing et al. 2010) that use the semantic representation for each component of the service grid, which results in the discovery at the service grid components level.

The model also provides useful application reusability through the use of semantic service grid representation and goal-based service request formulation scenario. Grid application developers can share their developed applications in a systematic manner as each application can be described formally, as well as the service that are needed to execute the applications.

Lastly, the model also provides standardization for the model components that are being suggested for the implementations. For example, based on the recommendation by OGF¹¹ the model introduces OWL⁵⁴ as an ontology language for the ontology and XML technology⁵⁵ service grid information, and SAWSDL (Jacek et al. 2007) for data exchange as recommended by W3C⁵⁶.

⁵⁴ <http://www.w3.org/TR/owl-features/>

⁵⁵ <http://www.w3.org/standards/xml/>

⁵⁶ <http://www.w3.org/>

CHAPTER 5

SEMANTIC REGISTRATION AND DISCOVERY MODEL

5.1 Introduction

In the previous chapter, we have presented the semantic description model for the service grids. Particularly, we have shown how the service grid can be represented and stored as metadata information. In this chapter, we will propose a new registration and discovery model for our RD system. The model addresses where the service grid metadata should be stored and how a service grid request/query can be sent to the stored metadata, and the selection of relevant service grid that meets the request specification. For this, the chapter starts with the methodology that is used to design the model. The chapter highlights the main components that form the model which are ontology and intelligent agents. The chapter thereafter focuses on the depiction of the model which contains the registry architecture, fault tolerance and load balancing strategy, and the discovery algorithm. An application of the proposed model to demonstrate how the discovery algorithm works with the remaining components (description and registration) is also included. The chapter is ended by presenting the complexity of the new RD system and a discussion on how the RD meets the identified intergrid RD requirements.

The main contributions of this chapter can be summarized as follows:

- A new semantic registry model that uses ontology to organize the service grids provider nodes and their metadata.
- Fault tolerance and load balancing scenarios for the proposed registry.
- An agent-based algorithm for the search and selection of the service grids.

5.2 Methodology

Registration and discovery components in any RD system are very much related, as the routing of request is subjected to the registration architecture. For this reason, we address the issues in registration and discovery jointly. We design a model for the two components that integrates super-peer architecture, ontology and intelligent agent.

Super-peer is used to grant distribution of the registry to where the service grid metadata is located. Namely, service grid provider nodes will be organized in sets of classes, and each class will have a head which will be the registry for the class.

Ontology, on the other hand, is to manage the distribution of the registries. More specifically, ontology defines the criteria under which the classes of the service grid provider nodes can be formed, and how to control the number of classes and the elements in each class.

Lastly, intelligent agent is used to deal with the dynamism of the service grid provider nodes status in their respective registries, and to abstract the discovery process from the end user. Especially, we define two agents to achieve that. The first one mainly focuses on the status of the service grid provider node metadata, and the second agent deals with the discovery abstraction with the help of some algorithms.

5.3 The Model Components

We have concluded in the previous chapters (2 & 3) that distribution of the registration is the ideal method in the intergrid RD system. However, the distribution should be systematic and dynamic, whereby the RD system can evolve with the scale of the intergrid system. Considering this, our model adopts the distribution method with systematic mechanism to manage the distribution. The model consists of three components, a domain ontology to organize the service grids provider nodes, an intelligent agent model to abstract the user interaction with the grid and cope with service metadata dynamism, and super-peer architecture to organize the service node providers. In the rest of this chapter, we use the term “node” to mean service grid provider node.

5.3.1 The Dictionary Ontology

We have stated in the previous chapter that ontology provides a formal representation to intergrid services; and to explore more on the use of ontology in addressing the RD problem, we have presented the idea of goal-based service request model. In the same line, there is another potential that ontology may provide for the intergrid system RD registration. This potential is the taxonomy of the ontology concepts. In the taxonomy of the concepts, the concepts are arranged hierarchically. Therefore, whenever we visit the concepts from the root concept, as we go down deeper into the subconcepts we will move from a more general class of concepts to a more specific class of concepts, and vice versa. We use this feature to classify nodes into several classes, which produce registry architecture to the RD system. The ontology that supplies service grid taxonomies is called Dictionary Ontology (DO).

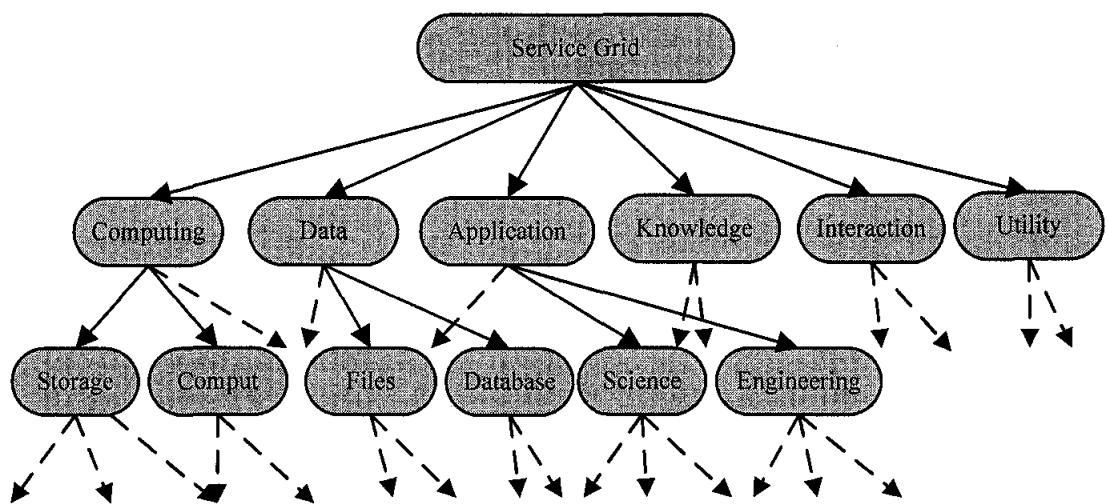


Figure 5.1 A fragment of the Dictionary Ontology

Figure 5.1 illustrates an example of a fragment of the DO, where the root concept is the service grid and the super-concepts are the children i.e. the six types of service grids that have been classified according to grid types from the nature of the emphasis perspective. Each of these super concepts will have its subconcepts that can be broken down further to generate several tiers of subconcepts until very specific service grid concepts have been defined. The DO may be the same as the service grid domain ontology by omitting the relations that are out of the hierarchical relation such

as the “use” relationship.

5.3.2 Intelligent Agent

Intelligent agents have the ability to react and work on behalf of the end user in dynamic environments such as the intergrid system. For this, we propose two intelligent agents for this model namely *Description Agent* and *Request Agent*.

Definition 5.1. *Description Agent* (DA) is a static agent that carries some information; automatically performs some set of functions and belongs to a service grid provider node.

The information carried by DA is the one that is needed for communication between the nodes. The DA functions are: describes service grid capabilities using the service grid domain ontology, informs its respective registry about its service status as well as updating them when there is a change on the service grid information.

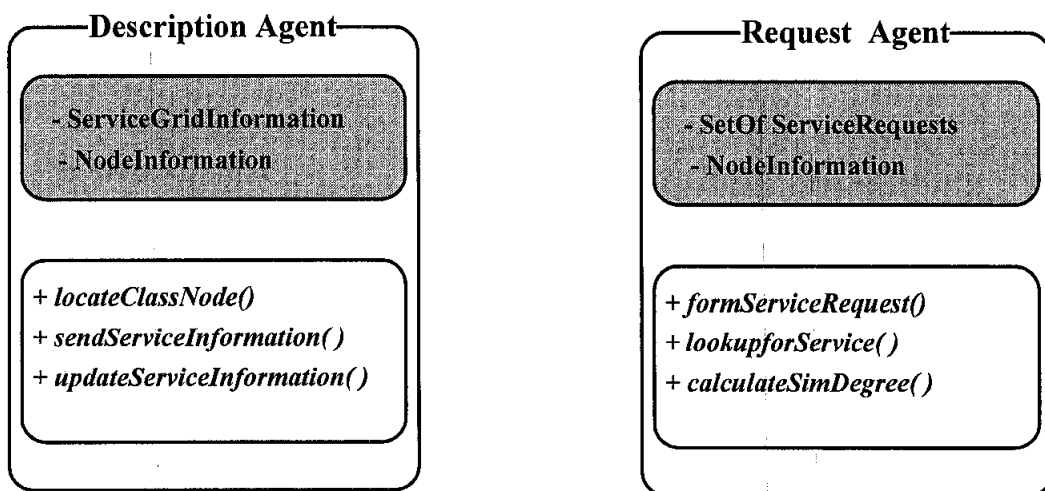


Figure 5.2 The proposed DA and RA agents

Definition 5.2 *Request Agent* (RA) is a mobile agent that carries some information; automatically performs some set of functions and belongs to a service grid node.

RA information consists of service grid request and node information. RA forms service requests, and acts on behalf of the end user by using goal template; RA then

roams the network to find the node that owns the requested service. Figure 5.2 shows a class diagram of DA and RA.

The reason for using DA and RA is that DA is adaptive to the dynamic nature of the intergrid. In this context, when a node changes its service status, the registry node of that node should be aware of that change, which in turn optimizes the request routing. While RA helps the user to formulate his/her service requests. Therefore, implementing the two agents will guarantee some abstraction on the RD system invocation.

5.4 The Model Description

The registration and discovery model consists of three elements registry architecture, fault tolerance and load balancing strategy, and discovery algorithm. In this section we discuss all these elements.

5.4.1 The Registry Architecture

Registry architecture is about the organization of the classes of nodes and the management of these classes. The registry architecture includes node class formulation, head appointment, node subscription.

5.4.1.1 Class Formulation

To support scalability and dynamism in intergrid environment, we model an intergrid system that contains some nodes to class based organization in which nodes are gathered together in a set of classes. This classification is based on the hierarchal relations among the service grids in the DO, which means their defined semantic relation on the DO. For example, nodes that provide service grids that belong to the computing concept in the DO can form a class of nodes called computing class. The computing class itself can be split into two classes, one that contains all the nodes that provide service grids under the compute subconcept, and another that contains all

nodes that provide the service grids under the storage subconcept. We systematize the formulation of the classes with a dedicated algorithm as illustrated in Figure 5.3.

```

Input : concept  $t \in T = \{t_1, t_2, t_i, \dots, t_t\}$ , node  $n \in N$ ,
Threshold;
/* where N is set of the nodes & T is list of
the concepts that is selected from DO */

Output: listOfClassNodes c;
/* a list of class nodes that related to a
concept*/

Step 1:

FOR ( $\forall n \in N$ ) DO
 $Sim(n, t) = (2|n \cap t|) / (|n| + |T|)$ 
IF  $Sim(n, t) > \text{Threshold}$ 
THEN DO
    ADD ( $n, c$ )
/* calculate the similarity between the node
and concept, then, add into class list the
nodes that have similarity degree higher than
the defined threshold */
END;
END;

Step 2:

Return c;

END;

```

Figure 5.3 Class Formulation Algorithm

```

Input: ListOfNodes  $c$ , HeadshipFeatures  $HFea$ ;

Output: ClassHead Head, SortedClassMembers
        SortedList;

Step 1:

For ( $\forall m \in c$ ) DO

    GET  $Sim(m, HFea)$ 

    /* compute the similarity degree between the
    class node member  $m$  and the predefined headship
    features*/

    END;

Step 2:

    SortedList =  $Sort(c, Sim(m, HFea))$ 

    /* sort class nodes according to their
    similarity degree*/

    Head =  $m \rightarrow \max(Sim(m, HFea) \in SortedList)$ 

    /* select the highest similarity degree node to
    be as head of the class*/

Step 3:

    Return Head, SortedList;

End;

```

Figure 5.4 Head Appointment Algorithm

5.4.1.2 Head Appointment

The class formulation algorithm gets a set of classes that corresponds to the selected concepts. Each class needs to have a head that will ease the communication between the different classes. This process is called head appointment process. In this two-step process, we first need to define the headship features, which a node needs to qualify for it to become a head. In the second step, a head appointment algorithm calculates the similarity between the nodes and the predefined headship features, and selects the class head based on the degrees of similarity. For the first step, we suggest performance capabilities and availability as the headship features. Performance capabilities refer to the basic infrastructure that is needed in order to manage the service grid metadata of the entire class. This may include, the speed of the server, network bandwidth, and reserved memory space for service grid metadata. Availability is the proportion of time when the node persists in an intergrid system. Figure 5.4 illustrates the head appointment algorithm.

The selected head maintains two kinds of information: a summary of the service metadata of its class and concept information of other classes. The first type of information allows it to forward the service request to relevant node within the class. The second information is helpful for forwarding the service request to the relevant class when the request is not related to the class of the requester node. Therefore, each node will have semantic description components as we have described in the previous chapter. Meanwhile, the member node may have the service grid information provider only.

5.4.1.3 Node Subscription

The first two components of the registry architecture produce a set of classes with their heads. In order to allow these classes to be joined by new nodes we provide a mechanism for that purpose, which is called node subscription. Subscription primarily is the procedure of assigning a new node to an existing class or set of classes that corresponds to its service concept. Subscription is done by the node subscription algorithm. In this algorithm, we assume that the new node has been given the

information about the selected service grid concepts during the settings, and the new node sends a message that contains its service concept to any existing node (member/head).

```

Input:   NewNode   nNew,   Concept   T  $\rightarrow$  nNewT,
ExistingNode n  $\in$  N, Threshold;

Output: ListOfHeads List;

/*list of heads for which the new node is
assigned to*/

1:  Send message from nNew to n  $\in$  N

2:  IF (n  $\rightarrow$  HeadOfClass) THEN DO

3:      For ( $\forall$ HeadOfClass  $\in$  ListOfHeads) DO

4:          GET Sim (tnNew, HeadOfClass)

/* calculate the similarity degree between the
new node concept with all Heads' concepts that
are available in the current HeadOfClass*/

          IF (Sim (tnNew, HeadOfClass) > Threshold)

              THEN DO

                  ADD (nNew, HeadOfClassList)

      END;

5:  ELSE forward the message to the HeadOfClass

      Go to Step 3 & 4

Return List;

END;

```

Figure 5.5 Node Subscription Algorithm

The algorithm takes the service concept of the new node, and calculates the similarity degree between the service concept and the related class heads. If the similarity degree attains the predefined threshold, the new node is added to the class of that head. Finally, the algorithm returns the list of heads, for which the node has been assigned to, if the new node has more than one service grids that belong to different concepts. Figure 5.5 illustrates the node subscription algorithm.

5.4.2 Fault Tolerance and Load Balancing Strategy

Some very crucial issues in the intergrid RD systems are the dynamicity of the service grid nodes status, and the management of the node of classes in terms of the number of classes and the number of nodes in each class. The first issue is related to fault tolerance of the system, whereas the second issue is related to load balancing in the registries. In this section, we have incorporated two approaches to deal with these situations, namely class maintenance to handle the dynamicity of the service grid nodes status and class management to handle the settings of the number of classes and their sizes.

5.4.2.1 Class Maintenance

Intergrid node dynamism has an effect on node organization. Thus, a class maintenance scenario to cope with this situation is required. Class maintenance takes effect in two cases: failure of a class head and failure of a class member. Both of these cases can take place in the intergrid system either voluntarily or due to other network problems. In this subsection, we propose two mechanisms to handle head replacement and member replacement.

In the case of head replacement, existing heads are supposed to replicate their resource information to their predecessors in the headship ranking. Remember that a head is selected based on the similarity degree with predefined headship features. Since the existing head has the highest similarity degree, a predecessor can be the second highest and so on. When the head wants to leave or fails the predecessor, the new head informs its class node about itself and performs all functions of the previous head. Figure 5.6 shows the head replacement algorithm.

```

Input:    MessageTime    t,    ClassHead    Head,
PredecessorHead PHead;

while (time = t) DO

Head send nodeInfo → PHead;

If ((time > t ∧ !nodeInfo) ∨ LeaveMessage) THEN DO

/*if the PHead does not receive the respective
info within the identified period of time or it
receives a leave message from the current head,
then PHead declares itself as a head */

For ((∀ Heads ∧ calassNode)

Inform about the new Head;

Send nodeInfo → PHead;

END;

```

Figure 5.6 Head Replacement Algorithm

Member replacement can be achieved by connecting the direct neighbors of the withdrawn member. As each class is a connected graph, each member is connected to

some neighbors say two (back neighbor and front neighbor) and its leader. A member informs its back neighbor about the front one, and likewise the front about the back. When the member in between the back and front members is dropped, the two remaining members fill the gap through their connections. Figure 5.7 shows the member replacement algorithm.

```

Input: MessageTime t, ClassNode Node,
        RightNeighbor RNode, LeftNeighbor LNode;

while (time = t) DO

    Node sends nodeInfo  $\rightarrow$  RNode  $\wedge$  LNode ;
    Node sends RNodeInfo to LNode;
    Node sends LNodeInfo to RNode;

    If ((time > t  $\wedge$  !nodeInfo)  $\vee$  LeaveMessage) THEN DO

        LNode connects to RNode  $\vee$  RNode connects to LNode

    END;

```

Figure 5.7 Member Replacement Algorithm

5.4.2.2 Class Management

The classes of the service provider nodes are supposed to be managed by their respective assigned heads. The management include accepting registration of new service grid provider nodes or assisting newly joined nodes to get their respective head, hosting the service grid metadata of the class members, updating the service grid metadata when is needed, processing the incoming service requests, forwarding the request of class members to the respective class and forwarding the replies, and

so on. The whole managing process involves a huge amount of messages, coming to and going from the head to the other heads and class members. As a matter of fact, if we do not have optimization strategy to manage this tremendous amount of traffic, we will eventually be in a situation of bottle neck in the head. The management strategy should be able to balance between the number of nodes and the number of classes in way that there would not be any management issues. In fact, using the dictionary ontology, the number of class C in a given intergrid system is between $(1 - S^h)$ (we treat the ontology as a rooted tree).

$$C = \begin{cases} 1 & \text{if } h = 0 \\ S^h & \text{if } h > 0 \end{cases} \quad (5.1)$$

Where S is the number of the leave concepts, and h is the total level of concept hierarchy. $C = 1$ when we assign all nodes to one concept which is the root, and $C = S^h$ when we assign the nodes to each leave concept. However, the two situations will rarely occur. Therefore, in our case we assume that the number of classes falls between the two situations. Previously, we have mentioned in chapter two that centralized registration models work effectively in small grids, where there are only few hundreds of nodes; but they have no fault tolerance. Since we have fault tolerance in our class heads, we use the idea of having few hundreds of nodes to be managed by one head. Therefore, we can define a variable called the max number of nodes in the class, μ that should be within hundred, to control the number of nodes in the class. Therefore, the number of class starts by selecting the most general concepts in the DO, then when the nodes under a particular class (concept) has reached the μ , we split the concept by selecting a number of more specific subconcept. This will ensure that every class can grow smoothly with a balanced management in the heads. Figure 5.8 illustrates the class management algorithm (CMA). In order to implement CMA, we first need to define the next concepts that will be used to split the current class concept when the number of members has exceeded the defined maximum class size. Upon that, CMA takes the set of this concepts and the set of the current class members, and formulates new classes based on the identified set of concepts. Namely, for each concept the CMA calls the class formulation algorithm which will return the list of nodes for that concept, calls the

head appointment algorithm to return a head for the class, and finally removes the list of nodes of the new class for the global list of the initial class (the class that is being split currently). This will continue until all the new classes have been formulated.

Apparently, the CMA provides a load balance based on load management. For example, the class will not be split only because the class size has exceeded the expected number of nodes that it can manage, if it does not then we do need to split. In another aspect, the split of a concept to few subconcepts will allow the distribution of current class members among new classes, and each class will have the chance to grow its member (since we have node subscription mechanism) until the class reaches its maximum size. Therefore, we conclude that CMA grants load balancing in the registry architecture.

```

Input: maximumNumberOfNodes  $\mu$ ;
        SetofNextConceptes  $Y = \{y_1, \dots, y_z\}$ ;
        SetOfClassNodes  $N = \{n_1, \dots, n_{\mu-1}\}$ ;
Output: listOfNewClassNodes  $L = \{l_1, \dots, l_z\}$ ;

while ((newNode  $\rightarrow$  Join)  $\vee$  existingNode  $\rightarrow$  leave
        ) DO

    Count numberOfMembers;

    /* while new nodes join the class and existing
    ones may leave keep track on the number of the
    current members of this class */

    If (numberOfMembers  $> \mu$  ) THEN DO
        For ( $i = 0$ ;  $i < \text{Size}(Z)$ ;  $i++$ )
            For( $j = 0$ ;  $j < \text{Size}(N)$ ;  $j++$ )
                ClassFormulation ( $n_j, y_i, \text{Threshold}$ ) ;

                /*once the number of the members exceeds
                the default maximum number in the class
                run the class formulation algorithm with
                regard the next defined concept*/

            Return the classList;

        HeadAppointment ( classList, HeadshipFeatures);

         $N = N - \text{classList}$ ;

        /*in each formed class run the head
        appointment algorithm to get a head for
        this class and remove the members of the
        new class form the list */

    END;

END;

```

Figure 5.8 Class Management Algorithm

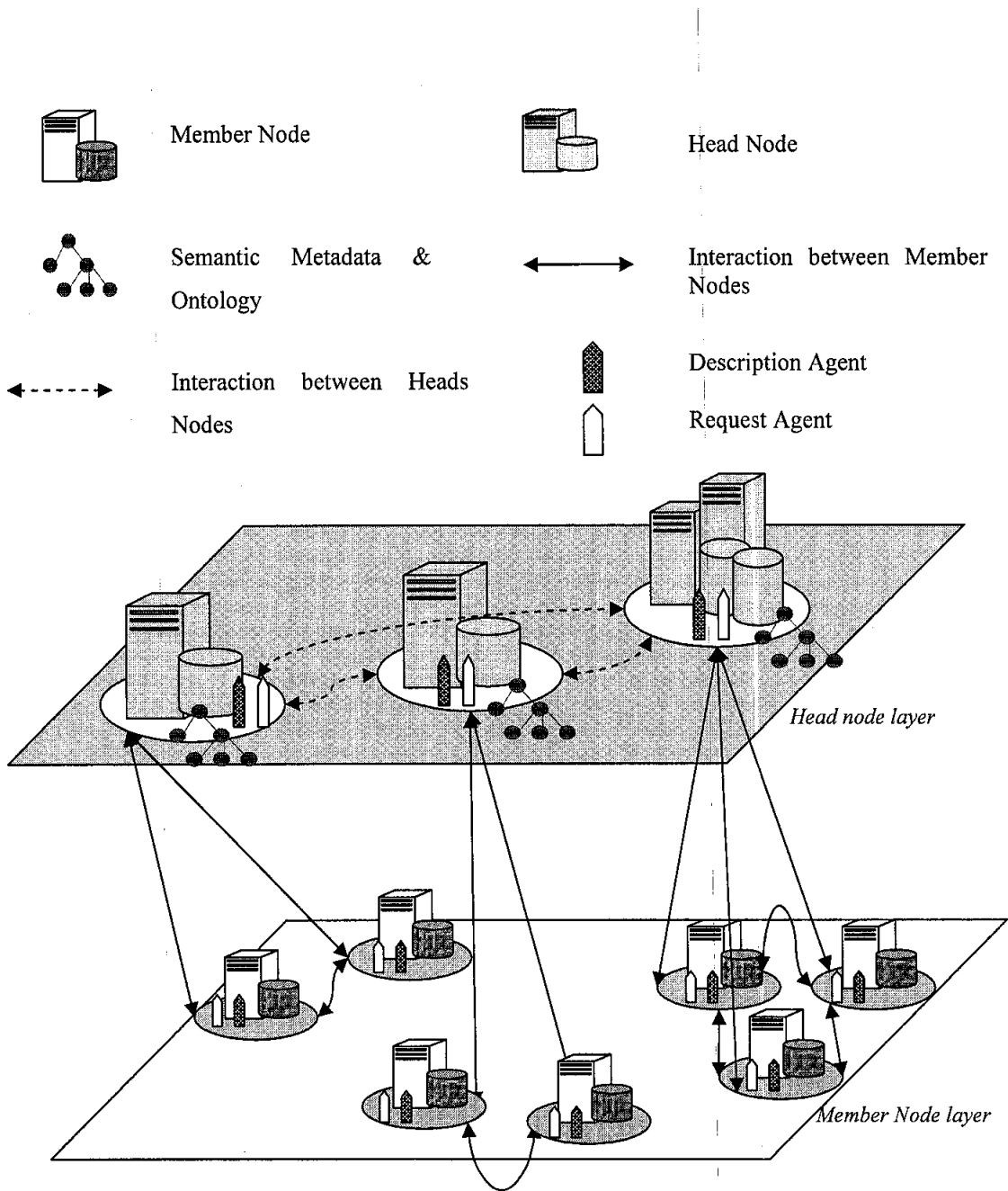


Figure 5.9 The overall intergrid system

5.4.3 The Discovery Algorithm

In the previous chapter, we have presented the description model component which confers the semantic representation of the service grid information. So far in this chapter, we have discussed the registration component, which is covered by the registry architecture. The only remaining component of our new RD system is

discovery. In the discovery component, we use the identified two agents to abstract the discovery process from the user, and to deal with the status change of the service grid metadata. Therefore, each service grid node is supposed to have both agents, and the communication between the nodes will be through the exchange of messages between the agents. Figure 5.9 shows the overall framework elements, where there are some nodes assigned to some classes with their heads. The collection of heads forms a head node layer, whereas the collection of classes and members forms the member node layer. Each node has two agents (DA and RA), and implements the service grid information provider element to describe their service grid information. In addition to service grid information provider implementation, heads implement the semantic description manager element to assist the members to describe and register their services. When this has been achieved, our model now is ready for the discovery process.

Our intergrid system initially consists of the set of nodes with their services and connections between these nodes. This can be represented as a graph structure $\therefore G = (N, E)$, $N = \{n_1, n_2, \dots, n_i, \dots, n_n\}$, $E = \{e_1, e_2, \dots, e_i, \dots, e_x\}$ where N represents the set of grid nodes (the total number of nodes regardless of the class structure) and E the set of connections between nodes. The overall number of services can be represented in a vector $S = [s_1, s_2, \dots, s_i, \dots, s_k]$. Therefore, the existence of service within the different nodes can be represented in an adjacency matrix that is made of the set of nodes, N and the vector of the services, S . Let us denote this matrix by an *intergrid service matrix*, M . Therefore we get:

$$\therefore M(N \times S) = \begin{pmatrix} n_1 s_1 & \cdots & n_1 s_k \\ \vdots & n_i s_j & \vdots \\ n_n s_1 & \cdots & n_n s_k \end{pmatrix}$$

Where the elements of M , $n_i s_j \in [0, 1]$, $n_i s_j = 1$ when node n_i has the service s_j , and 0 if otherwise.

Since the intergrid service matrix contains the existing services with the entire number of nodes, an adjacency matrix is made from nodes of a class and is a subset of the overall services, which is actually a submatrix of the intergrid service matrix (M), which we call *class service matrix* (m_c). Therefore, the service metadata in each

head will have a class service matrix(m_c). In addition to that, each node may have some neighbors with which it exchanges service metadata. The set of neighbors with the services that exist with them represents a submatrix of the class service matrix (m_c), which we call *node service matrix* (m_n).

The goal template matrix (ω) in chapter 4 (equation 4.7) can be recalled back here by having the set of goals $G = \{g_1, g_2, \dots, g_i, \dots, g_x\}$:

$$\therefore \omega = G \times S = \begin{pmatrix} g_1s_1 & \dots & g_1s_k \\ \vdots & g_is_j & \vdots \\ g_xs_1 & \dots & g_xs_k \end{pmatrix}$$

Where the elements of ω , $g_is_j \in [0, 1]$; $g_is_j = 1$ when a goal g_i requires s_j service grid, and it becomes 0 if otherwise. Each class head is supposed to have a copy of the goal template matrix. However, it is possible that a node in a class may have a submatrix of the goal template matrix, which may be obtained when a consumer among the member nodes gets an instance of goal. The submatrix of the goal template of the node is called node goal template matrix ω_n .

To allocate the services for user's goals, we develop an algorithm that searches for service on the network based on the cached information and dynamic matching. The cached information is the presence of a particular service in a node, which is illustrated in the service submatrices (m_n & m_c). Dynamic matchmaking is the similarity calculation between agents that represent service provider and requester, using the similarity function. Figure 5.10 illustrates the discovery algorithm.

```

Input:      NodeInformation,      ServiceRequest,
           Threshold;

Output: ListOfNodes List;

Step 1:
Get GoalInstance
Add ConcreteValue to ServiceRequest
Get NodesInformation Form DA

Step 2:
IF (NodeInformation → neighboringNode) THEN DO
    For (∀ neighboringNode → requestedServices)
        Get Sim(RA, DA)
        IF (Sim(RA, DA) ≥ Thrashold) THEN DO
            ADD(neighboringNode, list)
    ELSE Send ServiceRequest to ClassHeads
FOR( ∀ ClassHeads ∈ Class →
    requestedServiceConcept)
    IF (∃ ClassNode → requestedServices)
        THEN DO
/*send the service request to all the heads
that are available in the current head local
info and associated the service request
concepts*/
        FOR (VClassNode → requestedServices)
            Get Sim(RA, DA)
            IF Sim(RA, DA) ≥ Thrashold THEN DO
                ADD(ClassNode, list);

Step3:
Return List;
END;

```

Figure 5.10 The Discovery Algorithm

5.4.4 Application

In order to clarify the interaction between the components in our system in describing/discovering a service using the discovery algorithm, we introduce an example that shows how a user can describe or request a service.

Let assume that we build an intergrid with 1024 nodes distributed in different locations. Using the dictionary ontology, we can define the service concepts, say 4 concepts, each concept may contain 32 services so that the total number of services is $32 \times 4 = 128$. Then the two agents (DA and RA) in each node are initialized. Each node can be represented by its DA agent.

Implementing the class formulation and head appointment algorithms respectively on nodes or DAs (the DA represent the service grid as it carries the service grid information), we get a set of classes with their heads. For simplicity, we may have 4 classes as the selected service concepts are 4 and each class has 256 nodes. Each DA sends its service information to its head; therefore the head will have the entire information of class summary, which will be a class service matrix (m_c) of the previous section. In case new nodes want to subscribe to the system, the node subscription algorithm in section 5.4.1.3 is activated. On the other hand, if an existing class member or head node quits the system, the mechanism of class maintenance in section 5.4.1.4 will manage the exit.

Assuming an end user wants to run some applications. The steps for him/her to request and discover the services according to our new framework are as follows:

- a) Based on the goals that are stored in the service goal template of the semantic description manager or the head of that user' node, the user selects the preferred goal and obtains instance of the goal. The user then adds the concrete values of the service capabilities, which enable the *RA* to form a service request vector(s), say 6 services.
- b) If there is local information about some neighbouring nodes that has been given by their DAs (node service matrix(m_n)), *RA* sends a request to any neighbouring

node n_i , that is associated with all or part of the 6 requested services and the threshold of the similarity degree.

- c) Based on the description of services in RA and DA , the similarity degree of the two agents $sim(RA, DA)$ concerning the service properties of the requested service and provided service is calculated.
- d) If the similarity degree of $sim(RA, DA)$ reaches a user defined threshold value, then the node n_i is selected; and the check is done whether there are still remaining requested services to be searched.
- e) If there are remaining service request, steps c and d are repeated until none of the nodes in the class is any longer associated with the requested service.
- f) If so, then the remaining requested services are sent to a class head c_i .
- g) From the service matrix (m_c), head c_i sends the service request to another class head/heads c_j that may have the remaining requested service based on the concepts.
- h) For each head, the steps (b), (c), (d) and (e) are performed until all the 6 requested services are found.

These are the main steps that represent the invocation of the new RD system. It should be noted that the system can also work without the use of agents, if the developers are not concerned with the user interaction abstraction and frequent update of the service metadata in the set of subregistries at the class head. When that happens, all the interactions may be done manually by the end user.

5.5 Computational Complexity of the New RD System

A very critical feature in any given system is computing complexity, which includes the time and space that are need by the system to solve a presented problem. In the context of the RD system, time complexity is very important to ensure that the time taken from formulating the request until receiving the response for that request

(latency) is acceptable. In this section, we will discuss the time complexity of our new proposed RD system.

From the formalism that we have presented in section 5.4.3, let us bring the set of nodes $= \{n_1, n_2, \dots, n_i, \dots, n_n\}$, the connection between the nodes $E = \{e_1, e_2, \dots, e_i, \dots, e_x\}$ and the set of services $S = [s_1, s_2, \dots, s_i, \dots, s_k]$. For each, node n_i we assign weight $w(n_i)$, which refers to the time cost when service request passes through n_i . We do not take into consideration the time cost on the edges or connection between the nodes, as these are not included in our scope of study. The aggregation of nodes that connects to node n_i , we called by neighbor γ of n_i so we denote it as $\gamma(n_i)$.

Regardless of the distribution of nodes N into some classes $C = \{c_1, c_2, \dots, c_i, \dots, c_\mu\}$ and the number of classes identified as $|C|$, we can sort out N into two categories as service providers and heads, which we denote as N_P and N_H respectively. The reason for that is that head nodes are the one that is responsible for forwarding the request from their members or to respond to any request that has been forwarded to them. Hence, the head nodes can be a provider, but the member cannot forward the request as they do not have this privilege. All of these nodes have service metadata with them, which vary from one category to another. For example the service providers N_P have the detailed information about what they provide. Meanwhile the heads, N_H have a summary of the members in their classes that are service providers that will enable them to chose the right service provider when a request is sent to them, and other information about each other that will allow them to forward the request to the right heads among them when the request is beyond the concepts of their classes. Therefore, in any case we need to check the information of the nodes by calculating the similarity degree between a node and a request. However, the similarity calculation may vary from one case to another in terms of threshold value, and the features that we use as input for the similarity function. For example, when a request for a service s_i is sent by any member n_i to its head, the first similarity calculation is to check whether s_i belongs to the concept of this class or not, through $sim(s_i, c_i)$, where c_i is the concept of the class. If the s_i has a similarity degree that reaches the defined threshold, then it will

be forwarded to one of its member node n_j . So the next similarity calculation will be between the s_i and the selected member candidate, through $sim(s_i, n_j)$. To that end, we assume that the time for checking the information in any given node will be the same, which is $sim(a, b)$; where a is a service request, and b is an information input such as a class concept or provided service. Therefore, by using the Dice distance function, we can have the time complexity for information checking (τ) as follows:

$$\tau = w(n_i) = O(a \times b) \quad (5.2)$$

Since services S must belong to the dictionary ontology, concepts T are based on the classes that are formed, we donate the concept of every service s by $t(s)$. If we have several services $s_1, s_2, \dots, s_i, i \leq |S|$ have the same concept, then we will have this formula $t(s_1) = t(s_2) = \dots = t(s_i)$. Based on the intergrid service matrix M , we can formulate the existence of a service with a concept in a class $c_i \in C$ as $(c_i, t(s)) = 1$, when the service exists with a given class, and $(c_i, t(s)) = 0$ when the service does not exist.

Based on the discovery algorithm, the routing (R) of a service request $t(s_i)$ among the classes to reach a provider node n_i is given as:

$$R(n_i, t(s_i)) = n_u, n_u \in (\gamma(n_i) \cap N_H) \quad (5.3)$$

Therefore, the RD process can be seen as routing the request $t(s_i)$ from the head of the requester (n_0) to a provider node (n_i) i.e. when $t(s_i) = 1$. This can be formulated as path P and can be calculated through this equation:

$$P = n_0 e_0 n_1 e_1 \dots n_{i-1} e_{i-1} \quad (5.4)$$

Where $1 \leq i \leq N$, $n_i \in N_p, n_j \in N_H, 0 \leq j \leq i$. The shortest path (p_s) from the requester node (n_r) that belongs to a class (c_x) to reach n_i can be calculated as:

$$P_{min} = n_{i-1} e_{i-1} \quad (5.5)$$

Where n_{i-1} must be the head of n_r or the direct neighbor in class c_x , where the connection $e(n_r, n_{i-1}) = 1$. The weight of the path $W(P)$ is actually the summation of the sum of weight of each surpass node $w(n_j)$ and the connection time $w(e_j)$.

$$W(P) = \sum_{j=0}^{i-1} (w(n_j) + w(e_j)) \quad (5.6)$$

The weights of the shortest path P_{min} and the longest path P_{max} can be denoted as W_{min} and W_{max} respectively. Therefore, the time complexity of our RD system τ_{total} to find a given $t(s_i)$ at the worst case is the W_{max} of the path P_{max} .

$$\tau_{total} = O(W_{max}) \quad (5.7)$$

To that end, we conclude that the complexity of our RD system is linear, which renders the system as capable of providing high performance.

5.6 Discussion

As we have mentioned in chapter 2, the features of the RD system that are expected to be achieved mainly by the registration and discovery components are scalability, decentralization, and dynamism. This section discusses how the new model meets these characteristics.

Scalability, the model is scalable by using the class based node organization, which gives an opportunity for any class to grow rather than treating all the nodes as one group, which is the case of the current centralized RD methods.

Decentralization is achieved in our system as follows: each node maintains its service information; and each head node maintains a summary of information of other class members including the class heads as well (service concepts). This means that no node controls the entire intergrid system. In addition to that, we show how a head node can be succeeded when it wants to leave the system or fails due to connection.

Therefore, the system may not be completely down as in the centralization circumstances.

Regarding dynamism, we use intelligent agents, DAs to track the status of each node resources, which in turn allows nodes to update their subservice matrices. In addition to the above features, the model provides fault tolerance and load balancing. In this case, the model can tolerate the failure of member nodes and class heads as described in section 5.4.2, and split the class into subclasses when the number of member node rises beyond a manageable number, which has been predefined for each class (class size).

CHAPTER 6

RESULTS AND DISCUSSION

6.1 Introduction

This chapter presents a comprehensive quantitative evaluation with respect to the overall performance of the proposed RD framework. The aim is to examine the performance of the system and highlight the performance increase that is attainable by the framework. The chapter begins with the methodology that has been adopted to conduct the evaluation. The chapter then discusses the simulation tool that is used in this evaluation. The chapter thereafter identifies the experimental settings, and reports and discusses the achieved experimental results. The chapter lastly presents a comparative study between the proposed RD and the most promising related work.

Overall, the main contributions of this chapter can be summarized as follows:

- An extensive simulation to evaluate the proposed RD framework.
- An analysis of the performance of the proposed RD system and a comparative study with some of the related work.

6.2 Methodology

Evaluation of the proposed RD system is very important, as it allows an examination of the performance of the proposed system, and eventually to draw a conclusion on its performance. For this, we have chosen one of the P2P simulators called PeerfactSim.KOM to simulate the intergrid environment with the application of the proposed system.

The evaluation of the system is based on some common performance metrics found in the literature (Mastroianni et al. 2008) and (Mastroianni et al. 2005) . This includes the percentage of the discovered services in a given goal request, and the response time for the service request to be answered.

These metrics are calculated in different settings of the nodes and service requests. Therefore, we start with a few numbers of nodes, and scale them gradually to simulate the increase of the services in the actual intergrid system. We also vary the rate of service requests from small number of requests to bigger number to simulate the increase of users in the intergrid system. We analyze the results of the different settings by highlighting the causes of the effects of the different setting to the results.

6.3 Grid Simulation Tools

Generally, there are two simulation tools developed specially for grid technology research. These simulators are GridSim⁵⁷ (Buyya and Murshed 2002) and SimGrid⁵⁸ (Henri et al. 2008). GridSim provides a framework for modeling and simulation of grid entities such as resources, applications, users, and resource brokers/schedulers in order to design and evaluate scheduling algorithms. GridSim is originally based on a discrete event simulation package that is called SimJava (Howell and Ross 1998).

SimGrid also is a discrete event simulation tool that offers core modeling and simulation capabilities for simulating distributed and parallel scheduling applications in heterogeneous distributed environments. These environments range from simple network of workstations to Computational Grids. SimGrid relies on C and Java languages.

We have examined both simulators, and found that they mainly focus on the scheduling algorithms and frameworks; they do not address the simulation of the RD framework directly. As a matter of fact, most of the studies in RD field are using other alternatives such combining the simulation tools of P2P network with grid simulators

⁵⁷ <http://www.gridbus.org/gridsim/>

⁵⁸ <http://simgrid.gforge.inria.fr/>

or using them alone, or developing their own simulator. For example, Rajiv et al combine GridSim with PlanetSim (Garcia et al. 2005) to simulate their work (Rajiv et al. 2007). Studies such as (Han and Berry 2008) and (Mastroianni et al. 2008) have developed their own discrete event simulators. Therefore, we further examined two discrete event based P2P simulators which are PlanetSim and PeerfactSim.KOM⁵⁹ (Kovacevic et al. 2007) that provide the core entities to simulate RD systems. We find that, the second simulator (PeerfactSim.KOM) has more flexibility and documentation compared to the first one. Based on these findings, we opted for the second simulator.

6.4 PeerfactSim.KOM

PeerfactSim.KOM is an object-oriented java based Peer-to-Peer evaluation platform, which gives us the ability to create an overlay and simulate large-scale networks with it. Therefore, PeerfactSim.KOM allows us to conclude about functionality of any overlay network and answer the most important questions such as scalability, efficiency, and flexibility.

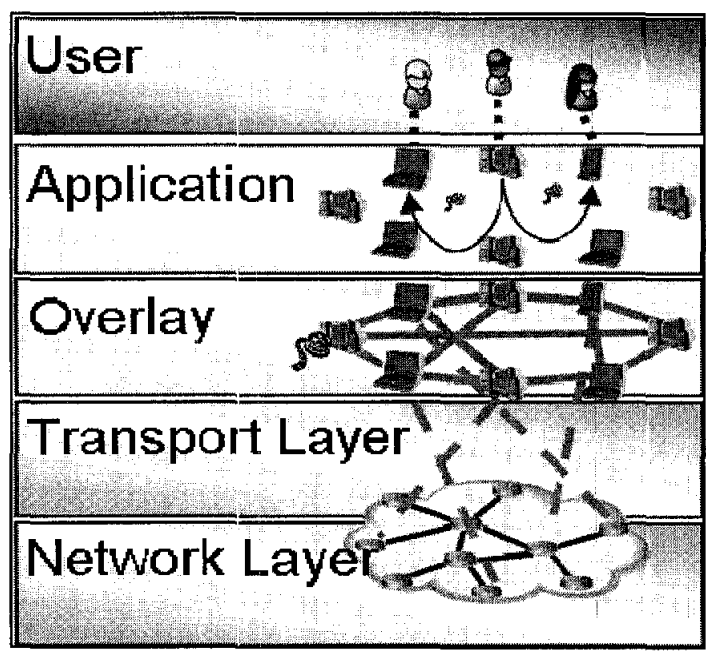


Figure 6.1 The layered architecture of PeerfactSim.KOM

⁵⁹ <http://peerfact.kom.e-technik.tu-darmstadt.de/>

PeerfactSim.KOM simulates distributed systems in four layers as illustrated in Figure 6.1 (taken from <http://peerfact.kom.e-technik.tu-darmstadt.de/>). The reason behind that is to effectively address the complexity of distributed systems and analyze them smoothly in distinguishable parts.

6.5 Experimental Setup

Our experimental setup initially is divided into two components namely intergrid environments and the semantic service grid RD system. This component arrangement follows the typical architecture of the PeerfactSim.KOM. Initially, we build our simulation of the new RD system by using the available packages in the PeerfactSim.KOM that provides the network and the transportation infrastructures. We also modify some of the implemented P2P networks in the simulator. We use an integrated development environment (IDE) NetBeans⁶⁰ 6.8 to edit the code of the simulation components.

We build an intergrid system that consists of n nodes. The size of the nodes n is scaled from 100 to 1000 with scale of 100 and 200. Since the creation of service grid domain ontology and dictionary ontology are outside the scope of our work, we simulate these ontologies by representing them numerically. Where the concepts of the ontology are simulated by positive integer values such as 1, 2, ..., k , and each concept has subconcepts/properties which are some predefined set of values. Based on that, the concepts are representing the services' concepts and the predefined values are representing the services themselves. Each of the nodes has a set of the services. The number of these services is varied between 1 and 6 services. The reason of having this range of numbers is that our RD system is based on aggregating the service grid resources and services metadata information, which obviously reduces the range of services in the intergrid system. The allocation of the size of services in each node is random, which is the same as the assignment of concepts to node. This is to simulate the fact that in intergrid system we may not be able to neither express precisely the number of the services in each node nor the type or the concept to which

⁶⁰ <http://netbeans.org/>

these services belong, but surely we can define the concepts in the first place. During the simulation of class formulation, each of these nodes will be joining a particular class based on the randomly assigned concept. The number of class is based on the number of concepts (if the selected concepts for the overall nodes are five, the corresponding number of the classes is also five). The selection of concepts is proportional to the size of the intergrid system. This is in correspondence to the super-peer architecture where the number of super peers is based on the size of the network. In (Yatin et al. 2003) the number of super-peer node is implemented to be 5% of the nodes that have very high capacity to handle queries. We have adopted the same percentage so as to systemize the distribution of the node to classes in way that allows us to conveniently discuss the performance of the system. Therefore, an intergrid system that has a size of 1000 nodes will have 10 classes. As we have discussed previously, centralized registration model is effective for handling the service metadata of hundred nodes; therefore, we set the class size to 100 nodes so that each head of the class can accommodate the service information of hundred nodes that belong to its class.

Table 6.1 Simulation parameters

Parameter	Value
Number of intergrid node (service grid) N	100 to 1000
Number of the provided service in each node	1 to 7 which is selected randomly
Number of the Classes/Concepts	10
Maximum size of each class	100 nodes
Number of neighbors for each head	1 to 9
Time to live TTL	2 to 5

For simplicity, during the simulation, nodes that have high capacity which are supposed to be the heads of classes will join the network first and declare themselves

as heads. Then they will connect with each other in order to form the network. The number of neighbors for each head varies from one to the total number of heads in the network. This is because the connection of the heads is random as it is based on which head starts the communication first. Table 6.1 presents the summary of the main parameters.

Our simulator is enabled by passing two configuration files, which are known as configuration file and action file. Configuration file holds the values of the above mentioned simulation parameters such as the size of network, the classes and procedures that will be called by the simulator to create the components of the system (host server, service and so on). Meanwhile, the action file holds the command that has been developed to simulate the processes of the system such as the class formulation, node subscription, discovery algorithm, and so on. It is suggested in the simulator documentation that the commands should be passed to the nodes in portion basis. As such, the total number of nodes should be divided into several groups, and each group should be given a name. When the group name command is given, all the nodes that belong to the group will be concerned. It should be noted that this has no effect to the simulated architecture, rather it eases the simulation of the different procedures.

6.6 Performance Indices

The performance indices used in this evaluation are request/query hit, and the response time for the request as defined in (Mastroianni et al. 2008).

Request/query hit or the percentage of the service requests that are answered successfully with regard to the total number of the generated requests by a node in a given time is very critical in proofing how efficient the RD system is.

Response time is the time interval between the generation of the service requests and the reception of the results. All of these metrics will have different values when we implement different simulation parameters. In order to have a fair examination, we vary the number of service requests that are generated by the nodes. In this case, the service requests are generated in percentage proportional to the intergrid size. For example, 25% of the total number of nodes that belong to several different classes can

generate requests at one time. Figure 6.2 illustrates the rate of service requests generation with regard to the different intergrid sizes. For example, in an intergrid size of 100 nodes we have a generation of 25, 50, 75 and 100 requests. Meanwhile, for an intergrid of size 800 nodes we have a generation of 200, 400, 600 and 800 requests.

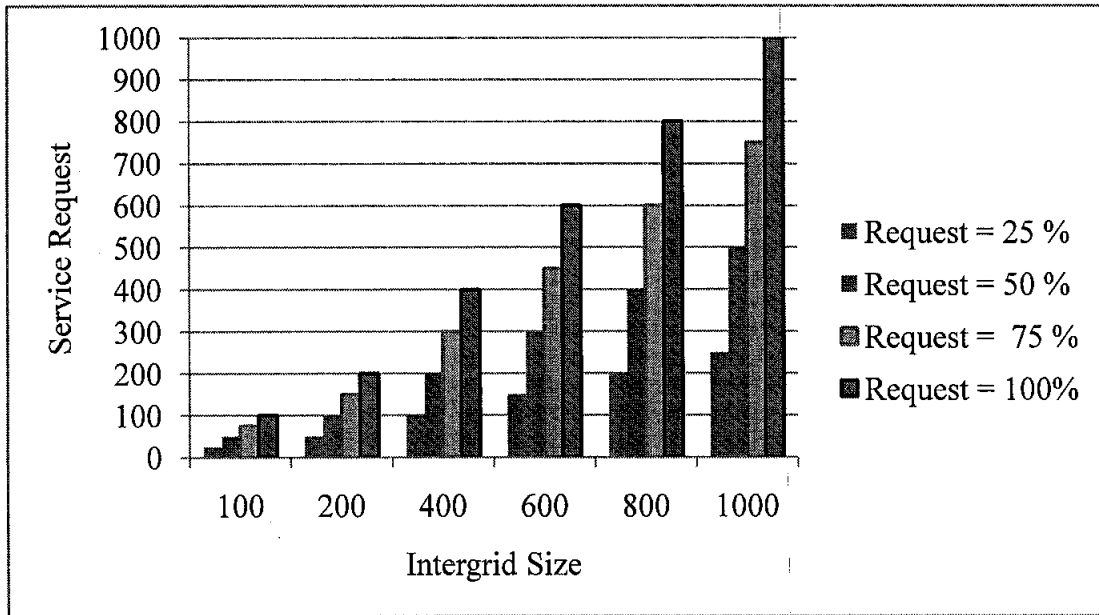


Figure 6.2 Proportion of the generated service requests to intergrid size

6.7 Performance of the New RD Framework

We conducted 16 (this number corresponds to the variation of service request generation and TTL values) independent experiments for different service request portions and intergrid sizes. In each experiment, the mechanisms and algorithms that we have designed in the previous chapters are simulated. This includes the formulation of classes, subscription of new node, the discovery algorithm, and so on.

We first start our evaluation with the first performance metric, which is the service request hit. Tables 6.2 - 6.5 and figures 6.3 - 6.6 show the simulation results for service requests generated by the nodes in percentages of 25%, 50%, 75% and 100% of the actual size of the intergrid. We control the forwarding of the request message from the requester node to the provider by the TTL values since we implement the super-peer architecture in our registry.

Table 6.2 Percentage of Discovered Services (SRH) for generated requests equivalent to 25% of the intergrid size with different TTL values (2-5)

Intergrid Size	SRH/TTL 2	SRH/TTL 3	SRH/TTL 4	SRH/TTL 5
100	25%	50%	70.83%	95.83%
200	19.14%	52.08%	78.72%	85.1%
400	19.54%	48.27%	75.58%	97.67%
600	20%	30.4%	57.48%	77.6%
800	17.57%	40.11%	58.18%	75.3%
1000	12.8%	31.86%	80.58%	91.78%

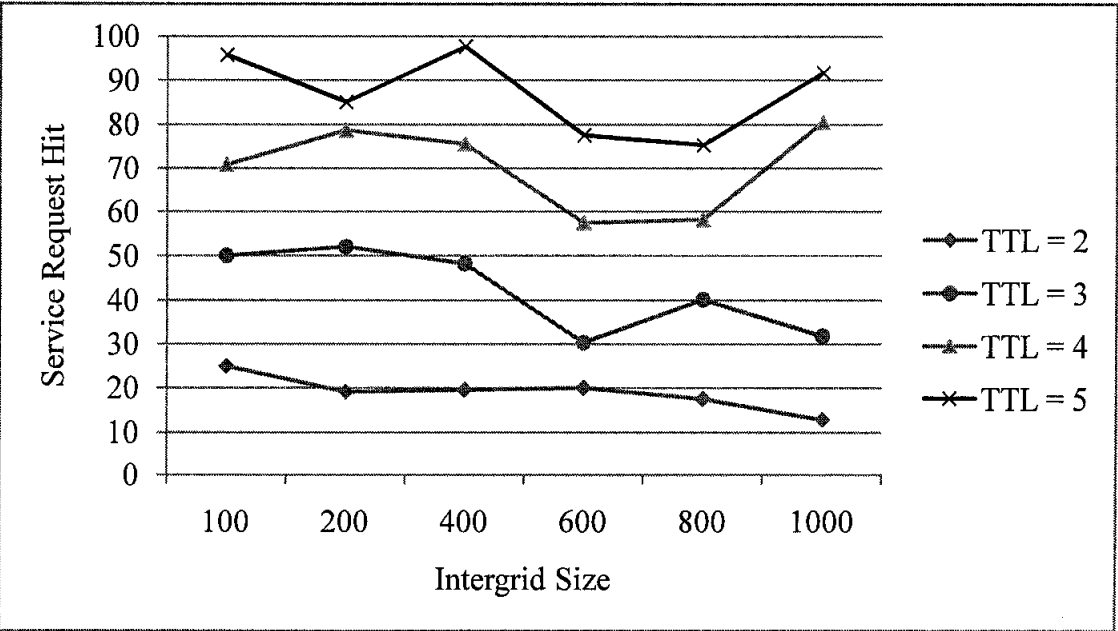


Figure 6.3 Discovered Services for generated requests equivalent to 25% of the intergrid size with different TTL values

It appears from all the figures (6.3- 6.6) that the rate of discovered services is low when the TTL is equal to 2. This is because the scope of service request forwarding is limited to within the classes only, or between the heads if it happens that the head node itself generated the request.

It is also very clear that the rate of discovered services becomes smaller with the increase of request rate and intergrid size. For example, in Figure 6.3 the rate of the discovered services achieves 25% initially, then drops gradually until 15.62% in Figure 6.6. This is because as the service requests increase, the portion of the requests that is sent out of the requester node classes may be higher. This may also happen when the size of the intergrid system is scaled up.

Table 6.3 Percentage of Discovered Services (SRH) for generated requests equivalent to 50% of the intergrid size with different TTL values (2-5)

Intergrid Size	SRH/TTL 2	SRH/TTL 3	SRH/TTL 4	SRH/TTL 5
100	18.333%	30.35%	58.33%	86.2%
200	16.66%	36.36%	84.42%	93.49%
400	20.98%	54.58%	79.16%	95.88%
600	20.38%	34.9%	64.54%	78.67%
800	18.16%	33.96%	61.97%	80.21%
1000	14.19%	29.95%	61.38%	92.33%

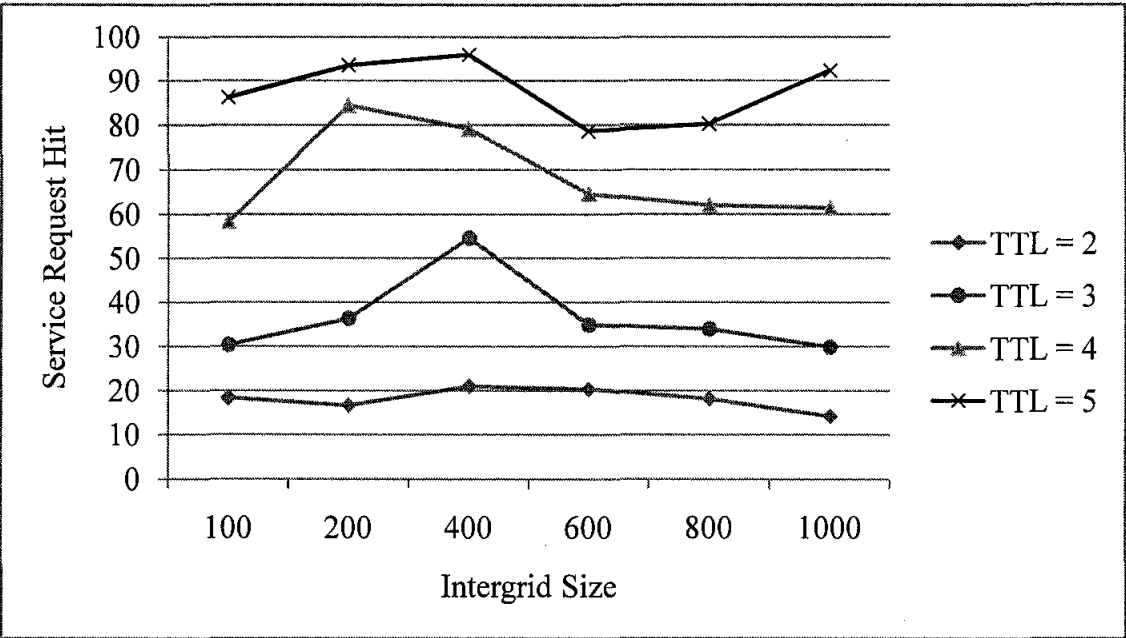


Figure 6.4 Discovered Services for generated requests equivalent to 50% of the intergrid size with different TTL values

Table 6.4 Percentage of Discovered Services (SRH) for generated requests equivalent to 75% of the intergrid size with different TTL values (2-5)

Intergrid Size	SRH/TTL 2	SRH/TTL 3	SRH/TTL 4	SRH/TTL 5
100	17.72%	34.61%	64.1%	87.34%
200	17.61%	38.6%	80.12%	94.93%
400	20.31%	50.15%	82.13%	97.49%
600	21.68%	35.71%	67.22%	83.92%
800	16.66%	33.8%	54.34%	71.87%
1000	16.35%	34.59%	72.98%	90.53%

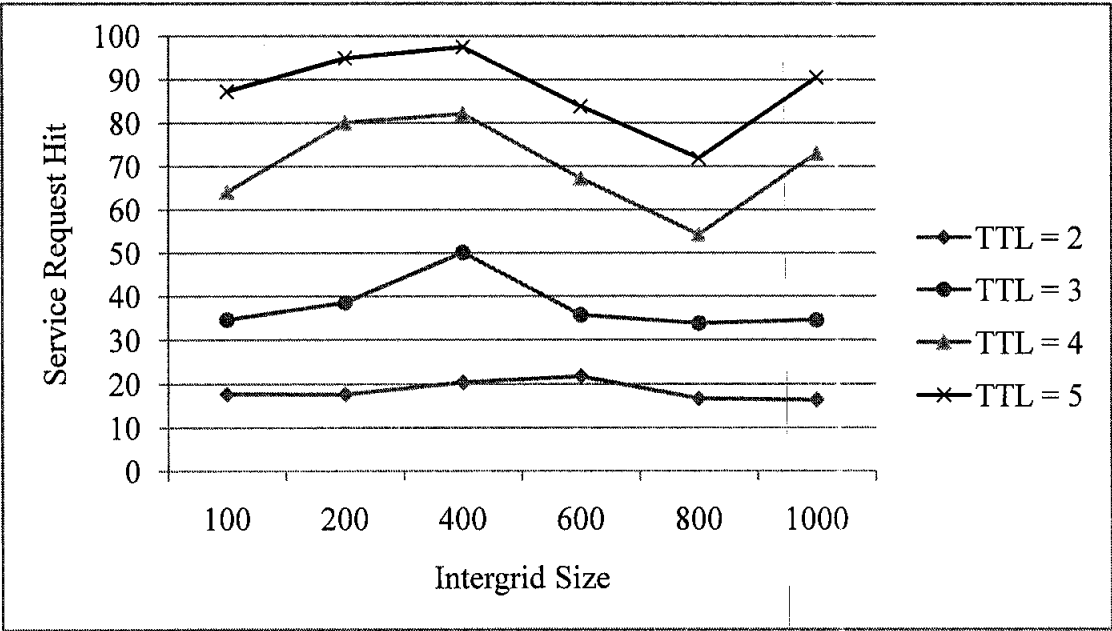


Figure 6.5 Discovered Services for generated requests equivalent to 75% of the intergrid size with different TTL values

Also the four figures (6.3 – 6.6) unambiguously indicate that the increase of TTL will allow the discovery of more services. For instance, the discovered service rate reaches its highest value 95.83% for an intergrid system consisting of 400 nodes. However, the cases of intergrid size 600 and 800 nodes appear to be different as the rate of discovered services decreases gradually until it reaches the lowest value at size of 800 nodes. The reason behind that is due to the implementation of the load balancing algorithm.

Table 6.5 Percentage of Discovered Services (SRH) for generated requests equivalent to 100% of the intergrid size with different TTL values (2-6)

Intergrid Size	SRH/TTL 2	SRH/TTL 3	SRH/TTL 4	SRH/TTL 5	SRH/TTL 6
100	15.62%	32.29%	58.33%	85.71%	97%
200	16.32%	38.19%	80.71%	94.94%	98.46%

Intergrid Size	SRH/TTL 2	SRH/TTL 3	SRH/TTL 4	SRH/TTL 5	SRH/TTL 6
400	18.62%	52.02%	83.79%	96.44%	97.47%
600	18.95%	33.89%	69.81%	86.34%	92.95%
800	16.94%	28.14%	57.86%	69.94%	85.56%
1000	16.07%	32.76%	65.45%	82.82%	94.45%

In fact, the initial idea of the load balancing mechanism is to split the concept from general to a more specific concept so that we get more classes when a class reaches the maximum predefined size. However, this is hard to be simulated with the simulator as the creation of the nodes, services, and concepts is supposed to be before the intergrid join process starts. Therefore, we simulate the load balancing algorithm by creating new classes during the join process. In this case, if a head of class gets 100 hundred nodes in its class it will reject any new node that wants to join the system. When this happens, the rejected node will create a new class of the same concept and accept other nodes that want join the intergrid and have the same service concepts. Therefore, in the case of intergrid size 600, there are few classes that created, and there are more in the case of size 800 nodes. So these nodes cannot reach the services that are available beyond the TTL of value 4 for instance. As can be observed in all of the figures (6.3 – 6.6) the rate of the discovered services starts increasing at TTL of value 5. To further investigate that observation, we increase the TTL value up to 6. As indicated in Figure 6.6 the rate of the discovered services is slightly increased at the 800 intergrid size. Meanwhile, it achieves the highest rate with the small intergrid sizes such as 100 and 200 nodes. In addition to that, the rate of discovered services also increases with intergrid size of 1000 nodes. This could possibly be because the created new classes have more number of nodes, which influences the rate of the local discovered services to be higher.

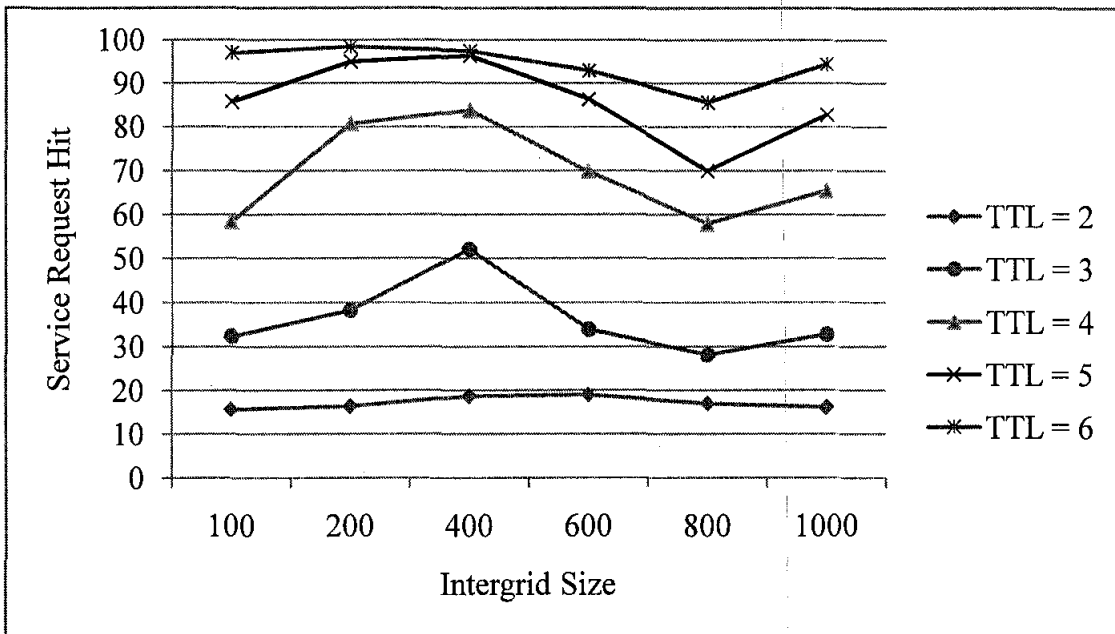


Figure 6.6 Discovered Services for generated requests equivalent to 100% of the intergrid size with different TTL values

All in all, it is observed that providing more TTL value causes the discovery of more services. However, one may argue that the increase of the TTL may inherit high traffic in the intergrid network. Nevertheless, in our case, the forwarding of service requests takes place only if the request has some semantic relation with the provider, if this not the case then the service request will be forwarded to all neighbors of the head node. Obviously, this will reduce the traffic in the intergrid system and the increase of the TTL value will not cause overhead on the network.

Our second point of discussion is on the service request response time of the proposed RD framework. In fact, we use the simulator timer to measure the time between the generation of service request by the requester node until when an answer is given to the requester node. For example, a node may generate a request at time 180000000 (simulation time) and a response may be given at the time of 180017503, therefore the response time is 17503 millisecond (ms). We calculated the average value of the response time in each set of generated service requests percentage. Tables 6.6 – 6.9 and figures 6.7 – 6.10 illustrate these values. It is apparent from figures 6.7 – 6.10 that the increase of service request generation will increase the response time.

This also happens when we increase TTL value. For example in Figure 6.7, the average response time for generated service request equivalent to 25% of intergrid size of 100 nodes and TTL value 5 is 33486ms. The value becomes considerably higher (35569ms) in figure 6.10 when the service request rate is equivalent to 100% of intergrid nodes. However, the increase of intergrid size does not affect the request response time much, as the curve of the response time fluctuates in all four figures (6.7-6.10).

Table 6.6 Average Response Time (RT) for generated requests equivalent to 25% of the intergrid size with different TTL values (2-5)

Intergrid Size	RT/TTL 2	RT/TTL 3	RT/TTL 4	RT/TTL 5
100	21982ms	27514ms	29983ms	33486ms
200	21873ms	25183ms	29104ms	31957ms
400	19790ms	21921ms	26960ms	26680ms
600	19308ms	22954ms	30787ms	34505ms
800	20058ms	24026ms	26912ms	28825ms
1000	16766ms	22780ms	29162ms	31242ms

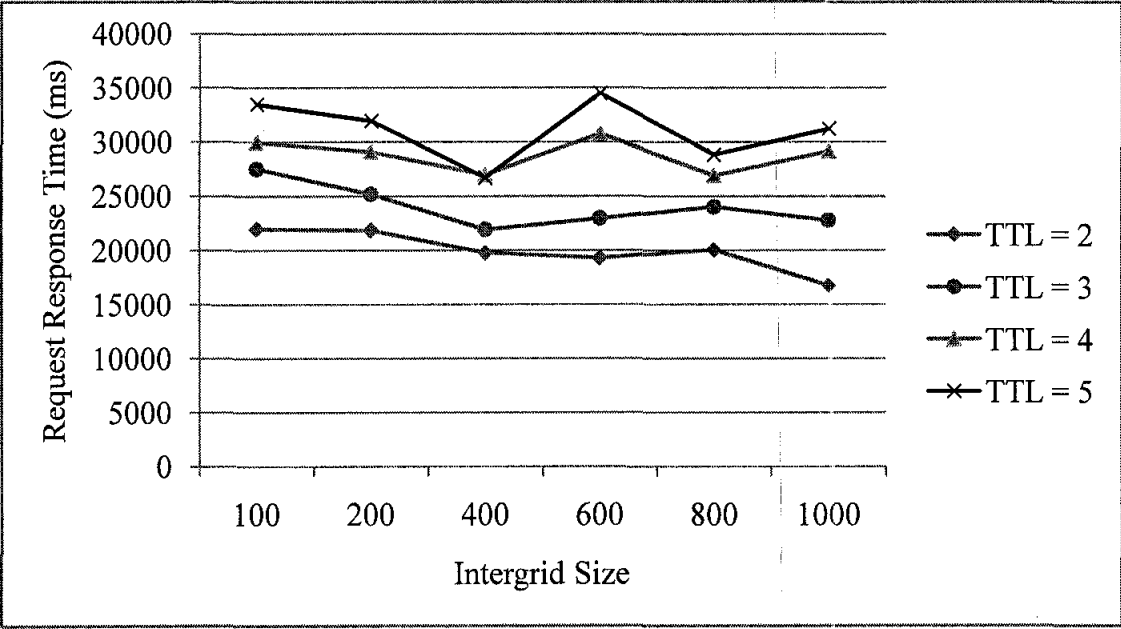


Figure 6.7 Service Request Response Time for generated requests equivalent to 25% of the intergrid size with different TTL values

Table 6.7 Average Response Time (RT) for generated requests equivalent to 50% of the intergrid size with different TTL values (2-5)

Intergrid Size	RT/TTL 2	RT/TTL 3	RT/TTL 4	RT/TTL 5
100	24916ms	27952ms	30975ms	35209ms
200	22098ms	23873ms	30039ms	29957ms
400	18806ms	23871ms	26255ms	27450ms
600	19122ms	21584ms	29640ms	31623ms
800	18215ms	23314ms	26816ms	29198ms
1000	16674ms	24054ms	31589ms	35752ms

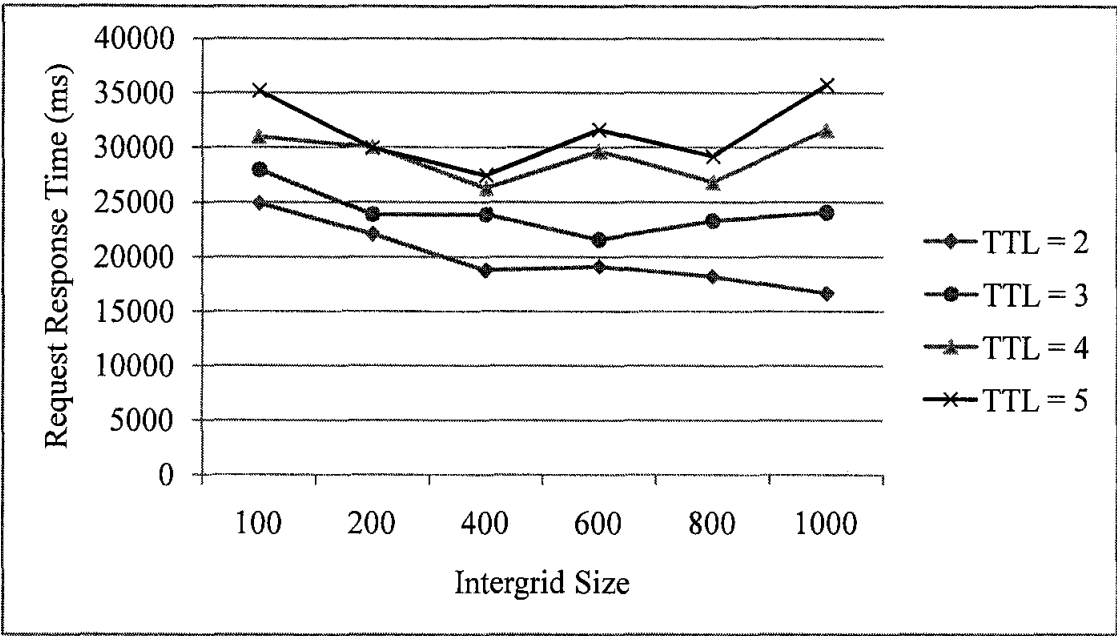


Figure 6.8 Service Request Response Time for generated requests that equivalent to 50% of the intergrid size with different TTL values

Table 6.8 Average Response Time (RT) for generated requests equivalent to 75% of the intergrid size with different TTL values (2-5)

Intergrid Size	RT/TTL 2	RT/TTL 3	RT/TTL 4	RT/TTL 5
100	24332ms	28128ms	31334ms	34390ms
200	21269ms	24054ms	30003ms	31107ms
400	18830ms	22325ms	26357ms	26881ms
600	18631ms	22136ms	29084ms	32979ms
800	19037ms	23308ms	25912ms	28767ms
1000	16624ms	21635ms	27743ms	30455ms

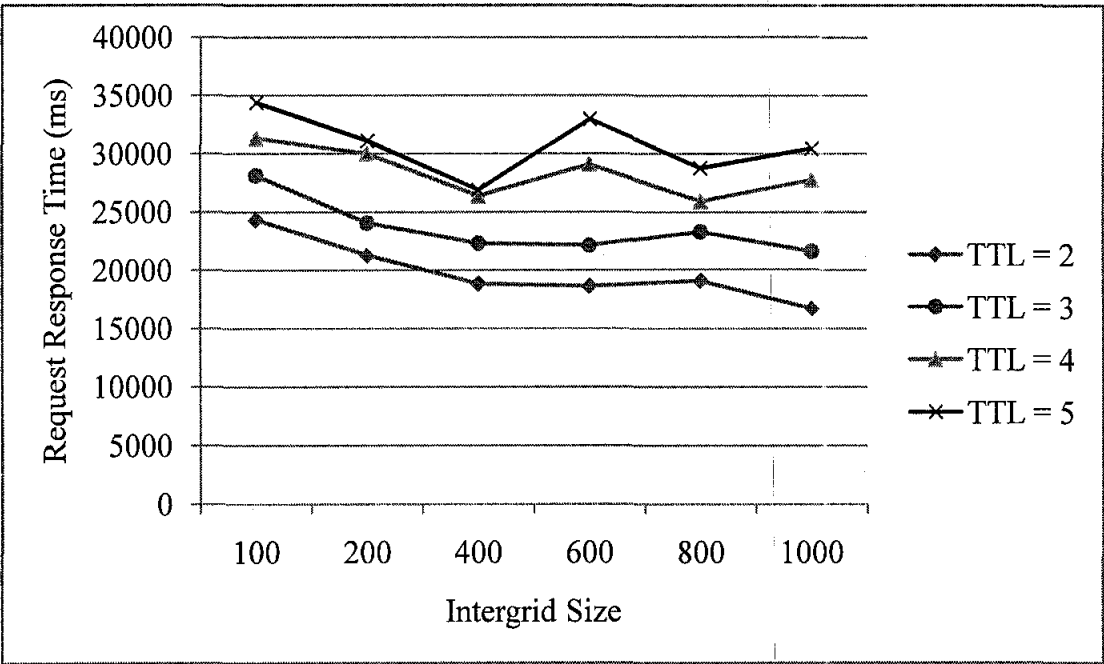


Figure 6.9 Service Request Response Time for generated requests equivalent to 75% of the intergrid size with different TTL values

Table 6.9 Average Response Time (RT) for generated requests equivalent to 100% of the intergrid size with different TTL values (2-5)

Intergrid Size	RT/TTL 2	RT/TTL 3	RT/TTL 4	RT/TTL 5
100	22823ms	28128ms	31063ms	35569ms
200	22034ms	24892ms	29529ms	31137ms
400	18730ms	23468ms	26769ms	27812ms
600	18936ms	22557ms	29485ms	32426ms
800	19036ms	21906ms	26385ms	28491ms
1000	18855ms	23763ms	30819ms	33176ms

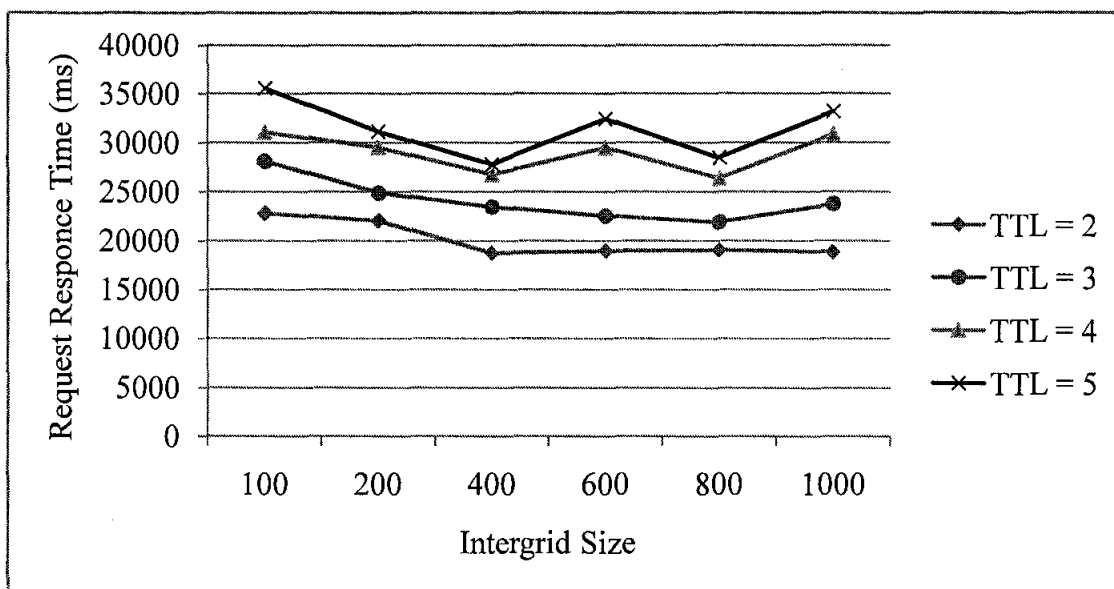


Figure 6.10 Service Request Response Time for generated requests equivalent to 100% of the intergrid size with different TTL values

Clearly, this indicates that the increase of the response time is not linearly related to the size of the intergrid nodes. This due to the decentralization of service requests processing as each head processes the service requests that are directed to it only. This ensures that the scale of the intergrid size will not cause performance degradation to the proposed RD system, which ensures sustainability of the system irrespective of the scale of the intergrid users as well as service grids.

Another aspect that is much related to the response time is the average number of hops that are crossed during the discovery process, which is supposed to be as low as possible with regard to the set TTL value. Tables 6.10 – 6.13 and figures 6.11- 6.14 show the average hops of the generated requests. Generally, the average hops values are slightly smaller than their respective defined TTL values, regardless of the number of generated service requests. For example the average hops in TTL 5 has a minimum value of 3.97, as shown in figure 6.11, for the intergrid size of 400 nodes and the request rate is 25%; then it fluctuates to 4.36 as the intergrid size is scaled up to 1000 . However, in all the cases the corresponding rate of the discovered services is good. Therefore, we deduce that having a TTL between 4 to 5 and an intergrid size of 400-1000 nodes will give an acceptable performance to our RD system. A further note on

the average hop values when the TTL value is 2 or 3 clearly indicate that the curve of the average hop is quite stable while scoring a poorer service request hits. This happens in the four cases (Figures 6.11 – 6.14). For example, in figure 6.14 the average hop value for TTL 2 starts with a value of 2.93 and maintains almost the same value to finally end with the value of 2.98 where the size of the intergrid is set to 1000 nodes. Therefore, we can deduce that our RD system can have good performance with TTL values such as 2 or 3 only if the number of concepts is reduced to three or four concepts and the intergrid size is limited to between 100-300 nodes.

Table 6.10 Average Hops (AH) for generated requests equivalent to 25% of the intergrid size with different TTL values (2-5)

Intergrid Size	AH/TTL 2	AH/TTL 3	AH/TTL 4	AH/TTL 5
100	2.83	3.5	3.82	4.3
200	3	3.52	4.05	4.45
400	3	3.45	4.06	3.97
600	3	3.47	4.02	4.52
800	3	3.58	4.03	4.44
1000	2.96	3.43	4.3	4.36

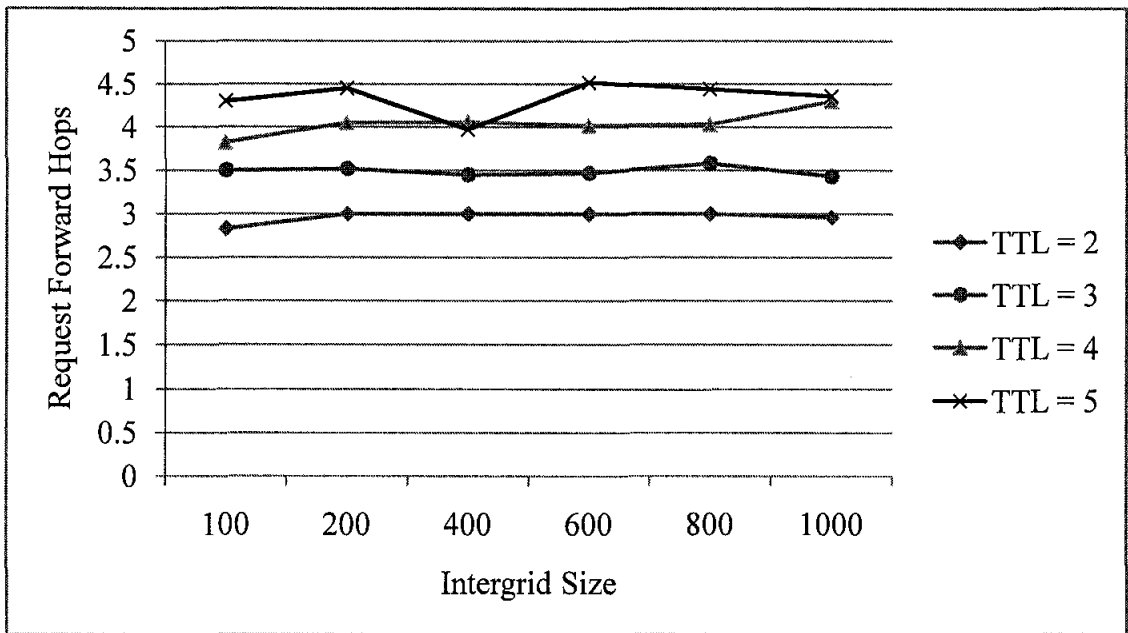


Figure 6.11 Average Hops for generated requests equivalent to 25% of the intergrid size with different TTL values

Table 6.11 Average Hops (AH) for generated requests equivalent to 50% of the intergrid size with different TTL values (2-5)

Intergrid Size	AH/TTL 2	AH/TTL 3	AH/TTL 4	AH/TTL 5
100	3	3.52	4.05	4.66
200	3	3.52	4.25	4.2
400	2.98	3.51	3.92	4.15
600	2.95	3.32	4.11	4.14
800	2.97	3.46	4.09	4.47
1000	2.96	3.51	4.28	4.81

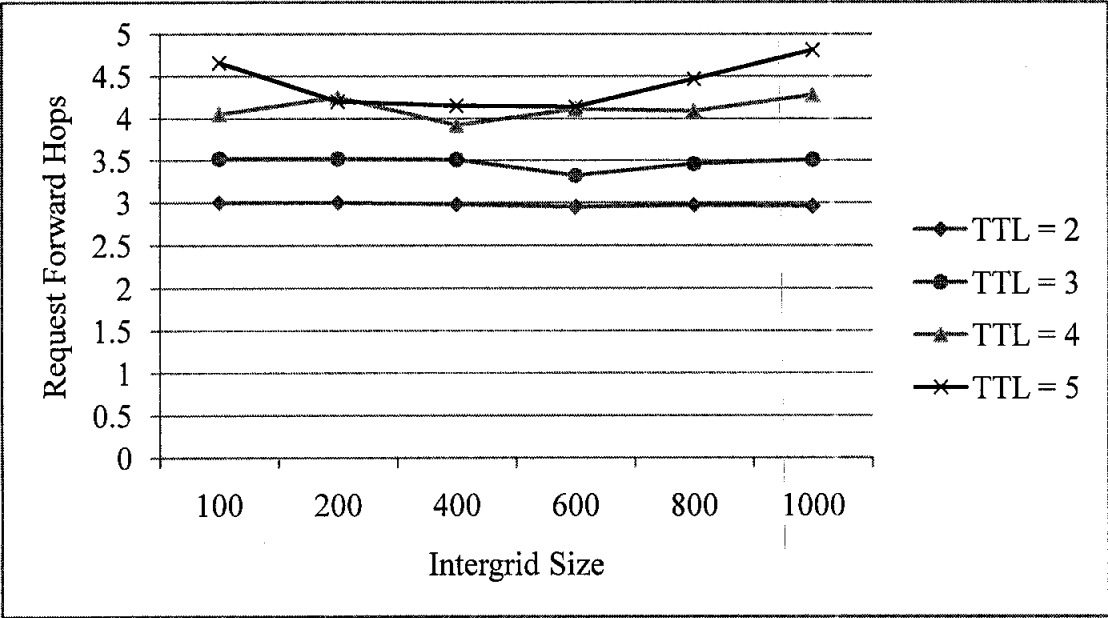


Figure 6.12 Average Hops for generated requests equivalent to 50% of the intergrid size with different TTL values

Table 6.12 Average Hops (AH) for generated requests equivalent to 75% of the intergrid size with different TTL values (2-5)

Intergrid Size	AH/TTL 2	AH/TTL 3	AH/TTL 4	AH/TTL 5
100	3	3.44	4.02	4.52
200	3	3.36	4.31	4.38
400	2.9	3.46	3.94	4.17
600	2.99	3.34	4.05	4.51
800	2.99	3.51	4.05	4.52
1000	2.97	3.53	4.2	4.61

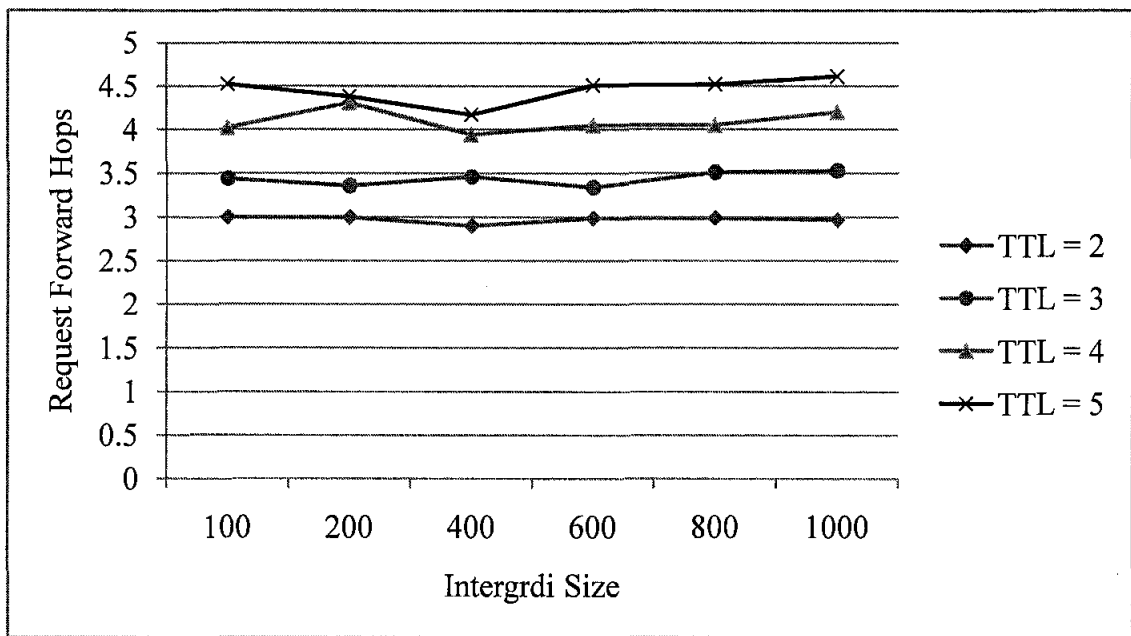


Figure 6.13 Average Hops for generated requests equivalent to 75% of the intergrid size with different TTL values

Table 6.13 Average Hops (AH) for generated requests equivalent to 75% of the intergrid size with different TTL values (2-5)

Intergrid Size	AH/TTL 2	AH/TTL 3	AH/TTL 4	AH/TTL 5
100	2.93	3.48	4.07	4.59
200	3	3.55	4.16	4.55
400	2.97	3.59	4.17	4.28
600	2.98	3.41	4.14	4.45
800	2.98	3.42	4.14	4.5
1000	2.98	3.46	4.22	4.65

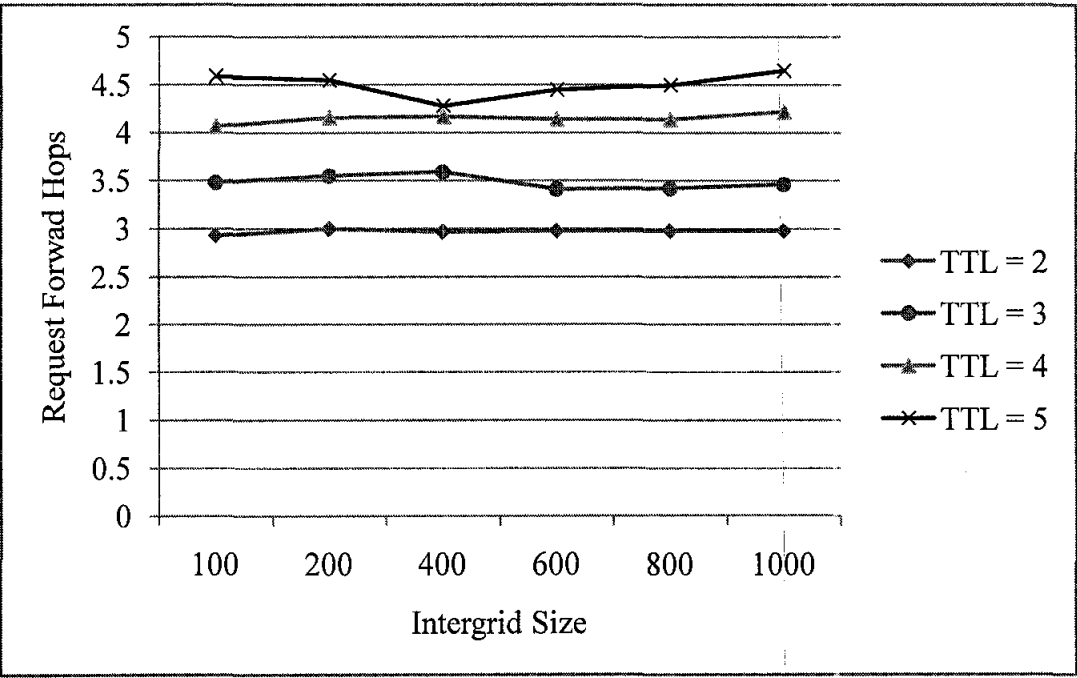


Figure 6.14 Average Hops for generated requests equivalent to 100% of the intergrid size with different TTL values

To further examine the performance of the proposed RD system, we conduct an additional experiment specifically to check dynamicity of the system with regard to node status. For this experiment, an intergrid size of 200 nodes with TTL value 5 has been selected as an stable intergrid size. During the simulation, while 75% of the nodes are generating their service requests, we ask the remaining 25% of the nodes to leave the network and rejoin it later, after 20 second (simulation time) of leaving the network. Then, these nodes will send their service requests (25%) to make the portion of the generated requests 100% of the intergrid size. A control experiment was also conducted i.e. by generating 100% service request of the same settings but without any portion of the nodes having to leave the network. Table 6.14 shows the result of the experiment as well as the result of the control experiment. In the column of intergrid size, the letter L marks the case where some nodes leave the intergrid, whereas the letter W denotes for when no node left the intergrid. From these results, it is noted that the RD system when 25% of the nodes left the network has achieved almost as much as the case when no node has left the system. In fact, the average response time and hops are even lesser. This is because the time interval between the

requests by the first active node and the newly joint nodes (the nodes that left the system before) allows the heads to respond faster than having all the requests simultaneously. The flexibility of the joining process and the fault tolerance mechanisms of the proposed RD enable it to sustain the dynamic nature of the intergrid system.

Table 6. 14 Performance of the proposed RD system in node fault condition

Intergrid Size	Service Request Hit	Average Response Time	Average Hops
200 L	94.91%	28887ms	4.21
200 W	94.94%	31137ms	4.55

With that, it is convincing that the proposed RD system is able to meet the performance requirements for the intergrid RD system. This includes scalability, decentralization and dynamism. From the service request hit rates obtained from different intergrid sizes, we can see that the proposed RD system can scale with the intergrid system as well. Decentralization feature of the proposed system has been proven by the response time, which did not show a linear dependency on the scale of the intergrid size. Lastly, the dynamism feature has been achieved by the fault tolerance mechanism.

6.8 Comparative Study

Since the aim of the study is to provide an advance progress beyond the state-of-art in this field, a comparative study to proof that is therefore needed, through which we can show the scientific advancement that has been made. Consequently, we compare the proposed RD system with the most promising scalable RDs that we have found in the literature. The most scalable RD systems are the super-peer based RDs (Mastroianni et al. 2005) and (Mastroianni et al. 2008) systems, which we have identified as the good candidate for the intergrid level. In fact, our RD system is also an extension of

the super-peer model with the addition of the semantic technology into the architecture and optimized discovery algorithm. Therefore, our comparative simulation is done by simulating the same system with and without the use of semantic technology. In the case of semantic technology, we have the implementation of class formulation, head appointment, and so on are based on the ontology, which represent the proposed RD system. Meanwhile, in case of without the semantic technology, we formulate the class, appoint the head and forward service request messages without involving the ontology, which is the case of the super-peer model.

In order to have a fair comparison between the two situations, we set the intergrid size in the range of 100 – 600 nodes as the stable range where the load balancing mechanism has no much effect on the performance, which will easy the discussion about the scalability of the systems. The random distribution of services to the nodes, the assignment of the number of services in any nodes, and the random generation of the service request for any given node are same in the two situations. The total number of service requests that should be generated by the nodes is equal to their sizes. Table 6.15 and figures 6.15 – 6.17 show the results of the two models in term of service request hit, average response time and average request forward hops.

Table 6.15 A comparison between the semantic super peer/RD and the super-peer model

RD system	Intergrid Size	Service Request Hit	Average Response Time	Average Hops
Super-Peer	100	55.10%	23919ms	3.33
	200	93.47%	30713ms	4.29
	400	90.97%	29072ms	4.29
	600	71.07%	30241ms	4.02
Semantic RD	100	85.71%	35569ms	4.59
	200	94.94%	31137ms	4.55
	400	96.44%	27812ms	4.28
	600	86.34%	32426ms	4.45

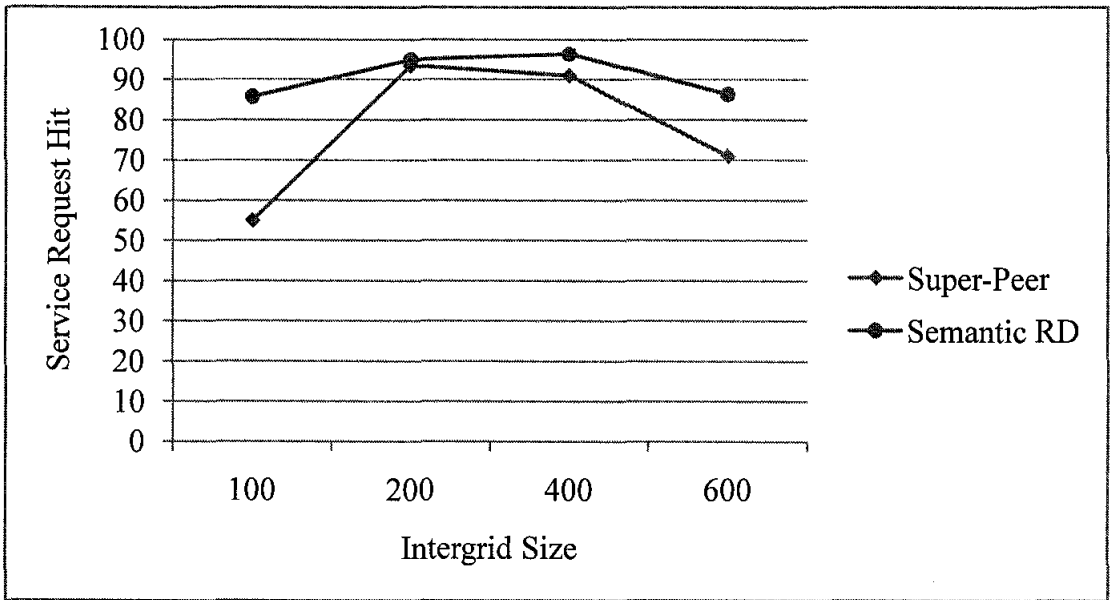


Figure 6.15 Discovered Services for generated requests equivalent to 100% of the intergrid obtained with the super-peer model and the semantic RD model

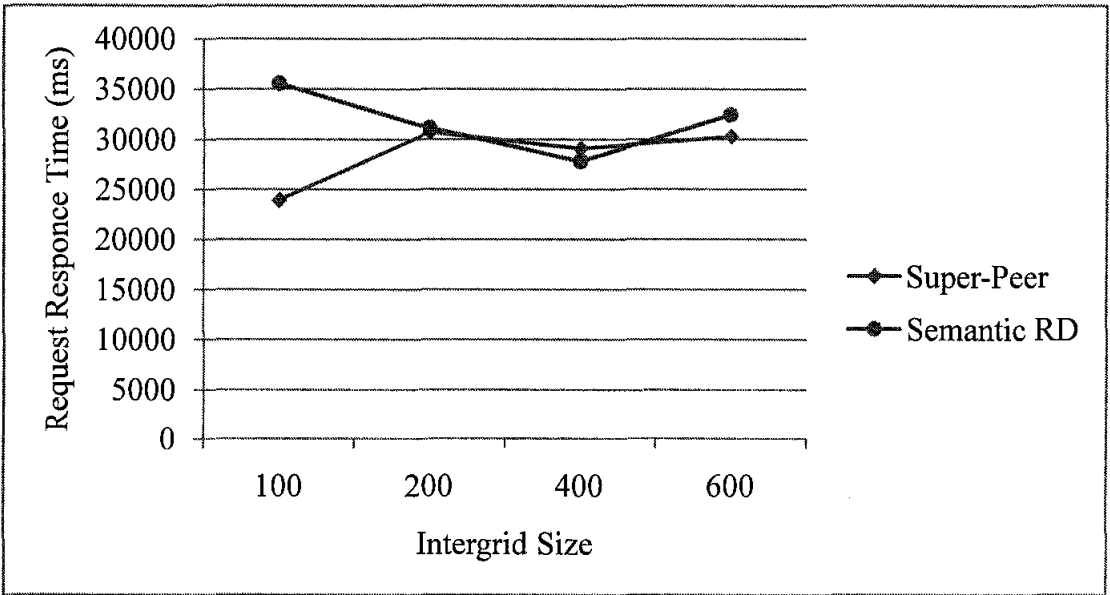


Figure 6.16 Average Response Time with the super-peer model and the semantic RD model

It is clear from figure 6.15 that the semantic RD system has a better request hit rate compared to the super-peer model in all the intergrid sizes. This is because in super-peer model the services in the classes are not organized in a particular relation, instead they are based on their joining time to the network, which makes it difficult to reach every node in the network. Meanwhile, the average response time of semantic RD model is also slightly higher most of time compared to the super-peer model. This because as the semantic RD model achieves high service request hit rate, it consumes more time. The average number of the hops of semantic is also a bit higher compared the super-peer. This is due to the discovery algorithm of the semantic RD, which optimizes the forwarding of messages in the network so that the service request can reach more nodes while scoring high service request rate.

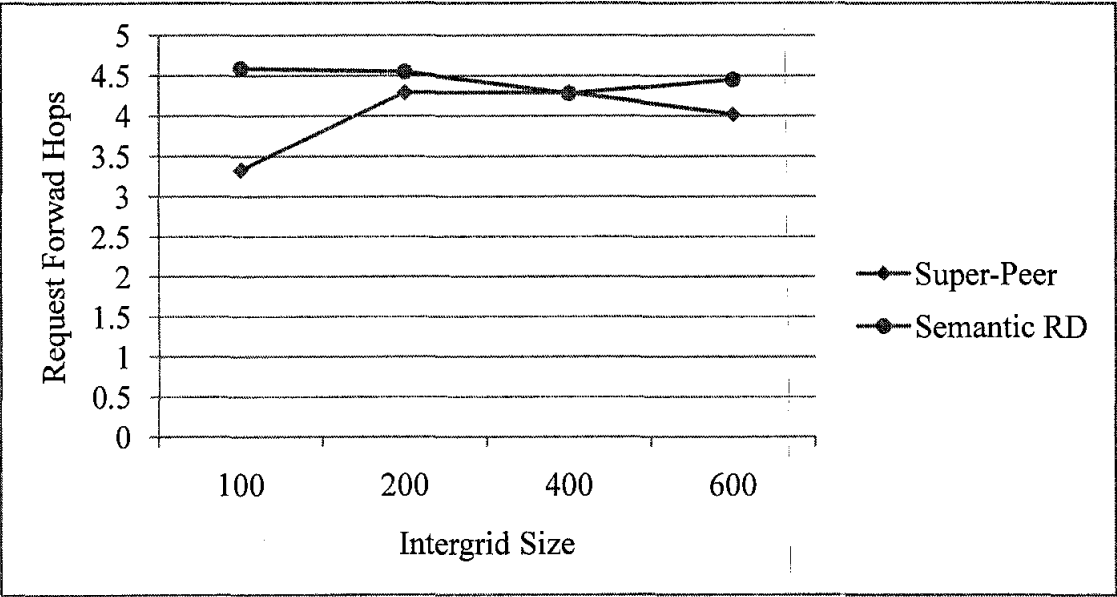


Figure 6.17 Average Hops obtained with the super-peer model and the semantic RD model

In short, based on the results of the comparative study on the intergrid of 100, 200, 400 and 600 nodes the semantic RD has a better performance than the super-peer model, but we cannot go as far as to generalize these findings because further investigation involving larger intergrid size than what we have used is needed.

CHAPTER 7

CONCLUSION

7.1 Introduction

This chapter concludes the thesis. Our aim has been to highlight the use of semantic technology in grid RD technology, and to develop a semantic-based scalable decentralized dynamic RD framework since it is very critical in intergrid system, which is the recent advancement in the grid technology. For this, we have presented in the first place an extensive review on the convergence of semantic technology and grid with focus on the RD part. More specifically, we have examined the current status of RD system using the semantic information, what have been achieved in meeting the recent grid technology requirements and the future outlook of this field. We then embarked on proposing the new RD framework. The framework has included: a conceptual model for semantic description that treats the small grids of the intergrid system as services (service grids) and their semantic representation has been based on that, a semantic registry architecture that specifies semantically the distribution of the service grids metadata directories and their management with regard to scalability and dynamism of the service grids metadata, and an agent based discovery algorithm that exploits the description model and the registry architecture to search and select the service grids on behalf of the intergrid user. We have shown the effectiveness of the framework through some discussions and analysis, and an extensive simulation work which has confirmed the effectiveness of the framework. In the following sections we summarize the contributions from this work, discuss the limitations of the work and recommend some future work.

7.2 Contributions

The main contributions of the thesis are as follows:

- A systematic review study on the convergence of grid technology and semantic technology.

This has included a discussion on the current semantic technology tools that have been used in the grid RD studies (semantic-based RD systems) and what we think will be useful for the grid technology in the future, key taxonomies for the semantic-based grid RD studies that have been based on the implemented semantic technology, an extensive qualitative evaluation and analysis of the semantic-based RD studies, a discussion on the future applications of the semantic-based RD studies in the Emerging Grids and Cloud systems. With this we have answered our first research question.

- An interoperable semantic description RD component model for the intergrid services metadata representation.

The model initially refined the architecture of the intergrid system by treating the overall grid system as service. The model introduced ontology as information model for services representation, a formal service request formulation that is known as goal-based service request formulation. The model also highlighted the available tools to implement the semantic service metadata creation and service request formulation, and the formal scenario that allows the model to work with the current related grid information services by not posing to them an additional overhead. Hence, we have answered our second research question.

- A semantic distributed registry architecture for indexing the service metadata for the discovery.

The architecture used super-peer model as an infrastructure to provide a decentralized set of registries, introduced ontology as formal criteria for the distribution of the registries. The architecture provided all the algorithms to build the registries. This included the distribution scenario of intergrid nodes into classes based on the semantic relation of the services that they provide, which is known as the class formulation mechanism and all the related mechanisms for

maintaining the architecture such as, the subscription of new nodes, the fault tolerance and load balancing. Thus, we have answered our third research question.

- An agent-based service search and selection algorithm.

The algorithm introduced intelligent agents as tools to hide the direct interaction of user with the entire proposed RD framework. Thus, the algorithm used two agents, one is to track the status of the services in the nodes and the second is to formulate the service request. The algorithm uses the semantic description of the services and the semantic registry architecture to search and select the specified service request by the user in an optimized manner. Hence, we have answered the fourth research question.

- An extensive experimental performance evaluation of the proposed RD framework.

A set of simulations have been conducted, which examined the performance of the new system by using the related performance metrics such as the response time and service request hit. The result and discussion have confirmed that our new framework has contributed some advancement in this field. By this, we have answered the last research question.

7.3 Limitations and Future Work

As the nature of knowledge, every work has to have some limitations to ensure the future research continuation in this field. Therefore, we identify the limitations of this work as follows:

- The presented review study on the semantic-based grid RD system does not include a quantitative evaluation and analysis due to the unavailability of the required tools, therefore further quantitative studies are needed.
- The proposed semantic description model for the service grid metadata creation and representation is yet to be implemented in a real intergrid system so that it can be compared quantitatively with other related studies. Therefore, further

- collaboration with the information service communities on the creation of the ontologies is recommended.
- We have simulated the proposed work as well as the comparative study for an intergrid system with size of 1000 nodes as the maximum size due to limitation of the computing platform resource. Therefore, further work to extend the simulator to work in grid environment is required so that additional analysis on the proposed system can be obtained.

REFERENCES

- Andreasen T, Bulskov H, and Knappe R. 2003. From ontology over similarity to query evaluation. In: Bernardi R, and Moortgat M, editors. 2nd CoLogNET-ElsNET Symposium - Questions and Answers: Theoretical and Applied Perspectives. Amsterdam, Holland. p 39-50.
- Andreozzi S, Bortoli ND, Fantinel S, Ghiselli A, Tortone G, and Vistoli C. 2005. GridICE: a monitoring service for Grid systems. *Future Generation Computer Systems* 21(4):559-571.
- Andreozzi S, Burke S, Donno F, Field L, Fisher S, Jensen J, Konya B, Litmaath M, Mambelli M, Schopf JM and others. 2007. GLUE Schema Specification version 1.3. GLUE Working Group. Report nr 14185.
- Andreozzi S, Burke S, Ehm F, Field L, Galang G, Horat D, Konya B, Litmaath M, Millar P, and Navarro J. 2009 -a. GLUE v. 2.0.1 - Reference Realizations to SQL Schema. GLUE Working Group. Report nr 15517.
- Andreozzi S, Burke S, Ehm F, Field L, Galang G, Konya B, Litmaath M, Millar P, and Navarro J. 2009 -b. GLUE v. 2.0 - Reference Realization to XML Schema. GLUE Working Group Report nr 15515.
- Andreozzi S, Burke S, Ehm F, Field L, Horat D, Konya B, Litmaath M, Millar P, and Navarro J. 2009 -c. GLUE v. 2.0.1 - Reference Realizations to LDAP Schema. GLUE Working Group. Report nr 5526.
- Andrzejak A, and Xu Z. 2002. Scalable, Efficient Range Queries for Grid Information Services. *Proceeding 2nd International Conference on Peer-to-Peer Computing (P2P 2002)*. p 33-40.
- Asadzadeh P, Buyya R, Kei CL, Nayar D, and Venugopal S. 2005. Global Grids and Software Toolkits: A Study of Four Grid Middleware Technologies. In: Yang L, and Guo M, editors. *High Performance Computing: Paradigm and Infrastructure*. New Jersey, USA: Wiley Press
- Assuncao MD, Buyya R, and Venugopal S. 2008. InterGrid: a case for internetworking islands of Grids. *Concurrency and Computation: Practice & Experience* 20(8):997-1024.

- Baker M, Buyya R, and Laforenza D. 2000. The Grid: International Efforts in Global Computing. Proceedings of the International Conference on Advances in Infrastructure for Electronic Business, Science, and Education on the Internet (SSGRR 2000). Aquila, Rome, Italy.
- Baker MA, and Smith GC. 2003. GridRM: an extensible resource monitoring system. Proceedings of the IEEE International Cluster Computing Conference. p 207-214.
- Balaton Z, and Gombás G. 2004. Resource and Job Monitoring in the Grid Lecture Notes in Computer Science (LNCS): Springer Berlin p404-411.
- Balis B, Bubak M, Szeplieniec T, Wismüller R, and Radecki M. 2004. Monitoring grid applications with grid-enabled OMIS monitor. Grid Computing: Springer Berlin / Heidelberg. p 558-567
- Bechhofer S, and Goble C. 2001. Towards Annotation using DAML+OIL. K-CAP 2001 Workshop on Knowledge Markup and Semantic Annotation. Victoria B.C-Canada: ACM.
- Berman F, Fox G, and Hey T. 2003. Grid Computing Making the Global Infrastructure a Reality. Hutchison D, editor. West Sussex: John Wiley & Sons Ltd.
- Berners-Lee T, Hendler J, and Lassila O. 2001. The Semantic Web. Scientific American 284(4):34-43.
- Bharambe AR, Agrawal M, and Seshan S. 2004. Mercury: supporting scalable multi-attribute range queries. Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications. Portland, Oregon, USA. p 353-366.
- Bonnassieux F, Harakaly R, and Primet P. 2002. MapCenter: an open grid status visualization tool. Proceedings of the ISCA 15th International Conference on Parallel and Distributed Computing Systems. Louisville, KY, USA.
- Brocco A, Malatras A, and Hirsbrunner B. 2010. Enabling efficient information discovery in a self-structured grid. Future Generation Computer Systems 26(6):838-846.
- Buyya R, and Murshed M. 2002. Gridsim: A toolkit for the modeling and simulation of distributed management and scheduling for Grid computing. Concurrency and Computation: Practice and Experience 14:13-15.
- Buyya R, Yeo CS, Venugopal S, Broberg J, and Brandic I. 2009. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. Future Generation Computer Systems 25(6):599-616.

- Cai M, Frank M, Chen J, and Szekely P. 2003. MAAN: A Multi-Attribute Addressable Network for Grid Information Services. Proceeding 4th International Workshop on Grid Computing (GRID 2003). p 184-191.
- Chandrasekaran B, Josephson JR, and Benjamins VR. 1999. What are ontologies, and why do we need them? IEEE Intelligent Systems 14(1):20-26.
- Chao-Tung Y, Wen-Jen H, and Kuan-Chou L. 2009. A Resource Broker with Cross Grid Information Services on Computational Multi-grid Environments. Proceedings of the 9th International Conference on Algorithms and Architectures for Parallel Processing. Taipei, Taiwan: Springer-Verlag.
- Chervenak A, Foster I, Kesselman C, Salisbury C, and Tuecke S. 2000. The data grid: Towards an architecture for the distributed management and analysis of large scientific data sets. Journal of Network and Computer Applications 23(3):187-200.
- Cooke A, Gray AJG, Ma L, Nutt W, Magowan J, Oevers M, Taylor P, Byrom R, Field L, Hicks S and others. 2003. R-GMA: An Information Integration System for Grid Monitoring On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE: Springer Berlin / Heidelberg. p 462-481.
- Corcho O, Alper P, Kotsiopoulos I, Missier P, Bechhofer S, and Goble C. 2006. An overview of S-OGSA: A Reference Semantic Grid Architecture. Web Semantics: Science, Services and Agents on the World Wide Web 4(2):102-115.
- Crespo A, and Garcia-Molina H. 2002. Routing Indices for Peer-to-Peer Systems. Proceedings of 22nd International Conference on Distributed Computing Systems (ICDCS'02). p 23-30.
- Czajkowski K, and D.F. Ferguson, I. Foster, J. Frey, S. Graham, I. Sedukhin, D. Snelling, S. Tuecke, W. Vambenepe. 2004. The WS-Resource Framework. <http://www.globus.org/wsrf/>.
- Dinda P, Gross T, Karrer R, Lowekamp B, Miller N, Steenkiste P, and Sutherland D. 2001. The Architecture of the Remos System. Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing: IEEE Computer Society. p 252-265.
- Edwards WK. 2006. Discovery systems in ubiquitous computing. IEEE Pervasive Computing 5(2):70-77.
- Erl T. 2005. Service-Oriented Architecture (SOA): Concepts, Technology, and Design Crawfordsville, Indiana Prentice Hall
- Fitzgerald S, Foster I, Kesselman C, Laszewski Gv, Smith W, and Tuecke S. 1997 A directory service for configuring high-performance distributed computations. 6th IEEE Symposium on High Performance Distributed Computing: IEEE Computer Society Press. p 365-375.

- Flahive A, Taniar D, Rahayu W, and Apduhan BO. 2009. Ontology tailoring in the Semantic Grid. *Computer Standards & Interfaces* 31(5):870-885
- Foster I, and Kesselman C. 1997. Globus: A Metacomputing Infrastructure Toolkit. *International Journal of High Performance Computing Applications* 11(2):115-128.
- Foster I, and Kesselman C. 2003. *The Grid 2: Blueprint for a New Computing Infrastructure*. San Francisco: Morgan Kaufman.
- Foster I, Kesselman C, and Tuecke S. 2001. The Anatomy of the Grid Enabling Scalable Virtual Organizations. *International Journal Supercomputer Applications* 15(3):200-222.
- Foster I, Kishimoto H, Savva A, Berry D, Djaoui A, Grimshaw A, Horn B, Maciel F, Siebenlist F, Subramaniam R and others. 2005. The Open Grid Services Architecture, Version 1.0. Open Grid Services Architecture WG. Report nr GFD-I.030.
- Garcia P, Pairot C, Mondejar R, Pujol J, Tejedor H, and Rallo R. 2005. PlanetSim: A New Overlay Network Simulation Framework. 123-136 p.
- Garzoglio G, Levshina T, Andreozzi S, Reddy S, Mambelli M, Roy A, Wang S, and Wenaus T. 2008. GLUE Schema v1.2 / v1.3 Mapping to Old ClassAd Format (v1.1).
- Groleau W, Vlassov V, and Popov K. 2007. Towards Semantics-Based Resource Discovery for the Grid. *Integrated Research in GRID Computing*: springerlink. p 175-187.
- Gruber TR. 1995. Toward principles for the design of ontologies used for knowledge sharing. *International Journal of Human-Computer Studies* 43(5-6):907-928.
- Han L, and Berry D. 2008. Semantic-supported and agent-based decentralized grid resource discovery. *Future Generation Computer Systems* 24 (8):806-812.
- Henri B, Raoul B, Rutger H, Cerial J, Thilo K, Jason M, Rob van N, John R, Luc R, Tim R and others. 2000. The distributed ASCI Supercomputer project. *SIGOPS Oper Syst Rev* 34(4):76-96.
- Henri C, Arnaud L, and Martin Q. 2008. SimGrid: A Generic Framework for Large-Scale Distributed Experiments. *Proceedings of the Tenth International Conference on Computer Modeling and Simulation*: IEEE Computer Society.
- Horrocks I, Fensel D, Broekstra J, Decker S, Erdmann M, Goble C, Harmelen FV, Klein M, Staab S, Studer R and others. 2000. The Ontology Inference Layer OIL.

- Howell F, and Ross M. 1998. Simjava: a discrete event simulation package for Java with applications in computer systems modeling. . First International Conference on Web-based Modeling and Simulation. San Diego CA: Society for Computer Simulation.
- Iamnitchi A, and Foster I. 2004. A peer-to-peer approach to resource location in Grid environments. Grid resource management: state of the art and future trends: Kluwer Academic Publishers. p 413-429.
- Ivan R, Francesc G, Julita C, Liana F, and Sadjadi SM. 2010. Grid broker selection strategies using aggregated resource information. Future Generation Computer Systems 26(1):72-86.
- Jacek K, Tomas V, Carine B, and Joel F. 2007. SAWSDL: Semantic Annotations for WSDL and XML Schema. IEEE Internet Computing 11(6):60-67.
- James F, Todd T, Miron L, Ian F, and Steven T. 2001. Condor-G: A Computation Management Agent for Multi-Institutional Grids. Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing: IEEE Computer Society.
- Jennings NR. 2001. An agent-based approach for building complex software systems. Communications of the ACM archive 44 (4):35 - 41.
- Jha S, Merzky A, and Fox G. 2009. Using Clouds to Provide Grids Higher-Levels of Abstraction and Explicit Support for Usage Modes. Concurrency and Computation:Practice and Experience 21(3):1087-1108
- Kashani FB, Chen CC, and Shahabi C. 2004. WSPDS:Web services Peer-to-Peer discovery service. In Proceedings International Conference on Internet Computing (IC '04).
- Kertész A, and Kacsuk P. 2007. Grid Meta-Broker Architecture: Towards an Interoperable Grid Resource Brokering Service. Euro-Par 2006: Parallel Processing. p 112-115.
- Kertész A, and Kacsuk P. 2010. GMBS: A new middleware service for making grids interoperable. Future Generation Computer Systems 26(4):542-553.
- Kou Y, Yu G, Shen D, Li D, and Nie T. 2007. PS-GIS: personalized and semantics-based grid information services. Proceedings of the 2nd international conference on Scalable information systems. Suzhou, China: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).
- Kovacevic A, Kaune S, Mukherjee P, Liebau N, and Steinmetz R. 2007. Benchmarking Platform for Peer-to-Peer Systems , vol. 46, no. 3, 2007. . it - Information Technology (Methods and Applications of Informatics and Information Technology) 49(5):312-319.

- Kurdi H, Li M, and Al-Raweshidy H. 2008. A Classification of Emerging and Traditional Grid Systems. *IEEE Distributed Systems Online* 9(3):1.
- Lamnitchi AL. 2003. Resource Discovery in Large Resource-Sharing Environments. Illinois: The University of Chicago. 1-1 p.
- Li J, and Vuong S. 2005. Grid Resource Discovery Using Semantic Communities. *Grid and Cooperative Computing - GCC 2005: Springer Berlin / Heidelberg*. p 657-667.
- Li J, and Vuong S. 2005 A Scalable Semantic Routing Architecture for Grid Resource Discovery. *Proceedings of the 11th International Conference on Parallel and Distributed Systems IEEE Computer Society* p29-35
- Li J, and Vuong S. 2006. Grid resource discovery based on semantic P2P communities. *Proceedings of the 2006 ACM symposium on Applied computing*. Dijon, France: ACM. p 754-758.
- Lua K, Crowcroft J, Pias M, Sharma R, and Lim S. 2004. A survey and comparison of peer-to-peer overlay network schemes. *Communications Surveys and Tutorials* 7(2):72.
- Ludwig SA, and Reyhani SMS. 2005. Introduction of semantic matchmaking to grid computing. *Journal of Parallel and Distributed Computing* 65(12):1533-1541
- Ludwig SA, and Reyhani SMS. 2006. Semantic approach to service discovery in a Grid environment. *Web Semantics: Science, Services and Agents on the World Wide Web* 4(1):1-13
- Maciel FB. 2008. Guidelines for Information Modeling for OGSA Entities. *Open Grid Forum*.
- Marzolla M, and Mordacchini M. 2005. Resource discovery in a dynamic Grid environment. *16th International Workshop on Database and Expert Systems Applications (DEXA'05)*.
- Marzolla M, Mordacchini M, and Orlando S. 2007. Peer-to-peer systems for discovering resources in a dynamic grid. *Parallel Computing* 33(4-5):339-358.
- Massie ML, Chun BN, and Culler DE. 2004. Ganglia Distributed Monitoring System: Design, Implementation, and Experience. *Parallel Computing* 30(7): 817-840.
- Mastroianni C, Talia D, and Verta O. 2005. A super-peer model for resource discovery services in large-scale Grids. *Future Generation Computer Systems* 21(8):1235-1248.
- Mastroianni C, Talia D, and Verta O. 2008. Designing an information system for Grids: Comparing hierarchical, decentralized P2P and super-peer models. *Parallel Computing* 34(10):593-611

- McGuinness DL, Fikes R, Hendler J, and Stein LA. 2002. DAML+OIL: An Ontology Language for the Semantic Web. *IEEE Intelligent Systems* 17(5):72-80.
- Meshkova E, Riihijärvi J, Petrova M, and Mähönen P. 2008. A survey on resource discovery mechanisms, peer-to-peer and service discovery frameworks. *Computer Networks* 52 (11):2097–2128.
- Michael PP, Paolo T, Schahram D, and Frank L. 2007. Service-Oriented Computing: State of the Art and Research Challenges. *IEEE Computer* 40(11):38-45.
- Miller B, Callaghan M, Cargille J, Hollingsworth J, Irvin R, Karavanic K, Kunchithapadam K, and Newhall T. 1995. The Paradyn Parallel Performance Measurement Tool. *IEEE Computer* 28(11):37-46.
- Nejdl W, Wolpers M, Siberski W, Schmitz C, Schlosser M, Brunkhorst I, and Loser A. 2003. Super-peer-based routing and clustering strategies for RDF-based peer-to-peer networks. *Proceedings of the 12th International Conference on World Wide Web WWW'03*. p 536–543.
- Newman HB, Legrand IC, Galvez P, Voicu R, and Cirstoiu C. 2003. MonALISA: a distributed monitoring service architecture. *Computing in High Energy and Nuclear Physics (CHEP03)* La Jolla, California.
- Oppenheimer D, Albrecht J, Patterson D, and Vahdat A. 2004. Scalable wide-area resource discovery. Univ. of California.
- Padmanabhan A. 2006. SOG: A Self-Organized Grouping Infrastructure for Grid Resource discovery [PhD Thesis]. Iowa: University of Iowa. 5 p.
- Parkin M, Burghe Svd, Corcho O, Snelling D, and Brooke J. 2006. The Knowledge of the Grid: A Grid Ontology. 6th Cracow Grid Workshop. Poland: Cracow.
- Pastore S. 2008. The service discovery methods issue: A web services UDDI specification framework integrated in a grid environment. *Journal of Network and Computer Applications* 31(2):93-107.
- Perez A, Sanchez A, and Abawajy J. 2009. An agent architecture for managing data resources in a grid environment. *Future Generation Computer Systems* 25 (7):747-755.
- Pernas AM, and Dantas MAR. 2005. Using Ontology for Description of Grid Resources. *Proceedings of the 19th International Symposium on High Performance Computing Systems and Applications (HPCS'05)*: IEEE Society p223-229
- Pulido JRG, Ruiz MAG, Herrera R, Cabello E, Legrand S, and Elliman D. 2006. Ontology languages for the semantic web: A never completely updated review. *Knowledge-Based Systems* 19 (7):489-497.

- Puppin D, Moncelli S, Baraglia R, Tonelotto N, and Silvestri F. 2005. A Grid Information Service Based on Peer-to-Peer. Lecture Notes in Computer Science (LNCS): Springer-Verlag. p 454-464.
- Rajiv R, Lipo C, Aaron H, Shanika K, and Rajkumar B. 2007. Decentralised Resource Discovery Service for Large Scale Federated Grids. Proceedings of the Third IEEE International Conference on e-Science and Grid Computing: IEEE Computer Society.
- Ranjan R, Harwood A, and Buyya R. 2008. Peer-to-Peer Based Resource Discovery in Global Grids A Tutorial IEEE Communications Surveys & Tutorials 10(2):6-33.
- Ratnasamy S, Francis P, Handley M, Karp RM, and Shenker S. 2001. A Scalable Content-Addressable Network. Proceedings of ACM SIGCOMM 2001 Conference-- on Applications, Technologies, Architectures, and Protocols for Computer Communication. p 161-172.
- Ratnasamy S, Hellerstein JM, and Shenker S. 2003. Range queries over DHTs. Intel Corporation.
- Resnik P. 1999. Semantic Similarity in a Taxonomy An Information-Based Measure and its Application to Problems of Ambiguity in Natural Language. Journal of Artificial Intelligence Research 11 95-130.
- Ribler RL, Vetter JS, Simitci H, and Reed DA. 1998. Autopilot: Adaptive Control of Distributed Applications. Proceedings of the 7th IEEE International Symposium on High Performance Distributed Computing: IEEE Computer Society. p 172-179.
- Rodr MA, guez, and Max JE. 2003. Determining Semantic Similarity among Entity Classes from Different Ontologies. IEEE Trans on Knowl and Data Eng 15(2):442-456.
- Rowstron A, and Druschel P. 2001. Pastry: Scalable, Decentralized Object Location and Routing for Large Scale Peer-to-Peer Systems. Proceeding IFIP/ACM International Conference on Distributed Systems Platforms LNCS, Springer. p 329-350.
- Ruay-Shiung C, and Min-Shuo H. 2010. A resource discovery tree using bitmap for grids. Future Generation Computer Systems 26(1):29-37.
- Said MP, and Kojima I. 2009. S-MDS: Semantic Monitoring and Discovery System. Journal of Grid Computing 7(2):205-224.
- Schmidt C, and Parashar M. 2003. Flexible Information Discovery in Decentralized Distributed Systems. Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing: IEEE Computer Society. p 226-235.

- Schoder D, Fischbach K, and Schmitt C. 2005 Peer-to-Peer Computing: The Evolution of a Disruptive Technology. Hershey, USA: Idea Group Publishing. 1-27 p.
- Schopf JM, Pearlman L, Miller N, Kesselman C, Foster I, D'Arcy M, and Chervenakand A. 2006. Monitoring the grid with the Globus Toolkit MDS4. *Journal of Physics: Conference Series* 46(1):521.
- Sheila AM, Tran Cao S, and Honglei Z. 2001. Semantic Web Services. *IEEE Intelligent Systems* 16(2):46-53.
- Shen H. 2009. A P2P-based intelligent resource discovery mechanism in Internet-based distributed systems. *Journal of Parallel and Distributed Computing* 69(2):197-209.
- Smith W. 2002. A System for Monitoring and Management of Computational Grids. *Proceedings of the 2002 International Conference on Parallel Processing (ICPP2002)* IEEE Computer Society. p 55
- Solomon M. 2004. The ClassAd Language Reference Manual Version 2.4.
- Somasundaram TS, Balachandar RA, Kandasamy V, Buyya R, Raman R, Mohanram N, and Varun S. 2006. Semantic-based Grid Resource Discovery and its Integration with the Grid Service Broker. *International Conference on Advanced Computing and Communications (ADCOM 2006)* p84-89.
- Spence D, and Harris T. 2003. XenoSearch: Distributed Resource Discovery in the XenoServer Open Platform. *Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing: IEEE Computer Society.* p 216-225.
- Srikumar V, Rajkumar B, and Lyle W. 2006. A Grid service broker for scheduling e-Science applications on global data Grids: Research Articles. *Concurr Comput : Pract Exper* 18(6):685-699.
- Stelling P, Foster I, Kesselman C, Lee C, and Laszewski Gv. 1999. A fault detection service for wide area distributed computations *Cluster Computing* 2(2):1386-7857 (Print) 1573-7543 (Online).
- Steven F. 2001. Grid Information Services for Distributed Resource Sharing. *Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing: IEEE Computer Society.*
- Stokes EJ, Andreozzi S, Drescher M, and Savva A. 2008. Information and Data Modeling in OGSA Grids. *Open Grid Forum.*
- Subramaniam R, Nakata T, Itoh S, Oyanagi Y, Takefusa A, Anzaki T, Mizoguchi K, Tazaki H, Mori T, Suzuki T and others. 2009. Guidelines of Requirements for Grid Systems v1.0. *Open Grid Forum.*

- Talia D, and Trunfio P. 2005. Peer-to-Peer protocols and Grid services for resource discovery on Grids. In: Grandinetti L, editor. *Grid Computing: The New Frontier of High Performance Computing* Elsevier Science.
- Tarricone L, and Esposito A. 2004. *Grid Computing for Electromagnetics*. London: Artech House. 88 p.
- Tierney B, Aydt R, Gunter D, Smith W, Swany M, Taylor V, and Wolski R. 2002 *A Grid Monitoring Architecture*. Global Grid Forum.
- Tierney B, Crowley B, Gunter D, Lee J, and Thompson M. 2001. A Monitoring Sensor Management System for Grid Environments. *Cluster Computing* 4(1):19-28.
- Tierney B, and Gunter D. 2003. NetLogger: a toolkit for distributed system performance tuning and debugging. *Proceedings of the IFIP/IEEE Eighth International Symposium on Integrated Network Management: IEEE Xplore*. p 97-100.
- Trunfio P, Talia D, Papadakis H, Fragopoulou P, Mordacchini M, Pennanend M, Popove K, Vlassov V, and Haridi S. 2007. Peer-to-Peer resource discovery in Grids: Models and systems. *Future Generation Computer Systems* 23(7):864-878.
- Truong HL, and Fahringer T. 2004. *SCALEA-G: A Unified Monitoring and Performance Analysis System for the Grid* Springer Berlin / Heidelberg. p 202-211.
- Tuttle S, Ehlenberger A, Gorthi R, Leiserson J, Macbeth R, Owen N, Ranahandola S, Storrs M, and Yang C. 2004. *Understanding LDAP Design and Implementation*: IBM.
- Weiss G. 1999. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence* The MIT Press
- Wolski R, Spring N, and Hayes J. 1999. The network weather service: a distributed resource performance forecasting service for metacomputing. *Future Generation Computer Systems* 15(5-6):757-768.
- Wooldridge M. 2006. *An Introduction to Multi-Agent Systems*. Chichester: John Wiley and Sons Limited.
- Xing W, Corcho O, Goble C, and Dikaiakos M. 2007. *Active Ontology: An Information Integration Approach for Highly Dynamic Information Sources* Europe Semantic Web Conference 2007 (ESWC-2007). Innsbruck, Austria.

- Xing W, Corcho O, Goble C, and Dikaiakos MD. 2010. An ActOn-based semantic information service for Grids. *Future Generation Computer Systems* 26(3):324-336.
- Xing W, Dikaiakos MD, and Sakellariou R. 2006. A Core Grid Ontology for the Semantic Grid. *Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid: IEEE Computer Society*.
- Yang B, and Garcia-Molina H. 2003. Designing a Super-Peer Network. *Proceedings of the 19th International Conference on Data Engineering (ICDE'03)*. Bangalore, India: IEEE. p 49-60.
- Yatin C, Sylvia R, Lee B, Nick L, and Scott S. 2003. Making gnutella-like P2P systems scalable. *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*. Karlsruhe, Germany: ACM.
- Yeo CS, Buyya R, Assuncao MD, Yu J, Sulistio A, Venugopal S, and Placek. M. 2006. Utility Computing on Global Grids. In: Bidgoli H, editor. *Handbook of Computer Networks*: John Wiley and Sons.
- Zanikolas S, and Sakellariou R. 2005. A taxonomy of grid monitoring systems. *Future Generation Computer Systems* 21(1):163-188.
- Zhang Q, Cheng L, and Boutaba R. 2010. Cloud computing: state-of-the-art and research challenges. *Journal of Internet Services and Applications* 1(1):7-18.
- Zhuge H. 2005. Semantic grid: scientific issues, infrastructure, and methodology. *Communications of the ACM* 48(4):117 - 119

APPENDIX A

Simulation Configuration File 1

```
<?xml version='1.0' encoding='utf-8'?>

<Configuration>

    <!-- Here the variables are defined. They can be referred as
    "&variable-name" and overridden via -->

    <Default>

        <Variable name="seed" value="0"/>

        <Variable name="size" value="1000"/>

        <Variable name="finishTime" value="5m"/>

        <Variable name="actions" value="config/SemanticRD-
actions.dat"/>

        <Variable name="gnpDataFile"
value="data/measured_data.xml" />

    </Default>

    <SimulatorCore
class="de.tud.kom.p2psim.impl.simengine.Simulator"
static="getInstance" seed="$seed" finishAt="$finishTime"
statusInterval="1h">

    </SimulatorCore>

    <NetLayer
class="de.tud.kom.p2psim.impl.network.simple.SimpleNetFactory"
downBandwidth="200" upBandwidth="100">

    <LatencyModel
class="de.tud.kom.p2psim.impl.network.simple.SimpleStaticLatencyModel
" latency="10"/>
```

```

</NetLayer>

    <!--<NetLayer
class="de.tud.kom.p2psim.impl.network.gnp.GnpNetLayerFactory"
gnpFile="$gnpDataFile" downBandwidth="200" upBandwidth="100">

        <LatencyModel
class="de.tud.kom.p2psim.impl.network.gnp.GnpLatencyModel" />

    </NetLayer> -->

    <TransLayer
class="de.tud.kom.p2psim.impl.transport.DefaultTransLayerFactory"/>

    <ComponentFactory
class="de.tud.kom.p2psim.impl.overlay.SemanticRD.SemanticRDOverlayNodeFactory" />

    <Monitor class="de.tud.kom.p2psim.impl.common.DefaultMonitor"
start="0" stop="$finishTime">

        <Analyzer
class="de.tud.kom.p2psim.impl.overlay.SemanticRD.Analyzer.SemanticRDMessageAnalyzer"/>

        <Analyzer
class="de.tud.kom.p2psim.impl.overlay.SemanticRD.Analyzer.SemanticRDQueryAnalyzer"/>

        <Analyzer
class="de.tud.kom.p2psim.impl.overlay.SemanticRD.Analyzer.SemanticRDAnalyzer"/>

    </Monitor>

    <HostBuilder
class="de.tud.kom.p2psim.impl.scenario.DefaultHostBuilder"
experimentSize="$size">

```

```
<Group groupID="Group1" size="10">  
    <NetLayer/>  
    <TransLayer/>  
    <ComponentFactory />  
</Group>
```

```
<Group groupID="Group2" size="198">  
    <NetLayer/>  
    <TransLayer/>  
    <ComponentFactory />  
</Group>
```

```
<Group groupID="Group3" size="198">  
    <NetLayer/>  
    <TransLayer/>  
    <ComponentFactory />  
</Group>
```

```
<Group groupID="Group4" size="198">  
    <NetLayer/>  
    <TransLayer/>  
    <ComponentFactory />  
</Group>
```

```
<Group groupID="Group5" size="198">  
    <NetLayer/>  
    <TransLayer/>  
    <ComponentFactory />  
</Group>
```

```

        <Group groupID="Group6" size="198">
            <NetLayer/>
            <TransLayer/>
            <ComponentFactory />
        </Group>

    </HostBuilder>

    <Scenario
class="de.tud.kom.p2psim.impl.scenario.CSVScenarioFactory"
        actionsFile="$actions"

        componentClass="de.tud.kom.p2psim.impl.overlay.SemanticRD.SemanticRDOverlayNode">
    </Scenario>
</Configuration>

```

Simulation Configuration File 2

```

<?xml version='1.0' encoding='utf-8'?>
<Configuration>
    <!-- Here the variables are defined. They can be referred as
    "&variable-name" and overridden via -->

    <Default>
        <Variable name="seed" value="0"/>
        <Variable name="size" value="400"/>
        <Variable name="finishTime" value="5m"/>
    </Default>

```

```

        <Variable name="actions" value="config/SemanticRD-
actions.dat"/>

        <Variable name="gnpDataFile"
value="data/measured_data.xml" />

    </Default>

    <SimulatorCore
class="de.tud.kom.p2psim.impl.simengine.Simulator"
static="getInstance" seed="$seed" finishAt="$finishTime"
statusInterval="1h">

    </SimulatorCore>

    <NetLayer
class="de.tud.kom.p2psim.impl.network.simple.SimpleNetFactory"
downBandwidth="200" upBandwidth="100">

        <LatencyModel
class="de.tud.kom.p2psim.impl.network.simple.SimpleStaticLatencyModel
" latency="10"/>

    </NetLayer>

    <!--<NetLayer
class="de.tud.kom.p2psim.impl.network.gnp.GnpNetLayerFactory"
gnpFile="$gnpDataFile" downBandwidth="200" upBandwidth="100">

        <LatencyModel
class="de.tud.kom.p2psim.impl.network.gnp.GnpLatencyModel" />

    </NetLayer> -->

    <TransLayer
class="de.tud.kom.p2psim.impl.transport.DefaultTransLayerFactory"/>

    <ComponentFactory
class="de.tud.kom.p2psim.impl.overlay.SemanticRD.SemanticRDOverlayNod
eFactory" />

```

```

    <Monitor class="de.tud.kom.p2psim.impl.common.DefaultMonitor"
start="0" stop="$finishTime">

        <Analyzer
class="de.tud.kom.p2psim.impl.overlay.SemanticRD.Analyzer.SemanticRDM
essageAnalyzer"/>

        <Analyzer
class="de.tud.kom.p2psim.impl.overlay.SemanticRD.Analyzer.SemanticRDQ
ueryAnalyzer"/>

        <Analyzer
class="de.tud.kom.p2psim.impl.overlay.SemanticRD.Analyzer.SemanticRDA
nalyzer"/>

    </Monitor>

```

```

    <HostBuilder
class="de.tud.kom.p2psim.impl.scenario.DefaultHostBuilder"
experimentSize="$size">

```

```

    <Group groupID="Group1" size="10">

        <NetLayer/>

        <TransLayer/>

        <ComponentFactory />

    </Group>

```

```

    <Group groupID="Group2" size="78">

        <NetLayer/>

        <TransLayer/>

        <ComponentFactory />

    </Group>

```

```

    <Group groupID="Group3" size="78">

        <NetLayer/>

        <TransLayer/>

```

```

        <ComponentFactory />
    </Group>

    <Group groupId="Group4" size="78">
        <NetLayer/>
        <TransLayer/>
        <ComponentFactory />
    </Group>

    <Group groupId="Group5" size="78">
        <NetLayer/>
        <TransLayer/>
        <ComponentFactory />
    </Group>

    <Group groupId="Group6" size="78">
        <NetLayer/>
        <TransLayer/>
        <ComponentFactory />
    </Group>

</HostBuilder>

<Scenario
class="de.tud.kom.p2psim.impl.scenario.CSVScenarioFactory"

    actionsFile="$actions"

    componentClass="de.tud.kom.p2psim.impl.overlay.SemanticRD.SemanticRDOverlayNode">

```

APPENDIX B

Sample of the Simulation Output File

SemanticRD-Analyzer

number of nodes 1000

Lookups: 990.0 Succeeded; 820.0 Rate: 82.82828282828282

Average Delay: 33176 ms

Average Hops: 4.65

QueryMessages/Query: 23.324242424242424

Load Balance Ratio Head: 1.093385711492276

Load Balance Ratio: 1.2056631892697467

180000000 Hops: 5.0 Delay: 40101 RDP: 2.227155696466509E-4

180000000 Hops: 6.0 Delay: 47115 RDP: 7.415014164305949

180000000 Hops: 5.0 Delay: 32780 RDP: 1.8205608971510833E-4

180000000 Hops: 5.0 Delay: 33523 RDP: 1.861899043938196E-4

180000000 Hops: 3.0 Delay: 16763 RDP: 9.311763054264948E-5

180000000 Hops: 5.0 Delay: 29443 RDP: 1.635228728361211E-4

180000000 Hops: 5.0 Delay: 35639 RDP: 1.9795219814615675E-4

1800000000 Hops: 3.0 Delay: 17645 RDP: 14.16131621187801
1800000000 Hops: 4.0 Delay: 31313 RDP: 1.7392395708890008E-4
1800000000 Hops: 5.0 Delay: 35843 RDP: 1.9907696891137896E-4
1800000000 Hops: 3.0 Delay: 28801 RDP: 5.564335394126739
1800000000 Hops: 5.0 Delay: 38292 RDP: 2.1268548500713686E-4
1800000000 Hops: 3.0 Delay: 25244 RDP: 5.079275653923541
1800000000 Hops: 6.0 Delay: 34674 RDP: 1.925907461696123E-4
1800000000 Hops: 4.0 Delay: 30192 RDP: 1.6768574691312184E-4
1800000000 Hops: 3.0 Delay: 19947 RDP: 1.1080345027734747E-4
1800000000 Hops: 6.0 Delay: 45656 RDP: 2.535780746097166E-4
1800000000 Hops: 4.0 Delay: 24614 RDP: 3.156045646877805
1800000000 Hops: 6.0 Delay: 39427 RDP: 2.1899074499021805E-4
1800000000 Hops: 3.0 Delay: 14659 RDP: 8.143110045652522E-5
1800000000 Hops: 5.0 Delay: 33694 RDP: 6.578289730573995
1800000000 Hops: 4.0 Delay: 19927 RDP: 1.1067610587351712E-4
1800000000 Hops: 5.0 Delay: 31691 RDP: 1.7601031746699542E-4
1800000000 Hops: 6.0 Delay: 42963 RDP: 2.3862558064002777E-4
1800000000 Hops: 4.0 Delay: 26819 RDP: 1.4897223847705235E-4
1800000000 Hops: 4.0 Delay: 28109 RDP: 1.561284698523473E-4
1800000000 Hops: 3.0 Delay: 19122 RDP: 4.565902578796561
1800000000 Hops: 4.0 Delay: 31906 RDP: 8.02061337355455
1800000000 Hops: 5.0 Delay: 32705 RDP: 1.8164107526917912E-4
1800000000 Hops: 6.0 Delay: 41128 RDP: 2.284307481872364E-4
1800000000 Hops: 4.0 Delay: 27476 RDP: 1.526204821808505E-4
1800000000 Hops: 6.0 Delay: 36683 RDP: 2.0374978136042702E-4
1800000000 Hops: 5.0 Delay: 33822 RDP: 1.8785092498946275E-4
1800000000 Hops: 5.0 Delay: 32484 RDP: 1.8041247176244074E-4
1800000000 Hops: 3.0 Delay: 17004 RDP: 22.2565445026178
1800000000 Hops: 5.0 Delay: 35683 RDP: 1.981978773436692E-4
1800000000 Hops: 6.0 Delay: 40491 RDP: 2.248935367293785E-4
1800000000 Hops: 3.0 Delay: 18896 RDP: 4.034158838599487
1800000000 Hops: 6.0 Delay: 41536 RDP: 2.306972096679282E-4
1800000000 Hops: 6.0 Delay: 41486 RDP: 2.3042364742260408E-4
1800000000 Hops: 3.0 Delay: 17363 RDP: 4.94531472514953
1800000000 Hops: 3.0 Delay: 16274 RDP: 9.04013266741874E-5
1800000000 Hops: 5.0 Delay: 39994 RDP: 2.221369384635469E-4
1800000000 Hops: 6.0 Delay: 40857 RDP: 51.32788944723618
1800000000 Hops: 5.0 Delay: 34153 RDP: 1.896978319547928E-4
1800000000 Hops: 6.0 Delay: 49410 RDP: 3.936110889827133

1800000000 Hops: 4.0 Delay: 27867 RDP: 1.5477707812957175E-4
1800000000 Hops: 5.0 Delay: 33746 RDP: 1.8742310749855992E-4
1800000000 Hops: 5.0 Delay: 41479 RDP: 2.303735383153338E-4
1800000000 Hops: 5.0 Delay: 34394 RDP: 1.910350612767704E-4
1800000000 Hops: 5.0 Delay: 30138 RDP: 1.6738662409587939E-4
1800000000 Hops: 4.0 Delay: 20237 RDP: 1.123974042525686E-4
1800000000 Hops: 3.0 Delay: 12413 RDP: 6.895455200089246E-5
1800000000 Hops: 5.0 Delay: 34508 RDP: 1.916665890920938E-4
1800000000 Hops: 5.0 Delay: 35387 RDP: 1.9654240110853312E-4
1800000000 Hops: 4.0 Delay: 25433 RDP: 1.412590245288162E-4
1800000000 Hops: 4.0 Delay: 27873 RDP: 1.5482170891305795E-4
1800000000 Hops: 5.0 Delay: 33330 RDP: 1.8511582871799434E-4
1800000000 Hops: 5.0 Delay: 34083 RDP: 1.893043061472584E-4
1800000000 Hops: 5.0 Delay: 26940 RDP: 1.4962427395362576E-4
1800000000 Hops: 5.0 Delay: 28829 RDP: 1.6011970593426435E-4
1800000000 Hops: 5.0 Delay: 27731 RDP: 1.5401787059394185E-4
1800000000 Hops: 4.0 Delay: 31306 RDP: 1.7388520882346628E-4
1800000000 Hops: 5.0 Delay: 30274 RDP: 24.296950240770464
1800000000 Hops: 4.0 Delay: 28746 RDP: 1.596565716385524E-4
1800000000 Hops: 5.0 Delay: 39159 RDP: 2.175002854764147E-4
1800000000 Hops: 5.0 Delay: 28565 RDP: 1.586515212753634E-4
1800000000 Hops: 5.0 Delay: 33979 RDP: 1.8871586222996725E-4
1800000000 Hops: 5.0 Delay: 42413 RDP: 2.3556408910022138E-4
1800000000 Hops: 3.0 Delay: 18496 RDP: 1.0274371034734929E-4
1800000000 Hops: 6.0 Delay: 46372 RDP: 2.575517431876989E-4
1800000000 Hops: 3.0 Delay: 20521 RDP: 12.302757793764988
1800000000 Hops: 6.0 Delay: 45459 RDP: 2.5247823306225205E-4
1800000000 Hops: 5.0 Delay: 32538 RDP: 1.8072021554726384E-4
1800000000 Hops: 4.0 Delay: 27297 RDP: 1.5160745305282797E-4
1800000000 Hops: 6.0 Delay: 42021 RDP: 2.3338904785694052E-4
1800000000 Hops: 5.0 Delay: 38849 RDP: 2.1577385469274642E-4
1800000000 Hops: 4.0 Delay: 27866 RDP: 1.5477397051731932E-4
1800000000 Hops: 6.0 Delay: 43247 RDP: 52.676004872107185
1800000000 Hops: 5.0 Delay: 42602 RDP: 2.3661447157590764E-4
1800000000 Hops: 6.0 Delay: 42849 RDP: 2.3799319631277784E-4
1800000000 Hops: 5.0 Delay: 38831 RDP: 2.156752656433773E-4
1800000000 Hops: 3.0 Delay: 12442 RDP: 6.91151425277893E-5
1800000000 Hops: 5.0 Delay: 42571 RDP: 2.3644490218381424E-4
1800000000 Hops: 5.0 Delay: 40600 RDP: 2.2550331896451365E-4

180000000 Hops: 6.0 Delay: 34300 RDP: 1.905091157834314E-4
 180000000 Hops: 4.0 Delay: 30375 RDP: 3.9310210948621718
 180000000 Hops: 5.0 Delay: 48816 RDP: 52.88840736728061
 180000000 Hops: 4.0 Delay: 39316 RDP: 2.1836857755482835E-4
 180000000 Hops: 5.0 Delay: 46428 RDP: 7.326495186997001
 180000000 Hops: 5.0 Delay: 41031 RDP: 2.2789237994386876E-4
 180000000 Hops: 5.0 Delay: 40417 RDP: 2.2447713024197866E-4
 180000000 Hops: 5.0 Delay: 29984 RDP: 1.6654630237714297E-4
 180000000 Hops: 5.0 Delay: 36225 RDP: 14.95664739884393
 180000000 Hops: 4.0 Delay: 28500 RDP: 1.5830734190624802E-4
 180000000 Hops: 4.0 Delay: 27591 RDP: 1.5324148138208475E-4
 180000000 Hops: 5.0 Delay: 32521 RDP: 1.8062575724964617E-4
 180000000 Hops: 6.0 Delay: 41500 RDP: 4.179675697451908
 180000000 Hops: 6.0 Delay: 55076 RDP: 3.0588173431188976E-4
 180000000 Hops: 6.0 Delay: 44162 RDP: 23.169989506820567
 180000000 Hops: 3.0 Delay: 12295 RDP: 6.829968026528207E-5
 180000000 Hops: 6.0 Delay: 40975 RDP: 2.2758138792254817E-4
 180000000 Hops: 5.0 Delay: 35814 RDP: 1.9891735173988115E-4
 180000000 Hops: 5.0 Delay: 36041 RDP: 2.00185967826253E-4
 180000000 Hops: 5.0 Delay: 35068 RDP: 1.947637433262677E-4
 180000000 Hops: 6.0 Delay: 37052 RDP: 2.0579537482393295E-4
 180000000 Hops: 5.0 Delay: 40555 RDP: 2.2524187842518098E-4
 180000000 Hops: 4.0 Delay: 39537 RDP: 2.195958940116699E-4
 180000000 Hops: 5.0 Delay: 37166 RDP: 2.0642578027053752E-4
 180000000 Hops: 3.0 Delay: 25590 RDP: 4.9088816420487245
 180000000 Hops: 5.0 Delay: 41097 RDP: 67.81683168316832
 180000000 Hops: 3.0 Delay: 19105 RDP: 1.0612751084053222E-4
 180000000 Hops: 4.0 Delay: 32778 RDP: 1.820491698490096E-4
 180000000 Hops: 3.0 Delay: 8729 RDP: 4.849019912751083E-5
 180000000 Hops: 6.0 Delay: 30162 RDP: 1.6753455866849786E-4
 180000000 Hops: 3.0 Delay: 11128 RDP: 3.2509494595384165
 180000000 Hops: 3.0 Delay: 8375 RDP: 4.652387700915654E-5
 180000000 Hops: 5.0 Delay: 41129 RDP: 12.489826905557242
 180000000 Hops: 6.0 Delay: 38108 RDP: 2.1166345273291215E-4
 180000000 Hops: 5.0 Delay: 36794 RDP: 2.043542132216443E-4
 180000000 Hops: 5.0 Delay: 38532 RDP: 2.1401369708767248E-4
 180000000 Hops: 5.0 Delay: 37373 RDP: 30.91232423490488
 180000000 Hops: 5.0 Delay: 33426 RDP: 1.8565273384536706E-4
 180000000 Hops: 4.0 Delay: 31474 RDP: 1.7481444988004904E-4

1800000000 Hops: 6.0 Delay: 42217 RDP: 2.344785119747138E-4
 1800000000 Hops: 3.0 Delay: 25582 RDP: 9.126650017838031
 1800000000 Hops: 3.0 Delay: 13846 RDP: 7.691503579404453E-5
 1800000000 Hops: 6.0 Delay: 39855 RDP: 2.2136526934054694E-4
 1800000000 Hops: 3.0 Delay: 20575 RDP: 1.1429229066530951E-4
 1800000000 Hops: 3.0 Delay: 14392 RDP: 7.99472739062819E-5
 1800000000 Hops: 6.0 Delay: 44734 RDP: 182.58775510204083
 1800000000 Hops: 5.0 Delay: 37208 RDP: 2.066602497274282E-4
 1800000000 Hops: 5.0 Delay: 36359 RDP: 2.019528320633978E-4
 1800000000 Hops: 6.0 Delay: 37732 RDP: 2.0957118465305305E-4
 1800000000 Hops: 5.0 Delay: 38197 RDP: 2.1215125544172525E-4
 1800000000 Hops: 6.0 Delay: 34871 RDP: 1.9368565652764014E-4
 1800000000 Hops: 5.0 Delay: 40508 RDP: 11.517770827409723
 1800000000 Hops: 5.0 Delay: 31585 RDP: 1.7542903744083887E-4
 1800000000 Hops: 6.0 Delay: 38155 RDP: 2.119222250599478E-4
 1800000000 Hops: 5.0 Delay: 29733 RDP: 31.530222693531282
 1800000000 Hops: 4.0 Delay: 35261 RDP: 1.958458855451562E-4
 1800000000 Hops: 6.0 Delay: 44242 RDP: 126.04558404558405
 1800000000 Hops: 6.0 Delay: 39258 RDP: 2.180495033691781E-4
 1800000000 Hops: 3.0 Delay: 25095 RDP: 22.44633273703041
 1800000000 Hops: 3.0 Delay: 20733 RDP: 6.2411198073449725
 1800000000 Hops: 5.0 Delay: 43537 RDP: 2.4179515807542428E-4
 1800000000 Hops: 4.0 Delay: 31900 RDP: 1.771750828228252E-4
 1800000000 Hops: 5.0 Delay: 42074 RDP: 2.336832518884569E-4
 1800000000 Hops: 5.0 Delay: 42200 RDP: 5.87661885531263
 1800000000 Hops: 6.0 Delay: 45327 RDP: 2.5175281515960843E-4
 1800000000 Hops: 5.0 Delay: 34670 RDP: 1.925614420267119E-4
 1800000000 Hops: 5.0 Delay: 38311 RDP: 2.127820666201206E-4
 1800000000 Hops: 3.0 Delay: 9737 RDP: 5.408967073050431E-5
 1800000000 Hops: 5.0 Delay: 34960 RDP: 1.9416268007901288E-4
 1800000000 Hops: 5.0 Delay: 37417 RDP: 2.0782068731000684E-4
 1800000000 Hops: 6.0 Delay: 53069 RDP: 7.085313751668892
 1800000000 Hops: 4.0 Delay: 30164 RDP: 1.6754800263606487E-4
 1800000000 Hops: 3.0 Delay: 15836 RDP: 8.796788139112128E-5
 1800000000 Hops: 5.0 Delay: 31267 RDP: 1.7365508945705834E-4
 1800000000 Hops: 5.0 Delay: 34508 RDP: 1.9166249804806454E-4
 1800000000 Hops: 6.0 Delay: 40225 RDP: 2.2341154736948247E-4
 1800000000 Hops: 3.0 Delay: 18119 RDP: 5.757546870034954
 1800000000 Hops: 6.0 Delay: 49653 RDP: 52.654294803817606

1800000000 Hops: 4.0 Delay: 30088 RDP: 9.62816
1800000000 Hops: 5.0 Delay: 31015 RDP: 1.7225737803787467E-4
1800000000 Hops: 5.0 Delay: 43454 RDP: 2.4133940247076475E-4
1800000000 Hops: 4.0 Delay: 33796 RDP: 4.354032465859315
1800000000 Hops: 5.0 Delay: 31490 RDP: 1.7490613222788067E-4
1800000000 Hops: 3.0 Delay: 8908 RDP: 4.948565939977239E-5
1800000000 Hops: 6.0 Delay: 34880 RDP: 1.93734249997009E-4
1800000000 Hops: 5.0 Delay: 42171 RDP: 2.3422218052446706E-4
1800000000 Hops: 3.0 Delay: 19955 RDP: 1.1084845775965785E-4
1800000000 Hops: 5.0 Delay: 38786 RDP: 2.1542378659286353E-4
1800000000 Hops: 6.0 Delay: 38630 RDP: 2.1455459385517976E-4
1800000000 Hops: 5.0 Delay: 33927 RDP: 1.8844278777373506E-4
1800000000 Hops: 6.0 Delay: 47750 RDP: 19.77225672877847
1800000000 Hops: 5.0 Delay: 37407 RDP: 25.726960110041265
1800000000 Hops: 6.0 Delay: 45151 RDP: 15.248564674096588
1800000000 Hops: 5.0 Delay: 34918 RDP: 1.9394011510483018E-4
1800000000 Hops: 6.0 Delay: 42157 RDP: 2.3414928167819223E-4
1800000000 Hops: 5.0 Delay: 36791 RDP: 2.0433227975386734E-4
1800000000 Hops: 5.0 Delay: 33608 RDP: 1.8666447195576813E-4
1800000000 Hops: 4.0 Delay: 27762 RDP: 1.5419994818788427E-4
1800000000 Hops: 4.0 Delay: 25961 RDP: 1.4420159797766494E-4
1800000000 Hops: 5.0 Delay: 45802 RDP: 9.53413821815154
1800000000 Hops: 4.0 Delay: 29211 RDP: 6.161358363214512
1800000000 Hops: 3.0 Delay: 21004 RDP: 1.1667462347159243E-4
1800000000 Hops: 6.0 Delay: 45574 RDP: 2.5311848960586034E-4
1800000000 Hops: 5.0 Delay: 38986 RDP: 2.1653666986796874E-4
1800000000 Hops: 4.0 Delay: 30705 RDP: 1.7054212467127526E-4
1800000000 Hops: 5.0 Delay: 37997 RDP: 2.1103926588349848E-4
1800000000 Hops: 4.0 Delay: 34595 RDP: 1.9214707698751574E-4
1800000000 Hops: 6.0 Delay: 47001 RDP: 33.23974540311174
1800000000 Hops: 5.0 Delay: 38917 RDP: 2.1614356918230688E-4
1800000000 Hops: 6.0 Delay: 36833 RDP: 2.045782777903744E-4
1800000000 Hops: 3.0 Delay: 20066 RDP: 1.1146405779075326E-4
1800000000 Hops: 6.0 Delay: 41391 RDP: 2.298882354169278E-4
1800000000 Hops: 4.0 Delay: 28236 RDP: 1.5682306375626773E-4
1800000000 Hops: 5.0 Delay: 33562 RDP: 1.8640285014967574E-4
1800000000 Hops: 5.0 Delay: 39804 RDP: 2.2106951056563305E-4
1800000000 Hops: 3.0 Delay: 16479 RDP: 9.15411571242218E-5
1800000000 Hops: 3.0 Delay: 13154 RDP: 7.30722522976332E-5

1800000000 Hops: 6.0 Delay: 37927 RDP: 2.1065840552310158E-4
 1800000000 Hops: 3.0 Delay: 20744 RDP: 13.638395792241946
 1800000000 Hops: 4.0 Delay: 25386 RDP: 1.4100831689115777E-4
 1800000000 Hops: 3.0 Delay: 22617 RDP: 7.386348791639452
 1800000000 Hops: 5.0 Delay: 43625 RDP: 2.4229320440235727E-4
 1800000000 Hops: 3.0 Delay: 16730 RDP: 13.690671031096564
 1800000000 Hops: 5.0 Delay: 36086 RDP: 2.0042792133234636E-4
 1800000000 Hops: 4.0 Delay: 26384 RDP: 1.465495759402167E-4
 1800000000 Hops: 3.0 Delay: 13580 RDP: 2.730197024527543
 1800000000 Hops: 6.0 Delay: 32011 RDP: 1.7780651921206632E-4
 1800000000 Hops: 3.0 Delay: 9803 RDP: 2.12969802302846
 1800000000 Hops: 5.0 Delay: 30055 RDP: 1.6692556003664242E-4
 1800000000 Hops: 6.0 Delay: 40715 RDP: 2.2614058027089703E-4
 1800000000 Hops: 5.0 Delay: 31523 RDP: 1.7507501016971263E-4
 1800000000 Hops: 5.0 Delay: 32355 RDP: 1.7971163156666053E-4
 1800000000 Hops: 5.0 Delay: 39013 RDP: 2.1668448061960797E-4
 1800000000 Hops: 4.0 Delay: 39008 RDP: 2.1665941015753057E-4
 1800000000 Hops: 4.0 Delay: 33375 RDP: 1.853795135224982E-4
 1800000000 Hops: 3.0 Delay: 15763 RDP: 8.756345906582438E-5
 1800000000 Hops: 4.0 Delay: 28405 RDP: 1.5777385002387187E-4
 1800000000 Hops: 4.0 Delay: 27863 RDP: 1.5476819317941133E-4
 1800000000 Hops: 4.0 Delay: 30663 RDP: 1.7030752814094045E-4
 1800000000 Hops: 3.0 Delay: 16012 RDP: 2.0009997500624843
 1800000000 Hops: 6.0 Delay: 36710 RDP: 5.641616720454895
 1800000000 Hops: 4.0 Delay: 36407 RDP: 2.0221179502450122E-4
 1800000000 Hops: 6.0 Delay: 48627 RDP: 13.744205765969474
 1800000000 Hops: 5.0 Delay: 39114 RDP: 18.511121628017037
 1800000000 Hops: 6.0 Delay: 42166 RDP: 25.17373134328358
 1800000000 Hops: 3.0 Delay: 20521 RDP: 23.641705069124423
 1800000000 Hops: 5.0 Delay: 35440 RDP: 1.968405273584339E-4
 1800000000 Hops: 4.0 Delay: 37910 RDP: 2.1055209102053039E-4
 1800000000 Hops: 5.0 Delay: 46657 RDP: 2.5913449367920836E-4
 1800000000 Hops: 5.0 Delay: 33279 RDP: 1.848377810890605E-4
 1800000000 Hops: 3.0 Delay: 21770 RDP: 10.542372881355933
 1800000000 Hops: 3.0 Delay: 10264 RDP: 5.7017561670084154E-5
 1800000000 Hops: 4.0 Delay: 28669 RDP: 1.5924085796590297E-4
 1800000000 Hops: 4.0 Delay: 34114 RDP: 1.8946853841706772E-4
 1800000000 Hops: 6.0 Delay: 44318 RDP: 2.461384346359776E-4
 1800000000 Hops: 4.0 Delay: 26419 RDP: 1.4673031930370284E-4

1800000000 Hops: 5.0 Delay: 35069 RDP: 1.9477115347521468E-4
1800000000 Hops: 5.0 Delay: 30955 RDP: 1.7194260988385334E-4
1800000000 Hops: 5.0 Delay: 27199 RDP: 1.5107473547021556E-4
1800000000 Hops: 6.0 Delay: 39067 RDP: 2.1697976913445872E-4
1800000000 Hops: 6.0 Delay: 43732 RDP: 8.094021839718675
1800000000 Hops: 3.0 Delay: 19480 RDP: 1.0820912230564354E-4
1800000000 Hops: 5.0 Delay: 38822 RDP: 2.1561631035916793E-4
1800000000 Hops: 5.0 Delay: 28678 RDP: 1.5927601182685758E-4
1800000000 Hops: 5.0 Delay: 29818 RDP: 1.6560466064321043E-4
1800000000 Hops: 5.0 Delay: 29543 RDP: 1.640909174880127E-4
1800000000 Hops: 6.0 Delay: 38969 RDP: 2.1644500840452486E-4
1800000000 Hops: 4.0 Delay: 34350 RDP: 2.683803422142355
1800000000 Hops: 3.0 Delay: 21844 RDP: 8.479813664596273
1800000000 Hops: 4.0 Delay: 24799 RDP: 1.3773379296360989E-4
1800000000 Hops: 3.0 Delay: 15531 RDP: 8.627493446846282E-5
1800000000 Hops: 3.0 Delay: 20731 RDP: 9.288082437275985
1800000000 Hops: 5.0 Delay: 31944 RDP: 1.7741300021972798E-4
1800000000 Hops: 3.0 Delay: 29720 RDP: 3.9069278296306034
1800000000 Hops: 4.0 Delay: 31155 RDP: 1.7304772395724792E-4
1800000000 Hops: 4.0 Delay: 22844 RDP: 2.8630154154655973
1800000000 Hops: 5.0 Delay: 31100 RDP: 1.7273530888280774E-4
1800000000 Hops: 4.0 Delay: 34523 RDP: 1.9175213007578478E-4
1800000000 Hops: 4.0 Delay: 38615 RDP: 2.1447720047928476E-4
1800000000 Hops: 5.0 Delay: 39517 RDP: 2.1947563113475962E-4
1800000000 Hops: 3.0 Delay: 22076 RDP: 5.899518973810796
1800000000 Hops: 5.0 Delay: 35373 RDP: 11.894082044384668
1800000000 Hops: 6.0 Delay: 48507 RDP: 21.44429708222812
1800000000 Hops: 5.0 Delay: 40412 RDP: 2.2445179224328443E-4
1800000000 Hops: 5.0 Delay: 32880 RDP: 1.8262237262475526E-4
1800000000 Hops: 5.0 Delay: 32596 RDP: 1.8104366015938862E-4
1800000000 Hops: 5.0 Delay: 34860 RDP: 1.9362066132186625E-4
1800000000 Hops: 5.0 Delay: 31418 RDP: 1.7450197841854085E-4
1800000000 Hops: 6.0 Delay: 44217 RDP: 14.699800531914894
1800000000 Hops: 4.0 Delay: 28541 RDP: 1.5853307013396153E-4
1800000000 Hops: 4.0 Delay: 30229 RDP: 1.6789696128661203E-4
1800000000 Hops: 5.0 Delay: 31989 RDP: 1.7767220321074538E-4
1800000000 Hops: 3.0 Delay: 18860 RDP: 31.019736842105264
1800000000 Hops: 4.0 Delay: 26776 RDP: 1.4871433855043553E-4
1800000000 Hops: 5.0 Delay: 34021 RDP: 1.8895999100305836E-4

1800000000 Hops: 5.0 Delay: 38487 RDP: 2.1376222261613423E-4
 1800000000 Hops: 6.0 Delay: 40696 RDP: 2.2603042737457304E-4
 1800000000 Hops: 5.0 Delay: 42248 RDP: 2.34650143789307E-4
 1800000000 Hops: 6.0 Delay: 39242 RDP: 2.179529479452525E-4
 1800000000 Hops: 6.0 Delay: 52925 RDP: 2.9394097210969756E-4
 1800000000 Hops: 3.0 Delay: 24238 RDP: 2.7659477347940205
 1800000000 Hops: 6.0 Delay: 45747 RDP: 2.540853550393368E-4
 1800000000 Hops: 4.0 Delay: 26058 RDP: 1.4472935061576623E-4
 1800000000 Hops: 5.0 Delay: 35133 RDP: 1.9513608004746052E-4
 1800000000 Hops: 5.0 Delay: 38900 RDP: 2.1604827227085296E-4
 1800000000 Hops: 4.0 Delay: 27536 RDP: 1.5293671172173529E-4
 1800000000 Hops: 4.0 Delay: 32579 RDP: 6.437265362576566
 1800000000 Hops: 3.0 Delay: 26935 RDP: 10.272692601067886
 1800000000 Hops: 3.0 Delay: 17475 RDP: 42.518248175182485
 1800000000 Hops: 5.0 Delay: 37978 RDP: 2.1093522345245553E-4
 1800000000 Hops: 5.0 Delay: 40517 RDP: 2.250298358782323E-4
 1800000000 Hops: 5.0 Delay: 35498 RDP: 1.971652789466573E-4
 1800000000 Hops: 6.0 Delay: 38817 RDP: 2.15596426683485E-4
 1800000000 Hops: 6.0 Delay: 40038 RDP: 4.730946472881957
 1800000000 Hops: 6.0 Delay: 50658 RDP: 6.492118415993849
 1800000000 Hops: 4.0 Delay: 20673 RDP: 2.2812844846612226
 1800000000 Hops: 6.0 Delay: 44139 RDP: 2.4515259739729575E-4
 1800000000 Hops: 5.0 Delay: 47432 RDP: 4.729956122856003
 1800000000 Hops: 6.0 Delay: 42158 RDP: 2.341515247514291E-4
 1800000000 Hops: 4.0 Delay: 28451 RDP: 1.5802428794037016E-4
 1800000000 Hops: 4.0 Delay: 36689 RDP: 2.0377743343075186E-4
 1800000000 Hops: 6.0 Delay: 48693 RDP: 32.354152823920266
 1800000000 Hops: 3.0 Delay: 24761 RDP: 4.88478989938844
 1800000000 Hops: 6.0 Delay: 56028 RDP: 20.478070175438596
 1800000000 Hops: 4.0 Delay: 29492 RDP: 1.638086158599644E-4
 1800000000 Hops: 6.0 Delay: 37333 RDP: 2.0735756840628038E-4
 1800000000 Hops: 5.0 Delay: 34052 RDP: 1.8912975142956477E-4
 1800000000 Hops: 4.0 Delay: 19804 RDP: 1.0999237151473059E-4
 1800000000 Hops: 5.0 Delay: 46354 RDP: 6.210342979635584
 1800000000 Hops: 5.0 Delay: 35350 RDP: 1.96333539191965E-4
 1800000000 Hops: 3.0 Delay: 13271 RDP: 7.372092132253411E-5
 1800000000 Hops: 6.0 Delay: 40390 RDP: 2.2433264370994095E-4
 1800000000 Hops: 6.0 Delay: 46102 RDP: 14.185230769230769
 1800000000 Hops: 6.0 Delay: 39739 RDP: 2.207167230022233E-4

1800000000 Hops: 4.0 Delay: 26614 RDP: 1.478205713536685E-4
 1800000000 Hops: 3.0 Delay: 13682 RDP: 4.3065785332074284
 1800000000 Hops: 3.0 Delay: 22549 RDP: 9.902942468159859
 1800000000 Hops: 5.0 Delay: 41745 RDP: 20.635195254572416
 1800000000 Hops: 4.0 Delay: 36815 RDP: 2.0447801464494703E-4
 1800000000 Hops: 3.0 Delay: 22203 RDP: 11.879614767255216
 1800000000 Hops: 4.0 Delay: 32151 RDP: 1.7857378911574677E-4
 1800000000 Hops: 5.0 Delay: 39143 RDP: 2.1741224408571545E-4
 1800000000 Hops: 6.0 Delay: 33206 RDP: 1.844271248679328E-4
 1800000000 Hops: 6.0 Delay: 43336 RDP: 53.30381303813038
 1800000000 Hops: 3.0 Delay: 26860 RDP: 9.230240549828178
 1800000000 Hops: 6.0 Delay: 33091 RDP: 1.8380276245592817E-4
 1800000000 Hops: 6.0 Delay: 47518 RDP: 2.639171239580597E-4
 1800000000 Hops: 5.0 Delay: 33311 RDP: 1.8501390378570517E-4
 1800000000 Hops: 6.0 Delay: 48584 RDP: 2.698212186731034E-4
 1800000000 Hops: 6.0 Delay: 36783 RDP: 2.0429699400992084E-4
 1800000000 Hops: 6.0 Delay: 40894 RDP: 2.2713221687708674E-4
 1800000000 Hops: 5.0 Delay: 40998 RDP: 2.277104019564588E-4
 1800000000 Hops: 5.0 Delay: 49119 RDP: 3.962807583703106
 1800000000 Hops: 3.0 Delay: 21105 RDP: 24.9468085106383
 1800000000 Hops: 5.0 Delay: 33925 RDP: 1.884322892327496E-4
 1800000000 Hops: 5.0 Delay: 33476 RDP: 1.859221230231515E-4
 1800000000 Hops: 3.0 Delay: 23036 RDP: 4.951848667239897
 1800000000 Hops: 5.0 Delay: 37238 RDP: 2.068254716160061E-4
 1800000000 Hops: 5.0 Delay: 37352 RDP: 2.0745785322255798E-4
 1800000000 Hops: 6.0 Delay: 39663 RDP: 2.2029381895689325E-4
 1800000000 Hops: 6.0 Delay: 36686 RDP: 2.0375822679533654E-4
 1800000000 Hops: 3.0 Delay: 23390 RDP: 18.373919874312648
 1800000000 Hops: 4.0 Delay: 28953 RDP: 1.608164027731873E-4
 1800000000 Hops: 5.0 Delay: 41658 RDP: 2.3136996495609795E-4
 1800000000 Hops: 5.0 Delay: 33853 RDP: 1.8802769726351023E-4
 1800000000 Hops: 3.0 Delay: 15201 RDP: 8.444174112859683E-5
 1800000000 Hops: 5.0 Delay: 33679 RDP: 1.8705127223717858E-4
 1800000000 Hops: 6.0 Delay: 45319 RDP: 111.34889434889435
 1800000000 Hops: 5.0 Delay: 36616 RDP: 2.0337127319892427E-4
 1800000000 Hops: 5.0 Delay: 45683 RDP: 40.4991134751773
 1800000000 Hops: 4.0 Delay: 25102 RDP: 1.394169231261573E-4
 1800000000 Hops: 3.0 Delay: 17198 RDP: 9.553392615917432E-5
 1800000000 Hops: 5.0 Delay: 38180 RDP: 2.1205618266935023E-4

1800000000 Hops: 3.0 Delay: 6228 RDP: 3.459733331443108E-5
 1800000000 Hops: 6.0 Delay: 45096 RDP: 2.5046698184687564E-4
 1800000000 Hops: 5.0 Delay: 32194 RDP: 1.7881023411493876E-4
 1800000000 Hops: 4.0 Delay: 32679 RDP: 15.473011363636363
 1800000000 Hops: 5.0 Delay: 37653 RDP: 2.0912598866361967E-4
 1800000000 Hops: 5.0 Delay: 40480 RDP: 2.248295413842815E-4
 1800000000 Hops: 6.0 Delay: 47285 RDP: 22.48454588682834
 1800000000 Hops: 4.0 Delay: 18017 RDP: 1.0008166290407311E-4
 1800000000 Hops: 5.0 Delay: 39994 RDP: 2.2213455724434867E-4
 1800000000 Hops: 5.0 Delay: 41130 RDP: 2.2844498663980069E-4
 1800000000 Hops: 5.0 Delay: 36471 RDP: 2.0256101865348658E-4
 1800000000 Hops: 3.0 Delay: 22162 RDP: 3.0797665369649807
 1800000000 Hops: 6.0 Delay: 46681 RDP: 2.592644353448477E-4
 1800000000 Hops: 3.0 Delay: 14599 RDP: 8.109723813270462E-5
 1800000000 Hops: 6.0 Delay: 36705 RDP: 2.0386065821594027E-4
 1800000000 Hops: 6.0 Delay: 43260 RDP: 2.4026575058212375E-4
 1800000000 Hops: 4.0 Delay: 45558 RDP: 4.0331090651558075
 1800000000 Hops: 6.0 Delay: 45260 RDP: 11.945104249142254
 1800000000 Hops: 4.0 Delay: 14114 RDP: 7.839040210888288E-5
 1800000000 Hops: 4.0 Delay: 30141 RDP: 1.6741646462195328E-4
 1800000000 Hops: 6.0 Delay: 41996 RDP: 2.3325167858340805E-4
 1800000000 Hops: 6.0 Delay: 51048 RDP: 6.429219143576826
 1800000000 Hops: 4.0 Delay: 34371 RDP: 1.9089439034304428E-4
 1800000000 Hops: 5.0 Delay: 33641 RDP: 1.8684291316899243E-4
 1800000000 Hops: 5.0 Delay: 45752 RDP: 2.541088648653422E-4
 1800000000 Hops: 5.0 Delay: 38795 RDP: 25.7090788601723
 1800000000 Hops: 3.0 Delay: 20341 RDP: 18.441523118766998
 1800000000 Hops: 5.0 Delay: 44528 RDP: 2.4731190350436936E-4
 1800000000 Hops: 3.0 Delay: 28902 RDP: 3.9146688338073954
 1800000000 Hops: 3.0 Delay: 17700 RDP: 3.392104254503641
 1800000000 Hops: 6.0 Delay: 51130 RDP: 12.197041984732824
 1800000000 Hops: 6.0 Delay: 41946 RDP: 2.3297848631468007E-4
 1800000000 Hops: 3.0 Delay: 19608 RDP: 1.0892065012874056E-4
 1800000000 Hops: 4.0 Delay: 37675 RDP: 2.0925486821383893E-4
 1800000000 Hops: 6.0 Delay: 36433 RDP: 2.023574057350687E-4
 1800000000 Hops: 4.0 Delay: 13340 RDP: 7.40922204762538E-5
 1800000000 Hops: 5.0 Delay: 35060 RDP: 1.9473550070057567E-4
 1800000000 Hops: 3.0 Delay: 21429 RDP: 1.1903575208178555E-4
 1800000000 Hops: 6.0 Delay: 36960 RDP: 2.0528533420077495E-4

1800000000 Hops: 6.0 Delay: 36939 RDP: 2.0516846575511193E-4
1800000000 Hops: 5.0 Delay: 45832 RDP: 12.709927897947864
1800000000 Hops: 6.0 Delay: 39235 RDP: 2.1791955469784434E-4
1800000000 Hops: 6.0 Delay: 49817 RDP: 112.7081447963801
1800000000 Hops: 4.0 Delay: 28847 RDP: 1.6021301516395888E-4
1800000000 Hops: 3.0 Delay: 12469 RDP: 6.9266241467084E-5
1800000000 Hops: 6.0 Delay: 46007 RDP: 2.5552779711656093E-4
1800000000 Hops: 6.0 Delay: 42490 RDP: 2.3599541736778198E-4
1800000000 Hops: 5.0 Delay: 33640 RDP: 11.845070422535212
1800000000 Hops: 3.0 Delay: 24415 RDP: 33.172554347826086
1800000000 Hops: 5.0 Delay: 32050 RDP: 1.7801095985431293E-4
1800000000 Hops: 5.0 Delay: 41160 RDP: 2.2860197103885172E-4
1800000000 Hops: 4.0 Delay: 27888 RDP: 1.548976449159447E-4
1800000000 Hops: 6.0 Delay: 34779 RDP: 1.9316685966505126E-4
1800000000 Hops: 4.0 Delay: 25174 RDP: 1.3981989293830726E-4
1800000000 Hops: 6.0 Delay: 47402 RDP: 43.3290676416819
1800000000 Hops: 6.0 Delay: 54238 RDP: 53.174509803921566
1800000000 Hops: 3.0 Delay: 20317 RDP: 1.1285941267888317E-4
1800000000 Hops: 5.0 Delay: 35868 RDP: 1.992187280000856E-4
1800000000 Hops: 4.0 Delay: 34587 RDP: 4.347831552482715
1800000000 Hops: 5.0 Delay: 34353 RDP: 1.9080470508306557E-4
1800000000 Hops: 6.0 Delay: 31742 RDP: 1.7631279629750905E-4
1800000000 Hops: 4.0 Delay: 36219 RDP: 11.63849614395887
1800000000 Hops: 4.0 Delay: 32747 RDP: 1.818823577112271E-4
1800000000 Hops: 6.0 Delay: 39272 RDP: 2.18125811726754E-4
1800000000 Hops: 5.0 Delay: 36088 RDP: 2.0043270871415137E-4
1800000000 Hops: 6.0 Delay: 36099 RDP: 2.0050056547169092E-4
1800000000 Hops: 3.0 Delay: 15045 RDP: 8.357478502573829E-5
1800000000 Hops: 6.0 Delay: 40909 RDP: 2.2721868318883285E-4
1800000000 Hops: 5.0 Delay: 29750 RDP: 1.652326509048973E-4
1800000000 Hops: 6.0 Delay: 40009 RDP: 2.2222014493125758E-4
1800000000 Hops: 4.0 Delay: 32080 RDP: 1.7817841507911485E-4
1800000000 Hops: 3.0 Delay: 16014 RDP: 8.89571566524141E-5
1800000000 Hops: 5.0 Delay: 37433 RDP: 2.079132159916606E-4
1800000000 Hops: 3.0 Delay: 18156 RDP: 1.0085523080410716E-4
1800000000 Hops: 6.0 Delay: 38929 RDP: 2.1621578509751656E-4
1800000000 Hops: 3.0 Delay: 10140 RDP: 5.632924163980866E-5
1800000000 Hops: 3.0 Delay: 17055 RDP: 9.474057804951298E-5
1800000000 Hops: 3.0 Delay: 19540 RDP: 6.284979092955934

1800000000 Hops: 5.0 Delay: 42118 RDP: 2.3392221975669034E-4
1800000000 Hops: 6.0 Delay: 52594 RDP: 9.508949557042126
1800000000 Hops: 5.0 Delay: 32900 RDP: 1.8272314457267914E-4
1800000000 Hops: 5.0 Delay: 41000 RDP: 2.277205667807169E-4
1800000000 Hops: 4.0 Delay: 28555 RDP: 1.5861231603887517E-4
1800000000 Hops: 3.0 Delay: 21345 RDP: 3.542738589211618
1800000000 Hops: 5.0 Delay: 33778 RDP: 1.876050574853321E-4
1800000000 Hops: 3.0 Delay: 14867 RDP: 8.258534629212793E-5
1800000000 Hops: 3.0 Delay: 20534 RDP: 1.1406337220755336E-4
1800000000 Hops: 4.0 Delay: 28827 RDP: 1.601169874353461E-4
1800000000 Hops: 4.0 Delay: 23397 RDP: 1.2994625389170896E-4
1800000000 Hops: 4.0 Delay: 32653 RDP: 1.8136669874794159E-4
1800000000 Hops: 5.0 Delay: 29882 RDP: 1.65976921709427E-4
1800000000 Hops: 6.0 Delay: 42448 RDP: 2.3576422029462606E-4
1800000000 Hops: 5.0 Delay: 43925 RDP: 2.4395952332447433E-4
1800000000 Hops: 3.0 Delay: 22395 RDP: 1.2440052846255485E-4
1800000000 Hops: 5.0 Delay: 34092 RDP: 1.8936312048026914E-4
1800000000 Hops: 5.0 Delay: 34952 RDP: 1.9413830945946468E-4
1800000000 Hops: 6.0 Delay: 32375 RDP: 1.7982758825153456E-4
1800000000 Hops: 3.0 Delay: 27755 RDP: 3.140771755120516
1800000000 Hops: 5.0 Delay: 37094 RDP: 2.0601849595556655E-4
1800000000 Hops: 6.0 Delay: 46300 RDP: 2.5715601026415723E-4
1800000000 Hops: 5.0 Delay: 33121 RDP: 1.8396042699680613E-4
1800000000 Hops: 6.0 Delay: 36866 RDP: 2.047608696190689E-4
1800000000 Hops: 6.0 Delay: 44846 RDP: 2.490776030858919E-4
1800000000 Hops: 5.0 Delay: 44748 RDP: 2.4853679019045484E-4
1800000000 Hops: 4.0 Delay: 28003 RDP: 1.555444152014624E-4
1800000000 Hops: 6.0 Delay: 39342 RDP: 2.1850922301981502E-4
1800000000 Hops: 3.0 Delay: 15713 RDP: 8.728510251388928E-5
1800000000 Hops: 5.0 Delay: 36851 RDP: 2.046764665247115E-4
1800000000 Hops: 5.0 Delay: 39932 RDP: 2.2178365477680155E-4
1800000000 Hops: 5.0 Delay: 29399 RDP: 1.6328519299944351E-4
1800000000 Hops: 4.0 Delay: 26122 RDP: 2.435163605854386
1800000000 Hops: 5.0 Delay: 37667 RDP: 2.092133453797682E-4
1800000000 Hops: 4.0 Delay: 31826 RDP: 1.7676926000675307E-4
1800000000 Hops: 6.0 Delay: 42086 RDP: 2.337543490619825E-4
1800000000 Hops: 5.0 Delay: 47261 RDP: 4.28243928959768
1800000000 Hops: 4.0 Delay: 27718 RDP: 1.5395559342588398E-4
1800000000 Hops: 4.0 Delay: 28336 RDP: 1.573782201461928E-4

180000000 Hops: 5.0 Delay: 43268 RDP: 2.403089799869073E-4
180000000 Hops: 5.0 Delay: 33449 RDP: 1.85788072802308E-4
180000000 Hops: 5.0 Delay: 32208 RDP: 1.7889201424726482E-4
180000000 Hops: 5.0 Delay: 33367 RDP: 1.8533620419022902E-4
180000000 Hops: 6.0 Delay: 35864 RDP: 4.59853827413771
180000000 Hops: 5.0 Delay: 34421 RDP: 1.911739877900799E-4
180000000 Hops: 5.0 Delay: 31516 RDP: 1.7505397631835637E-4
180000000 Hops: 5.0 Delay: 40174 RDP: 2.2313067285672573E-4
180000000 Hops: 6.0 Delay: 46995 RDP: 2.6101278302815184E-4
180000000 Hops: 4.0 Delay: 34415 RDP: 1.9114318302847593E-4
180000000 Hops: 5.0 Delay: 33974 RDP: 1.887016940756516E-4
180000000 Hops: 5.0 Delay: 33278 RDP: 1.8484060941190185E-4
180000000 Hops: 3.0 Delay: 24570 RDP: 4.8044583496284705
180000000 Hops: 6.0 Delay: 39665 RDP: 2.2030353845306204E-4
180000000 Hops: 6.0 Delay: 39867 RDP: 2.2142802307348095E-4
180000000 Hops: 5.0 Delay: 33979 RDP: 1.8873533494953598E-4
180000000 Hops: 4.0 Delay: 30690 RDP: 1.7046931647008714E-4
180000000 Hops: 6.0 Delay: 48548 RDP: 15.034995354598948
180000000 Hops: 3.0 Delay: 19554 RDP: 4.603107344632768
180000000 Hops: 3.0 Delay: 28069 RDP: 3.2604251364850736
180000000 Hops: 3.0 Delay: 23896 RDP: 11.145522388059701
180000000 Hops: 5.0 Delay: 34487 RDP: 1.915371673827294E-4
180000000 Hops: 5.0 Delay: 32117 RDP: 1.783824676399835E-4
180000000 Hops: 3.0 Delay: 18439 RDP: 1.024283473048121E-4
180000000 Hops: 3.0 Delay: 6551 RDP: 3.639231973951032E-5
180000000 Hops: 5.0 Delay: 35372 RDP: 1.9645126987151635E-4
180000000 Hops: 6.0 Delay: 29929 RDP: 1.6624023113685362E-4
180000000 Hops: 4.0 Delay: 33660 RDP: 1.869544900839333E-4
180000000 Hops: 4.0 Delay: 27620 RDP: 1.5341732878390037E-4
180000000 Hops: 3.0 Delay: 22543 RDP: 18.25344129554656
180000000 Hops: 6.0 Delay: 39870 RDP: 2.2144188503773723E-4
180000000 Hops: 4.0 Delay: 27881 RDP: 1.5486891086295733E-4
180000000 Hops: 6.0 Delay: 39695 RDP: 2.2047607001502397E-4
180000000 Hops: 6.0 Delay: 40463 RDP: 2.2473487097572997E-4
180000000 Hops: 3.0 Delay: 6599 RDP: 3.6658553362379566E-5
180000000 Hops: 6.0 Delay: 34052 RDP: 34.43073811931244
180000000 Hops: 5.0 Delay: 44359 RDP: 2.463725736044937E-4
180000000 Hops: 6.0 Delay: 48142 RDP: 10.373195432018962
180000000 Hops: 3.0 Delay: 16217 RDP: 9.00842328961777E-5

180000000 Hops: 5.0 Delay: 40915 RDP: 2.2724316089588358E-4
180000000 Hops: 5.0 Delay: 43782 RDP: 7.85045723507262
180000000 Hops: 6.0 Delay: 44254 RDP: 2.457867666669757E-4
180000000 Hops: 6.0 Delay: 49078 RDP: 52.04453870625663
180000000 Hops: 4.0 Delay: 26448 RDP: 1.4689013376105052E-4
180000000 Hops: 6.0 Delay: 37951 RDP: 2.107879262235482E-4
180000000 Hops: 5.0 Delay: 34304 RDP: 48.24753867791843
180000000 Hops: 5.0 Delay: 30285 RDP: 1.682027256894854E-4
180000000 Hops: 5.0 Delay: 43093 RDP: 2.393404709035012E-4
180000000 Hops: 5.0 Delay: 38077 RDP: 2.1147896867288176E-4
180000000 Hops: 4.0 Delay: 30417 RDP: 21.270629370629372
180000000 Hops: 3.0 Delay: 12839 RDP: 7.132104824064823E-5
180000000 Hops: 3.0 Delay: 27530 RDP: 3.0153340635268346
180000000 Hops: 5.0 Delay: 36814 RDP: 2.044694384366898E-4
180000000 Hops: 5.0 Delay: 36453 RDP: 2.0246166233215906E-4
180000000 Hops: 3.0 Delay: 26859 RDP: 2.572208389197472
180000000 Hops: 6.0 Delay: 51517 RDP: 2.8612165355977864E-4
180000000 Hops: 4.0 Delay: 36029 RDP: 2.0011228149095753E-4
180000000 Hops: 5.0 Delay: 38913 RDP: 2.161353236741863E-4
180000000 Hops: 6.0 Delay: 48966 RDP: 74.87155963302752
180000000 Hops: 5.0 Delay: 34348 RDP: 1.9077819697363395E-4
180000000 Hops: 6.0 Delay: 46996 RDP: 2.610178035902789E-4
180000000 Hops: 5.0 Delay: 41317 RDP: 2.2948399376775748E-4
180000000 Hops: 5.0 Delay: 36369 RDP: 2.019995999775856E-4
180000000 Hops: 6.0 Delay: 38071 RDP: 2.114528603280237E-4
180000000 Hops: 4.0 Delay: 29165 RDP: 1.6199890867225845E-4
180000000 Hops: 6.0 Delay: 39361 RDP: 2.1862157853355492E-4
180000000 Hops: 4.0 Delay: 26916 RDP: 1.4949843790628537E-4
180000000 Hops: 4.0 Delay: 30883 RDP: 1.715295904484006E-4
180000000 Hops: 3.0 Delay: 16894 RDP: 9.384651448662382E-5
180000000 Hops: 6.0 Delay: 46430 RDP: 2.5787237054950013E-4
180000000 Hops: 3.0 Delay: 22830 RDP: 4.216845216106391
180000000 Hops: 5.0 Delay: 29527 RDP: 1.6400192376639375E-4
180000000 Hops: 3.0 Delay: 27729 RDP: 6.789666993143976
180000000 Hops: 3.0 Delay: 14002 RDP: 7.778211147424243E-5
180000000 Hops: 6.0 Delay: 36335 RDP: 2.0181000721145164E-4
180000000 Hops: 4.0 Delay: 28598 RDP: 1.5885056226394635E-4
180000000 Hops: 5.0 Delay: 35223 RDP: 1.9563178653126886E-4
180000000 Hops: 5.0 Delay: 37947 RDP: 44.022041763341065

180000000 Hops: 6.0 Delay: 42240 RDP: 2.3460989758839187E-4
180000000 Hops: 4.0 Delay: 28851 RDP: 1.6024614554449064E-4
180000000 Hops: 5.0 Delay: 40416 RDP: 2.2447810049670613E-4
180000000 Hops: 6.0 Delay: 43817 RDP: 2.433682972138926E-4
180000000 Hops: 5.0 Delay: 30183 RDP: 1.6763698729424604E-4
180000000 Hops: 5.0 Delay: 31146 RDP: 1.7298952565806628E-4
180000000 Hops: 5.0 Delay: 32624 RDP: 18.55745164960182
180000000 Hops: 3.0 Delay: 14687 RDP: 8.158564905822236E-5
180000000 Hops: 6.0 Delay: 38534 RDP: 2.140287913436346E-4
180000000 Hops: 5.0 Delay: 44732 RDP: 2.48448615242217E-4
180000000 Hops: 4.0 Delay: 28969 RDP: 1.609059389282839E-4
180000000 Hops: 5.0 Delay: 29958 RDP: 1.663934673073828E-4
180000000 Hops: 5.0 Delay: 45466 RDP: 2.525244937400713E-4
180000000 Hops: 4.0 Delay: 29661 RDP: 1.6474756023115758E-4
180000000 Hops: 5.0 Delay: 43097 RDP: 9.288146551724138
180000000 Hops: 5.0 Delay: 37281 RDP: 2.0705147075966823E-4
180000000 Hops: 5.0 Delay: 41738 RDP: 13.174873737373737
180000000 Hops: 5.0 Delay: 36784 RDP: 11.919637070641608
180000000 Hops: 3.0 Delay: 15498 RDP: 8.609081171897815E-5
180000000 Hops: 5.0 Delay: 42629 RDP: 2.3675765147673065E-4
180000000 Hops: 6.0 Delay: 44769 RDP: 678.3181818181819
180000000 Hops: 3.0 Delay: 12129 RDP: 6.737659717087396E-5
180000000 Hops: 6.0 Delay: 35821 RDP: 1.9895120871704102E-4
180000000 Hops: 3.0 Delay: 19402 RDP: 1.0777726690694076E-4
180000000 Hops: 3.0 Delay: 20155 RDP: 1.1195880271573004E-4
180000000 Hops: 6.0 Delay: 44485 RDP: 2.470726761569308E-4
180000000 Hops: 5.0 Delay: 39069 RDP: 13.556210964607912
180000000 Hops: 5.0 Delay: 32116 RDP: 1.7837552152873576E-4
180000000 Hops: 6.0 Delay: 43467 RDP: 2.4142308486126685E-4
180000000 Hops: 6.0 Delay: 41843 RDP: 2.323987856301182E-4
180000000 Hops: 6.0 Delay: 35569 RDP: 1.9755712223196127E-4
180000000 Hops: 3.0 Delay: 28542 RDP: 3.943354517822603
180000000 Hops: 6.0 Delay: 37361 RDP: 2.0751223852040224E-4
180000000 Hops: 4.0 Delay: 26585 RDP: 1.4766623609151108E-4
180000000 Hops: 5.0 Delay: 35558 RDP: 1.97484167887868E-4
180000000 Hops: 4.0 Delay: 24896 RDP: 3.3194666666666666
180000000 Hops: 6.0 Delay: 39041 RDP: 2.168432826812656E-4
180000000 Hops: 5.0 Delay: 39386 RDP: 2.1875764917229394E-4
180000000 Hops: 5.0 Delay: 44788 RDP: 2.4875603515115834E-4

1800000000 Hops: 3.0 Delay: 14544 RDP: 8.079281482566817E-5
 1800000000 Hops: 4.0 Delay: 36078 RDP: 6.485349631493798
 1800000000 Hops: 6.0 Delay: 38678 RDP: 2.148276107567192E-4
 1800000000 Hops: 3.0 Delay: 20046 RDP: 1.113537725184411E-4
 1800000000 Hops: 3.0 Delay: 26002 RDP: 18.22144358794674
 1800000000 Hops: 5.0 Delay: 31532 RDP: 1.7513602826757258E-4
 1800000000 Hops: 4.0 Delay: 30854 RDP: 1.7136480262928058E-4
 1800000000 Hops: 6.0 Delay: 37065 RDP: 2.0586468011647393E-4
 1800000000 Hops: 3.0 Delay: 8894 RDP: 4.940737481563339E-5
 1800000000 Hops: 5.0 Delay: 44193 RDP: 47.21474358974359
 1800000000 Hops: 5.0 Delay: 34594 RDP: 1.921309582028081E-4
 1800000000 Hops: 3.0 Delay: 20830 RDP: 152.04379562043795
 1800000000 Hops: 5.0 Delay: 40245 RDP: 2.2352165625764996E-4
 1800000000 Hops: 5.0 Delay: 28508 RDP: 1.5833369591757105E-4
 1800000000 Hops: 5.0 Delay: 46426 RDP: 2.5785244591784327E-4
 1800000000 Hops: 5.0 Delay: 31564 RDP: 1.7530704420071314E-4
 1800000000 Hops: 5.0 Delay: 32638 RDP: 1.8128115197012587E-4
 1800000000 Hops: 3.0 Delay: 21714 RDP: 9.838695061169007
 1800000000 Hops: 4.0 Delay: 35693 RDP: 1.9824570023221734E-4
 1800000000 Hops: 6.0 Delay: 40219 RDP: 2.2338560149468345E-4
 1800000000 Hops: 5.0 Delay: 43463 RDP: 249.78735632183907
 1800000000 Hops: 6.0 Delay: 57196 RDP: 5.197746274082152
 1800000000 Hops: 5.0 Delay: 38038 RDP: 2.1127483210363232E-4
 1800000000 Hops: 4.0 Delay: 25189 RDP: 2.781470848056537
 1800000000 Hops: 4.0 Delay: 33192 RDP: 3.2785460292374555
 1800000000 Hops: 4.0 Delay: 32990 RDP: 7.782495871667846
 1800000000 Hops: 6.0 Delay: 35505 RDP: 1.9720511721090677E-4
 1800000000 Hops: 5.0 Delay: 38219 RDP: 2.1227335324859766E-4
 1800000000 Hops: 5.0 Delay: 40972 RDP: 2.2755998330256685E-4
 1800000000 Hops: 5.0 Delay: 33941 RDP: 1.8851423390494674E-4
 1800000000 Hops: 5.0 Delay: 38197 RDP: 2.1216026287543602E-4
 1800000000 Hops: 5.0 Delay: 39341 RDP: 2.1850183520491726E-4
 1800000000 Hops: 4.0 Delay: 37126 RDP: 10.951622418879056
 1800000000 Hops: 5.0 Delay: 31363 RDP: 1.7419912019752068E-4
 1800000000 Hops: 6.0 Delay: 49063 RDP: 2.724957811001618E-4
 1800000000 Hops: 5.0 Delay: 34577 RDP: 1.9204560297992658E-4
 1800000000 Hops: 5.0 Delay: 34066 RDP: 1.8920176444280355E-4
 1800000000 Hops: 4.0 Delay: 23057 RDP: 1.2805780070477277E-4
 1800000000 Hops: 5.0 Delay: 34105 RDP: 1.8943425433559074E-4

1800000000 Hops: 5.0 Delay: 43348 RDP: 2.4075349646161841E-4
 1800000000 Hops: 5.0 Delay: 40513 RDP: 2.2501558204996054E-4
 1800000000 Hops: 3.0 Delay: 10806 RDP: 6.0028523381151524E-5
 1800000000 Hops: 5.0 Delay: 37153 RDP: 2.0635133101135091E-4
 1800000000 Hops: 6.0 Delay: 40473 RDP: 2.2479336705772592E-4
 1800000000 Hops: 5.0 Delay: 33446 RDP: 1.8576608450832771E-4
 1800000000 Hops: 3.0 Delay: 14230 RDP: 592.9166666666666
 1800000000 Hops: 5.0 Delay: 33356 RDP: 1.8527305705448688E-4
 1800000000 Hops: 3.0 Delay: 22251 RDP: 6.388458225667528
 1800000000 Hops: 5.0 Delay: 38032 RDP: 2.1123278194145888E-4
 1800000000 Hops: 5.0 Delay: 37170 RDP: 2.0644781802013076E-4
 1800000000 Hops: 3.0 Delay: 11475 RDP: 6.374451478450279E-5
 1800000000 Hops: 3.0 Delay: 20484 RDP: 8.761334473909324
 1800000000 Hops: 6.0 Delay: 34790 RDP: 1.932261316532224E-4
 1800000000 Hops: 6.0 Delay: 45707 RDP: 2.538624279474368E-4
 1800000000 Hops: 5.0 Delay: 39793 RDP: 2.2101469087032927E-4
 1800000000 Hops: 3.0 Delay: 15595 RDP: 5.528181495923431
 1800000000 Hops: 6.0 Delay: 40417 RDP: 2.2448379682375171E-4
 1800000000 Hops: 4.0 Delay: 28902 RDP: 1.605186065041093E-4
 1800000000 Hops: 3.0 Delay: 24525 RDP: 3.644672313865359
 1800000000 Hops: 5.0 Delay: 40543 RDP: 2.2518524475947065E-4
 1800000000 Hops: 5.0 Delay: 32671 RDP: 1.8145834110330238E-4
 1800000000 Hops: 6.0 Delay: 40473 RDP: 2.2478850785485126E-4
 1800000000 Hops: 4.0 Delay: 30349 RDP: 1.6856190644988037E-4
 1800000000 Hops: 3.0 Delay: 21056 RDP: 11.678313921242374
 1800000000 Hops: 4.0 Delay: 25293 RDP: 1.4047617050840178E-4
 1800000000 Hops: 3.0 Delay: 15193 RDP: 8.439791191799951E-5
 1800000000 Hops: 3.0 Delay: 12129 RDP: 6.737704443486917E-5
 1800000000 Hops: 4.0 Delay: 24913 RDP: 1.3838060860694835E-4
 1800000000 Hops: 5.0 Delay: 37898 RDP: 2.1048626136433086E-4
 1800000000 Hops: 5.0 Delay: 36521 RDP: 2.028419264158962E-4
 1800000000 Hops: 4.0 Delay: 32578 RDP: 1.8093305395362242E-4
 1800000000 Hops: 4.0 Delay: 33415 RDP: 14.904103479036575
 1800000000 Hops: 4.0 Delay: 36179 RDP: 2.00945128044575E-4
 1800000000 Hops: 4.0 Delay: 32175 RDP: 1.7870739814192073E-4
 1800000000 Hops: 6.0 Delay: 37451 RDP: 2.0801477813055784E-4
 1800000000 Hops: 3.0 Delay: 19071 RDP: 7.047671840354767
 1800000000 Hops: 3.0 Delay: 17482 RDP: 9.711210475944137E-5
 1800000000 Hops: 3.0 Delay: 27739 RDP: 2.7711288711288713

1800000000 Hops: 4.0 Delay: 36360 RDP: 2.0195310761037457E-4
 1800000000 Hops: 5.0 Delay: 46676 RDP: 2.592430785261092E-4
 1800000000 Hops: 5.0 Delay: 43268 RDP: 2.4030841675842075E-4
 1800000000 Hops: 5.0 Delay: 39968 RDP: 2.2198294037109896E-4
 1800000000 Hops: 5.0 Delay: 35824 RDP: 1.989755602425058E-4
 1800000000 Hops: 6.0 Delay: 36469 RDP: 2.025604779718549E-4
 1800000000 Hops: 5.0 Delay: 42685 RDP: 2.3707361857046948E-4
 1800000000 Hops: 6.0 Delay: 46749 RDP: 8.797327813323298
 1800000000 Hops: 4.0 Delay: 30331 RDP: 1.6845870251097885E-4
 1800000000 Hops: 4.0 Delay: 28300 RDP: 1.5718001327610153E-4
 1800000000 Hops: 4.0 Delay: 28561 RDP: 1.5863232002470134E-4
 1800000000 Hops: 4.0 Delay: 28535 RDP: 19.885017421602786
 1800000000 Hops: 5.0 Delay: 37730 RDP: 5.446016166281755
 1800000000 Hops: 5.0 Delay: 32164 RDP: 1.7864251626904072E-4
 1800000000 Hops: 4.0 Delay: 25016 RDP: 1.3894078325045087E-4
 1800000000 Hops: 5.0 Delay: 38988 RDP: 2.1654668019215114E-4
 1800000000 Hops: 3.0 Delay: 26532 RDP: 15.100739897552646
 1800000000 Hops: 3.0 Delay: 18679 RDP: 1.0376054397299806E-4
 1800000000 Hops: 5.0 Delay: 43018 RDP: 100.04186046511627
 1800000000 Hops: 3.0 Delay: 21960 RDP: 12.441926345609065
 1800000000 Hops: 6.0 Delay: 36868 RDP: 2.047756312155511E-4
 1800000000 Hops: 3.0 Delay: 19889 RDP: 12.83989670755326
 1800000000 Hops: 5.0 Delay: 42227 RDP: 2.3453165510855733E-4
 1800000000 Hops: 5.0 Delay: 31326 RDP: 1.739911085911301E-4
 1800000000 Hops: 4.0 Delay: 35316 RDP: 28.16267942583732
 1800000000 Hops: 3.0 Delay: 23258 RDP: 5.746973066468989
 1800000000 Hops: 4.0 Delay: 36450 RDP: 3.460224036453389
 1800000000 Hops: 5.0 Delay: 31896 RDP: 1.771639756740353E-4
 1800000000 Hops: 3.0 Delay: 22653 RDP: 4.000883080183681
 1800000000 Hops: 6.0 Delay: 33417 RDP: 1.856146847438421E-4
 1800000000 Hops: 6.0 Delay: 36805 RDP: 2.0442039256735563E-4
 1800000000 Hops: 6.0 Delay: 44992 RDP: 2.498881690459695E-4
 1800000000 Hops: 5.0 Delay: 36464 RDP: 2.02520104552448E-4
 1800000000 Hops: 3.0 Delay: 17590 RDP: 9.771135726213555E-5
 1800000000 Hops: 5.0 Delay: 36036 RDP: 2.0015032046656774E-4
 1800000000 Hops: 3.0 Delay: 19954 RDP: 36.21415607985481
 1800000000 Hops: 4.0 Delay: 28387 RDP: 5.856612337528368
 1800000000 Hops: 5.0 Delay: 31762 RDP: 1.7640875921197993E-4
 1800000000 Hops: 3.0 Delay: 11361 RDP: 6.311126749773223E-5

1800000000 Hops: 5.0 Delay: 36719 RDP: 2.039350359025412E-4
 1800000000 Hops: 4.0 Delay: 28661 RDP: 7.976899526857779
 1800000000 Hops: 4.0 Delay: 28211 RDP: 1.5669625049217876E-4
 1800000000 Hops: 5.0 Delay: 46619 RDP: 2.589249029693936E-4
 1800000000 Hops: 5.0 Delay: 30700 RDP: 1.7052000024083867E-4
 1800000000 Hops: 5.0 Delay: 33974 RDP: 1.8868797327884165E-4
 1800000000 Hops: 5.0 Delay: 40834 RDP: 89.15720524017468
 1800000000 Hops: 5.0 Delay: 39140 RDP: 2.1738416260846333E-4
 1800000000 Hops: 3.0 Delay: 23347 RDP: 8.468262604280014
 1800000000 Hops: 6.0 Delay: 50861 RDP: 9.767812560015363
 1800000000 Hops: 5.0 Delay: 41936 RDP: 2.3291174471024416E-4
 1800000000 Hops: 4.0 Delay: 39419 RDP: 4.594289044289044
 1800000000 Hops: 5.0 Delay: 33347 RDP: 1.8521570650963643E-4
 1800000000 Hops: 5.0 Delay: 31465 RDP: 1.7476687576224102E-4
 1800000000 Hops: 6.0 Delay: 53086 RDP: 2.94834644866905E-4
 1800000000 Hops: 3.0 Delay: 20683 RDP: 1.1489100333341668E-4
 1800000000 Hops: 5.0 Delay: 39622 RDP: 2.2006344205427486E-4
 1800000000 Hops: 6.0 Delay: 38760 RDP: 2.152859596579106E-4
 1800000000 Hops: 5.0 Delay: 37042 RDP: 2.0573634725473916E-4
 1800000000 Hops: 5.0 Delay: 30215 RDP: 1.6782794551194552E-4
 1800000000 Hops: 6.0 Delay: 44509 RDP: 2.4720929784222644E-4
 1800000000 Hops: 3.0 Delay: 17776 RDP: 18.751054852320674
 1800000000 Hops: 6.0 Delay: 43273 RDP: 2.4034485379223196E-4
 1800000000 Hops: 6.0 Delay: 40471 RDP: 2.2478794569628452E-4
 1800000000 Hops: 5.0 Delay: 46179 RDP: 2.5648216616665264E-4
 1800000000 Hops: 3.0 Delay: 26952 RDP: 5.123954372623574
 1800000000 Hops: 4.0 Delay: 27499 RDP: 1.5273055732618364E-4
 1800000000 Hops: 5.0 Delay: 32628 RDP: 1.8121528810548235E-4
 1800000000 Hops: 6.0 Delay: 30941 RDP: 1.7186241501903206E-4
 1800000000 Hops: 3.0 Delay: 10045 RDP: 5.580115346456002E-5
 1800000000 Hops: 4.0 Delay: 26764 RDP: 1.486662651870222E-4
 1800000000 Hops: 5.0 Delay: 34992 RDP: 1.9434343310473765E-4
 1800000000 Hops: 5.0 Delay: 39276 RDP: 2.181514516063975E-4
 1800000000 Hops: 5.0 Delay: 32671 RDP: 1.814519979822178E-4
 1800000000 Hops: 5.0 Delay: 31684 RDP: 1.7597170270161773E-4
 1800000000 Hops: 3.0 Delay: 19771 RDP: 6.6145868183338905
 1800000000 Hops: 4.0 Delay: 40536 RDP: 7.487255264130034
 1800000000 Hops: 6.0 Delay: 47773 RDP: 146.0948012232416
 1800000000 Hops: 3.0 Delay: 16290 RDP: 6.516

1800000000 Hops: 5.0 Delay: 36732 RDP: 2.0401967980096584E-4
 1800000000 Hops: 6.0 Delay: 46242 RDP: 63.95850622406639
 1800000000 Hops: 4.0 Delay: 31588 RDP: 4.323569668765399
 1800000000 Hops: 3.0 Delay: 20424 RDP: 1.1345300503397715E-4
 1800000000 Hops: 4.0 Delay: 38762 RDP: 2.1529334098174008E-4
 1800000000 Hops: 3.0 Delay: 25603 RDP: 7.043466299862448
 1800000000 Hops: 4.0 Delay: 25878 RDP: 1.4372797110440171E-4
 1800000000 Hops: 5.0 Delay: 37083 RDP: 2.0595895811082045E-4
 1800000000 Hops: 5.0 Delay: 39962 RDP: 2.2194588984531462E-4
 1800000000 Hops: 6.0 Delay: 36266 RDP: 2.0143203591962104E-4
 1800000000 Hops: 4.0 Delay: 27525 RDP: 1.528742950079003E-4
 1800000000 Hops: 6.0 Delay: 42750 RDP: 2.3743852189027102E-4
 1800000000 Hops: 5.0 Delay: 39454 RDP: 2.1913688892257472E-4
 1800000000 Hops: 5.0 Delay: 33463 RDP: 1.85849859450176E-4
 1800000000 Hops: 5.0 Delay: 39422 RDP: 2.1894847360195483E-4
 1800000000 Hops: 4.0 Delay: 23660 RDP: 1.31411675483033E-4
 1800000000 Hops: 5.0 Delay: 42662 RDP: 2.3694943448969918E-4
 1800000000 Hops: 5.0 Delay: 31894 RDP: 1.7714233686688436E-4
 1800000000 Hops: 5.0 Delay: 40855 RDP: 2.269113620736767E-4
 1800000000 Hops: 4.0 Delay: 28625 RDP: 1.589843494362365E-4
 1800000000 Hops: 6.0 Delay: 39567 RDP: 2.1976812110964755E-4
 1800000000 Hops: 6.0 Delay: 33826 RDP: 1.8787905805240724E-4
 1800000000 Hops: 4.0 Delay: 34271 RDP: 1.903503476714025E-4
 1800000000 Hops: 4.0 Delay: 32757 RDP: 1.819368302795139E-4
 1800000000 Hops: 5.0 Delay: 35075 RDP: 1.9480921393651843E-4
 1800000000 Hops: 3.0 Delay: 30094 RDP: 3.0184553660982947
 1800000000 Hops: 3.0 Delay: 21770 RDP: 20.046040515653775
 1800000000 Hops: 5.0 Delay: 29309 RDP: 1.627837185556427E-4
 1800000000 Hops: 5.0 Delay: 35198 RDP: 1.9550230935228325E-4
 1800000000 Hops: 4.0 Delay: 23322 RDP: 1.2952892985216457E-4
 1800000000 Hops: 3.0 Delay: 24397 RDP: 1.355106334161479E-4
 1800000000 Hops: 5.0 Delay: 44602 RDP: 2.4771936369702994E-4
 1800000000 Hops: 5.0 Delay: 31726 RDP: 1.7621639048379057E-4
 1800000000 Hops: 3.0 Delay: 12033 RDP: 6.684410657793671E-5
 1800000000 Hops: 5.0 Delay: 34231 RDP: 1.9013344873054138E-4
 1800000000 Hops: 4.0 Delay: 32494 RDP: 1.8047170317939945E-4
 1800000000 Hops: 5.0 Delay: 34461 RDP: 1.9139611880262175E-4
 1800000000 Hops: 5.0 Delay: 35244 RDP: 1.9575029682463292E-4
 1800000000 Hops: 5.0 Delay: 43726 RDP: 31.299928418038654

1800000000 Hops: 6.0 Delay: 37260 RDP: 2.0694886753369615E-4
 1800000000 Hops: 5.0 Delay: 33475 RDP: 1.859313421178019E-4
 1800000000 Hops: 4.0 Delay: 26789 RDP: 7.563241106719367
 1800000000 Hops: 5.0 Delay: 31811 RDP: 1.7668378155301796E-4
 1800000000 Hops: 5.0 Delay: 36023 RDP: 2.000741701269386E-4
 1800000000 Hops: 6.0 Delay: 54085 RDP: 3.003792615158778E-4
 1800000000 Hops: 5.0 Delay: 33205 RDP: 1.8442947351947678E-4
 1800000000 Hops: 5.0 Delay: 29744 RDP: 1.65199799198711E-4
 1800000000 Hops: 5.0 Delay: 29701 RDP: 1.6497396762336588E-4
 1800000000 Hops: 3.0 Delay: 23506 RDP: 4.115196078431373
 1800000000 Hops: 5.0 Delay: 32315 RDP: 1.7948677900885706E-4
 1800000000 Hops: 6.0 Delay: 50943 RDP: 8.366398423386434
 1800000000 Hops: 4.0 Delay: 28737 RDP: 1.5961709848219454E-4
 1800000000 Hops: 5.0 Delay: 30013 RDP: 53.49910873440285
 1800000000 Hops: 6.0 Delay: 43798 RDP: 2.4326085696815358E-4
 1800000000 Hops: 3.0 Delay: 27500 RDP: 3.2126168224299065
 1800000000 Hops: 3.0 Delay: 20670 RDP: 21.98936170212766
 1800000000 Hops: 5.0 Delay: 40428 RDP: 2.2454775023349568E-4
 1800000000 Hops: 5.0 Delay: 36481 RDP: 2.0262403935135357E-4
 1800000000 Hops: 4.0 Delay: 26544 RDP: 1.474253417053273E-4
 1800000000 Hops: 6.0 Delay: 51780 RDP: 11.846259437199725
 1800000000 Hops: 3.0 Delay: 19071 RDP: 7.047671840354767
 1800000000 Hops: 4.0 Delay: 25291 RDP: 632.275
 1800000000 Hops: 5.0 Delay: 36713 RDP: 2.0390631468747742E-4
 1800000000 Hops: 5.0 Delay: 46650 RDP: 4.890962465925771
 1800000000 Hops: 5.0 Delay: 29091 RDP: 1.615812274151271E-4
 1800000000 Hops: 3.0 Delay: 12419 RDP: 18.42581602373887
 1800000000 Hops: 4.0 Delay: 31469 RDP: 1.7478554376421955E-4
 1800000000 Hops: 6.0 Delay: 45643 RDP: 2.5350489695503313E-4
 1800000000 Hops: 4.0 Delay: 26220 RDP: 1.4562472270139747E-4
 1800000000 Hops: 6.0 Delay: 41081 RDP: 2.2817025478893296E-4
 1800000000 Hops: 3.0 Delay: 18872 RDP: 1.04831798235184E-4
 1800000000 Hops: 5.0 Delay: 43208 RDP: 8.97548815953469

Failed Requests:

1800000000 data available: true
 1800000000 data available: true
 1800000000 data available: true

[illegible]

[illegible]

1800000000 data available: true
1800000000 data available: true
1800000000 data available: true
1800000000 data available: true
1800000000 data available: true
1800000000 data available: true
1800000000 data available: true
1800000000 data available: true
1800000000 data available: true
1800000000 data available: true

second success

180 82.82828282828282

second hops

180 4.65

second delay

180 33

second rdp

180 7.295063211660706

second queries

180 990

Loadbalance Heads

Peer Messages

1	6.027892122229336
2	5.902841123313996
3	5.655199235762595
4	5.599653262659064
5	5.546360087170829
6	5.513051852490682
7	5.2021657815798
8	4.758917485137064
9	4.514025945594523
10	4.292616020459272
11	0.4825271358456311

Average Head load: 351874.63636363635

Loadbalance Member

Peer	Messages
1	0.12299169311976259
2	0.12289494720990324
3	0.12282584298857514
4	0.1228120221443095
5	0.1228120221443095
6	0.12279820130004389
7	0.1225494261032627
8	0.1225356052589971
9	0.12243885934913774
10	0.12241121766060652
11	0.12239739681634088
12	0.12235593428354401
13	0.12232829259501278
14	0.12232829259501278
15	0.12225918837368469
16	0.12191366726704415
17	0.12189984642277851
18	0.1218860255785129
19	0.1218860255785129
20	0.12187220473424729
21	0.12178927966865355
22	0.12144375856201302
23	0.1214299377177474
24	0.12137465434068492
25	0.1213608334964193
26	0.1213608334964193
27	0.1213608334964193
28	0.12131937096362244
29	0.12131937096362244
30	0.12131937096362244
31	0.12130555011935681
32	0.12130555011935681
33	0.12127790843082557
34	0.12127790843082557

35	0.12127790843082557
36	0.12126408758655995
37	0.12126408758655995
38	0.12126408758655995
39	0.12026898679943522
40	0.11978525725013847
41	0.1195917654304198
42	0.11956412374188854
43	0.11948119867629481
44	0.1192738860123105
45	0.11926006516804488
46	0.11924624432377927
47	0.11923242347951366
48	0.11921860263524801
49	0.1192047817909824
50	0.1192047817909824
51	0.1192047817909824
52	0.11919096094671679
53	0.11919096094671679
54	0.11919096094671679
55	0.11917714010245115
56	0.11917714010245115
57	0.11909421503685744
58	0.11869341055315441
59	0.11869341055315441
60	0.1186795897088888
61	0.1186795897088888
62	0.1186795897088888
63	0.1186795897088888
64	0.1186243063318263
65	0.11861048548756069
66	0.11858284379902945
67	0.1185552021104982
68	0.11818203931532643
69	0.1181682184710608
70	0.11815439762679518
71	0.11814057678252957
72	0.11814057678252957
73	0.11812675593826395

74	0.11808529340546708
75	0.11807147256120147
76	0.11807147256120147
77	0.11804383087267022
78	0.1180300100284046
79	0.1180300100284046
80	0.11562518312618651
81	0.11518291610968663
82	0.11514145357688976
83	0.11500324513423357
84	0.1149617826014367
85	0.11487885753584297
86	0.1148373950030461
87	0.1148373950030461
88	0.11478211162598362
89	0.114768290781718
90	0.11475446993745239
91	0.11474064909318678
92	0.11469918656038991
93	0.11469918656038991
94	0.11468536571612427
95	0.11465772402759304
96	0.11463008233906179
97	0.11458861980626495
98	0.11456097811773369
99	0.11451951558493682
100	0.11450569474067121
101	0.11447805305213996
102	0.11396668181431198
103	0.11377318999459328
104	0.11375936915032767
105	0.1136488023962027
106	0.1135796981748746
107	0.11344148973221838
108	0.11344148973221838
109	0.11340002719942152
110	0.11337238551089027
111	0.11335856466662465
112	0.11331710213382779

113	0.11331710213382779
114	0.11330328128956217
115	0.11320653537970282
116	0.11320653537970282
117	0.11312361031410909
118	0.11309596862557784
119	0.11304068524851536
120	0.1129992227157185
121	0.11295776018292164
122	0.11295776018292164
123	0.11294393933865601
124	0.11290247680585914
125	0.11290247680585914
126	0.11288865596159353
127	0.11287483511732792
128	0.11287483511732792
129	0.11286101427306228
130	0.11286101427306228
131	0.11286101427306228
132	0.11286101427306228
133	0.11286101427306228
134	0.11286101427306228
135	0.11284719342879666
136	0.11284719342879666
137	0.11284719342879666
138	0.11283337258453105
139	0.11283337258453105
140	0.11283337258453105
141	0.11279191005173418
142	0.11277808920746857
143	0.11276426836320294
144	0.11276426836320294
145	0.11276426836320294
146	0.11276426836320294
147	0.11276426836320294
148	0.11275044751893731
149	0.11275044751893731
150	0.11275044751893731
151	0.1127366266746717

152	0.11272280583040609
153	0.11272280583040609
154	0.11270898498614045
155	0.11269516414187483
156	0.11268134329760922
157	0.11266752245334358
158	0.11266752245334358
159	0.11265370160907796
160	0.11265370160907796
161	0.11265370160907796
162	0.11265370160907796
163	0.11265370160907796
164	0.11265370160907796
165	0.11263988076481235
166	0.11262605992054674
167	0.1126122390762811
168	0.1126122390762811
169	0.11259841823201548
170	0.11257077654348425
171	0.11252931401068739
172	0.11252931401068739
173	0.11252931401068739
174	0.11252931401068739
175	0.11246020978935926
176	0.1123496430352343
177	0.1123496430352343
178	0.1122805388139062
179	0.11210086783845312
180	0.11203176361712502
181	0.11201794277285941
182	0.11197648024006254
183	0.1118935551744688
184	0.11187973433020319
185	0.11187973433020319
186	0.11185209264167194
187	0.11183827179740632
188	0.11183827179740632
189	0.11183827179740632
190	0.11183827179740632

191	0.11182445095314071
192	0.11179680926460946
193	0.11176916757607823
194	0.11175534673181259
195	0.11174152588754697
196	0.11172770504328136
197	0.1117000633547501
198	0.1117000633547501
199	0.11165860082195324
200	0.11164477997768762
201	0.11157567575635953
202	0.11154803406782828
203	0.11152039237929705
204	0.1114927506907658
205	0.11131307971531273
206	0.1112992588710471
207	0.1112992588710471
208	0.11128543802678148
209	0.11128543802678148
210	0.111230154649719
211	0.111230154649719
212	0.111230154649719
213	0.11121633380545339
214	0.11121633380545339
215	0.11121633380545339
216	0.11118869211692213
217	0.11117487127265652
218	0.1111610504283909
219	0.11114722958412526
220	0.11113340873985965
221	0.11109194620706278
222	0.11107812536279717
223	0.11102284198573469
224	0.11102284198573469
225	0.11102284198573469
226	0.11102284198573469
227	0.11099520029720343
228	0.11098137945293782
229	0.11098137945293782

230	0.1109675586086722
231	0.11093991692014095
232	0.11093991692014095
233	0.11093991692014095
234	0.11092609607587534
235	0.11092609607587534
236	0.11091227523160971
237	0.11091227523160971
238	0.11089845438734408
239	0.11087081269881285
240	0.11087081269881285
241	0.11087081269881285
242	0.11087081269881285
243	0.11087081269881285
244	0.11087081269881285
245	0.11087081269881285
246	0.11087081269881285
247	0.11087081269881285
248	0.11085699185454723
249	0.11085699185454723
250	0.11085699185454723
251	0.11080170847748473
252	0.11076024594468788
253	0.11073260425615664
254	0.11073260425615664
255	0.1107049625676254
256	0.1107049625676254
257	0.11067732087909415
258	0.11066350003482853
259	0.11066350003482853
260	0.11059439581350042
261	0.11059439581350042
262	0.1105805749692348
263	0.11056675412496918
264	0.11056675412496918
265	0.11056675412496918
266	0.11045618737084421
267	0.11033179977245361
268	0.110317978928188

269	0.110317978928188
270	0.11029033723965675
271	0.11026269555112551
272	0.11023505386259427
273	0.11015212879700055
274	0.11013830795273491
275	0.1101244871084693
276	0.11009684541993807
277	0.11009684541993807
278	0.11008302457567243
279	0.11008302457567243
280	0.11008302457567243
281	0.11006920373140681
282	0.1100553828871412
283	0.11004156204287557
284	0.11004156204287557
285	0.11004156204287557
286	0.11004156204287557
287	0.11004156204287557
288	0.11004156204287557
289	0.11002774119860995
290	0.11002774119860995
291	0.11001392035434433
292	0.11001392035434433
293	0.10995863697728185
294	0.10995863697728185
295	0.10994481613301624
296	0.10988953275595373
297	0.10987571191168811
298	0.1098618910674225
299	0.10984807022315689
300	0.10980660769036002
301	0.10976514515756315
302	0.10972368262476628
303	0.10962693671490693
304	0.10961311587064132
305	0.1094749074279851
306	0.10933669898532888
307	0.10930905729679766

308	0.10921231138693831
309	0.10918466969840705
310	0.10910174463281334
311	0.10906028210001648
312	0.10896353619015713
313	0.10896353619015713
314	0.10888061112456339
315	0.10886679028029778
316	0.10886679028029778
317	0.10885296943603216
318	0.1088115069032353
319	0.10875622352617281
320	0.10868711930484472
321	0.10868711930484472
322	0.10867329846057908
323	0.10864565677204785
324	0.10864565677204785
325	0.10863183592778221
326	0.1086180150835166
327	0.1086180150835166
328	0.10859037339498537
329	0.10859037339498537
330	0.10859037339498537
331	0.10859037339498537
332	0.10850744832939163
333	0.10850744832939163
334	0.10839688157526667
335	0.10836923988673541
336	0.10830013566540732
337	0.1082863148211417
338	0.10827249397687606
339	0.10820338975554797
340	0.1081619272227511
341	0.10806518131289175
342	0.10792697287023555
343	0.10784404780464181
344	0.1078302269603762
345	0.10781640611611057
346	0.10780258527184494

347	0.10778876442757933
348	0.10778876442757933
349	0.10777494358331371
350	0.10742942247667318
351	0.10741560163240754
352	0.10740178078814193
353	0.10740178078814193
354	0.10740178078814193
355	0.10740178078814193
356	0.10738795994387632
357	0.10734649741107945
358	0.1073188557225482
359	0.1073188557225482
360	0.10729121403401697
361	0.10729121403401697
362	0.10729121403401697
363	0.10729121403401697
364	0.10726357234548571
365	0.1072497515012201
366	0.10711154305856388
367	0.1070562596815014
368	0.10702861799297017
369	0.10700097630443892
370	0.10700097630443892
371	0.10694569292737643
372	0.1067936636404546
373	0.10658635097647029
374	0.10650342591087655
375	0.10621318818129853
376	0.10619936733703289
377	0.10614408395997041
378	0.10611644227143918
379	0.10611644227143918
380	0.10607497973864231
381	0.10604733805011106
382	0.10604733805011106
383	0.10603351720584545
384	0.10603351720584545
385	0.10603351720584545

386	0.10601969636157983
387	0.10601969636157983
388	0.10599205467304858
389	0.10592295045172048
390	0.10588148791892361
391	0.10588148791892361
392	0.10585384623039236
393	0.10585384623039236
394	0.10581238369759552
395	0.10579856285332988
396	0.10578474200906426
397	0.1057432794762674
398	0.1057432794762674
399	0.1057432794762674
400	0.1057432794762674
401	0.1057432794762674
402	0.1057432794762674
403	0.1057432794762674
404	0.1057432794762674
405	0.1057432794762674
406	0.1057432794762674
407	0.1057432794762674
408	0.1057432794762674
409	0.1057432794762674
410	0.1057432794762674
411	0.1057432794762674
412	0.1057432794762674
413	0.1057432794762674
414	0.1057432794762674
415	0.1057432794762674
416	0.1057432794762674
417	0.10571563778773617
418	0.10561889187787682
419	0.10545304174668936
420	0.10545304174668936
421	0.10541157921389249
422	0.10541157921389249
423	0.10537011668109564
424	0.10530101245976752

425	0.10530101245976752
426	0.1052871916155019
427	0.1052871916155019
428	0.1052871916155019
429	0.10517662486137694
430	0.10490020797606452
431	0.10469289531208018
432	0.10465143277928333
433	0.1046376119350177
434	0.10438883673823651
435	0.1043059116726428
436	0.1043059116726428
437	0.1043059116726428
438	0.10427826998411155
439	0.10426444913984594
440	0.10425062829558032
441	0.10423680745131468
442	0.10423680745131468
443	0.10419534491851784
444	0.10419534491851784
445	0.1041815240742522
446	0.10416770322998659
447	0.10415388238572097
448	0.10414006154145533
449	0.10414006154145533
450	0.10412624069718972
451	0.1041124198529241
452	0.10409859900865849
453	0.1039880322545335
454	0.10389128634467418
455	0.10382218212334605
456	0.10382218212334605
457	0.10373925705775235
458	0.10371161536922109
459	0.10369779452495548
460	0.10365633199215861
461	0.10365633199215861
462	0.10329699004125246
463	0.1031864232871275

464	0.10315878159859625
465	0.1030620356887369
466	0.1029929314674088
467	0.10295146893461193
468	0.10288236471328382
469	0.10288236471328382
470	0.1028685438690182
471	0.10277179795915885
472	0.10275797711489323
473	0.1027441562706276
474	0.1027441562706276
475	0.102730335426362
476	0.10268887289356513
477	0.1026750520492995
478	0.10263358951650264
479	0.10263358951650264
480	0.10263358951650264
481	0.10259212698370578
482	0.10259212698370578
483	0.10256448529517453
484	0.10256448529517453
485	0.10255066445090891
486	0.1025368436066433
487	0.10242627685251833
488	0.10239863516398708
489	0.10237099347545583
490	0.10235717263119021
491	0.10213603912294028
492	0.10210839743440903
493	0.10202547236881532
494	0.10202547236881532
495	0.10201165152454968
496	0.10197018899175281
497	0.1019563681474872
498	0.10192872645895597
499	0.10192872645895597
500	0.10165230957364353
501	0.1016246678851123
502	0.10159702619658105

503	0.10158320535231544
504	0.10158320535231544
505	0.10152792197525295
506	0.10151410113098731
507	0.10148645944245609
508	0.10148645944245609
509	0.10148645944245609
510	0.10144499690965922
511	0.10144499690965922
512	0.1014311760653936
513	0.10141735522112799
514	0.10140353437686235
515	0.10138971353259674
516	0.10136207184406548
517	0.10133443015553426
518	0.10132060931126864
519	0.10132060931126864
520	0.10125150508994052
521	0.10125150508994052
522	0.1012376842456749
523	0.10118240086861242
524	0.10107183411448746
525	0.10103037158169059
526	0.10100272989315934
527	0.10100272989315934
528	0.10100272989315934
529	0.10097508820462811
530	0.1009612673603625
531	0.10093362567183124
532	0.10093362567183124
533	0.10093362567183124
534	0.10090598398329999
535	0.10087834229476876
536	0.10082305891770628
537	0.10082305891770628
538	0.10082305891770628
539	0.10082305891770628
540	0.10080923807344064
541	0.10080923807344064

542	0.10080923807344064
543	0.10079541722917502
544	0.1007677755406438
545	0.10075395469637816
546	0.10074013385211254
547	0.10072631300784693
548	0.10072631300784693
549	0.10072631300784693
550	0.10071249216358132
551	0.10067102963078445
552	0.1006572087865188
553	0.10064338794225319
554	0.10062956709798758
555	0.10062956709798758
556	0.10060192540945632
557	0.10058810456519071
558	0.10058810456519071
559	0.1005742837209251
560	0.10056046287665948
561	0.10054664203239384
562	0.10054664203239384
563	0.10053282118812823
564	0.10051900034386262
565	0.10050517949959697
566	0.10050517949959697
567	0.10039461274547201
568	0.1003807919012064
569	0.10035315021267514
570	0.10033932936840953
571	0.10028404599134705
572	0.10027022514708143
573	0.10025640430281581
574	0.10025640430281581
575	0.10025640430281581
576	0.10022876261428457
577	0.10020112092575331
578	0.10020112092575331
579	0.10020112092575331
580	0.1001873000814877

581	0.10014583754869083
582	0.1001181958601596
583	0.10007673332736274
584	0.10007673332736274
585	0.10007673332736274
586	0.10004909163883148
587	0.10003527079456587
588	0.10002144995030024
589	0.10000762910603463
590	0.099993808261769
591	0.099993808261769
592	0.099993808261769
593	0.09997998741750339
594	0.09996616657323776
595	0.09995234572897214
596	0.09993852488470652
597	0.09993852488470652
598	0.0999247040404409
599	0.09989706235190965
600	0.09989706235190965
601	0.09988324150764404
602	0.09988324150764404
603	0.09988324150764404
604	0.0998555998191128
605	0.09984177897484717
606	0.09978649559778469
607	0.09978649559778469
608	0.09977267475351906
609	0.09977267475351906
610	0.09977267475351906
611	0.09974503306498782
612	0.09974503306498782
613	0.0997312122207222
614	0.09970357053219096
615	0.09967592884365971
616	0.0996621079993941
617	0.09963446631086284
618	0.09963446631086284
619	0.09962064546659723

620	0.09941333280261291
621	0.09941333280261291
622	0.09926130351569108
623	0.09923366182715983
624	0.09917837845009735
625	0.09916455760583173
626	0.09915073676156612
627	0.09913691591730048
628	0.09913691591730048
629	0.09910927422876925
630	0.09899870747464429
631	0.09892960325331616
632	0.09884667818772246
633	0.09877757396639433
634	0.09851497792534754
635	0.09840441117122257
636	0.09826620272856636
637	0.09825238188430074
638	0.09821091935150388
639	0.09821091935150388
640	0.0980865317531133
641	0.0978930399333946
642	0.09783775655633212
643	0.09782393571206648
644	0.09782393571206648
645	0.09774101064647277
646	0.09738166869556661
647	0.09736784785130098
648	0.09728492278570726
649	0.09720199772011354
650	0.09716053518731667
651	0.09713289349878543
652	0.09713289349878543
653	0.09704996843319169
654	0.09700850590039484
655	0.09684265576920738
656	0.09681501408067614
657	0.09677355154787927
658	0.09671826817081679

659	0.09664916394948868
660	0.0965938805724262
661	0.0965938805724262
662	0.09651095550683247
663	0.09649713466256685
664	0.09649713466256685
665	0.0964694929740356
666	0.09638656790844188
667	0.09635892621991064
668	0.09611015102312946
669	0.0960686884903326
670	0.09597194258047324
671	0.09595812173620763
672	0.09593048004767639
673	0.09588901751487952
674	0.09588901751487952
675	0.0958751966706139
676	0.09583373413781704
677	0.09583373413781704
678	0.09581991329355141
679	0.0958060924492858
680	0.0958060924492858
681	0.09577845076075456
682	0.09577845076075456
683	0.09577845076075456
684	0.09577845076075456
685	0.09504594601467664
686	0.09501830432614539
687	0.09493537926055166
688	0.0948248125064267
689	0.0947695291293642
690	0.09475570828509859
691	0.09470042490803611
692	0.09470042490803611
693	0.09452075393258302
694	0.09447929139978617
695	0.09445164971125491
696	0.09442400802272367
697	0.09439636633419243

698	0.0943825454899268
699	0.0943825454899268
700	0.09435490380139556
701	0.09435490380139556
702	0.09434108295712995
703	0.09432726211286432
704	0.09429962042433308
705	0.09428579958006747
706	0.0942443370472706
707	0.0942443370472706
708	0.09417523282594249
709	0.09414759113741125
710	0.09411994944888001
711	0.09409230776034877
712	0.09407848691608314
713	0.09407848691608314
714	0.0940508452275519
715	0.09403702438328629
716	0.09403702438328629
717	0.0939126367848957
718	0.09377442834223948
719	0.09376060749797387
720	0.09373296580944263
721	0.09373296580944263
722	0.093719144965177
723	0.093719144965177
724	0.09370532412091137
725	0.09369150327664576
726	0.09365004074384889
727	0.09363621989958328
728	0.09360857821105203
729	0.0935809365225208
730	0.09351183230119268
731	0.09347036976839582
732	0.09344272807986458
733	0.09344272807986458
734	0.09342890723559896
735	0.09340126554706771
736	0.09337362385853647

737	0.09335980301427085
738	0.09334598217000523
739	0.09333216132573961
740	0.09333216132573961
741	0.09331834048147399
742	0.09331834048147399
743	0.09331834048147399
744	0.09330451963720837
745	0.09330451963720837
746	0.09327687794867712
747	0.09327687794867712
748	0.09327687794867712
749	0.0932630571044115
750	0.0932630571044115
751	0.09324923626014588
752	0.09323541541588026
753	0.09323541541588026
754	0.09323541541588026
755	0.09323541541588026
756	0.09323541541588026
757	0.09320777372734902
758	0.0931939528830834
759	0.0931939528830834
760	0.09315249035028654
761	0.09315249035028654
762	0.09315249035028654
763	0.09312484866175529
764	0.09309720697322404
765	0.09300046106336471
766	0.09297281937483345
767	0.09294517768630221
768	0.09287607346497412
769	0.09284843177644288
770	0.09284843177644288
771	0.09280696924364601
772	0.09279314839938038
773	0.09276550671084914
774	0.09275168586658353
775	0.0927378650223179

776	0.09271022333378666
777	0.09268258164525542
778	0.09268258164525542
779	0.09264111911245855
780	0.09257201489113046
781	0.09248908982553672
782	0.09244762729273986
783	0.09241998560420862
784	0.09239234391567737
785	0.09230941885008365
786	0.09230941885008365
787	0.09230941885008365
788	0.09229559800581803
789	0.09226795631728678
790	0.09225413547302116
791	0.09224031462875554
792	0.09224031462875554
793	0.09222649378448992
794	0.0922126729402243
795	0.0922126729402243
796	0.0922126729402243
797	0.0922126729402243
798	0.09218503125169306
799	0.09218503125169306
800	0.09210210618609933
801	0.09206064365330247
802	0.09204682280903684
803	0.0920191811205056
804	0.09200536027623998
805	0.09183951014505252
806	0.09161837663680258
807	0.0915630932597401
808	0.09153545157120885
809	0.09153545157120885
810	0.09153545157120885
811	0.09146634734988075
812	0.09145252650561513
813	0.09143870566134951
814	0.09142488481708388

815	0.09138342228428702
816	0.0913696014400214
817	0.0913696014400214
818	0.09134195975149016
819	0.09127285553016205
820	0.09125903468589644
821	0.09124521384163081
822	0.09123139299736518
823	0.09120375130883394
824	0.09116228877603709
825	0.09114846793177146
826	0.09114846793177146
827	0.09114846793177146
828	0.09112082624324022
829	0.09106554286617774
830	0.09106554286617774
831	0.09105172202191211
832	0.09105172202191211
833	0.0910379011776465
834	0.09102408033338087
835	0.09102408033338087
836	0.09101025948911526
837	0.09096879695631839
838	0.09095497611205276
839	0.09094115526778715
840	0.09091351357925591
841	0.09089969273499028
842	0.09089969273499028
843	0.09089969273499028
844	0.09087205104645904
845	0.09087205104645904
846	0.09083058851366219
847	0.09076148429233408
848	0.09073384260380284
849	0.09069238007100597
850	0.09063709669394349
851	0.09056799247261538
852	0.09055417162834976
853	0.09054035078408414

854	0.09047124656275603
855	0.09047124656275603
856	0.09045742571849041
857	0.09045742571849041
858	0.09045742571849041
859	0.0903883214971623
860	0.09036067980863106
861	0.09030539643156858
862	0.09023629221024047
863	0.09007044207905301
864	0.0900566212347874
865	0.09000133785772492
866	0.08990459194786557
867	0.08982166688227185
868	0.08980784603800622
869	0.08980784603800622
870	0.08973874181667811
871	0.08973874181667811
872	0.08973874181667811
873	0.08973874181667811
874	0.08971110012814687
875	0.08968345843961563
876	0.08966963759535002
877	0.08964199590681876
878	0.08962817506255315
879	0.08962817506255315
880	0.08962817506255315
881	0.08961435421828752
882	0.0896005333740219
883	0.0896005333740219
884	0.0896005333740219
885	0.0896005333740219
886	0.08958671252975628
887	0.08954524999695943
888	0.08951760830842817
889	0.08951760830842817
890	0.08950378746416256
891	0.08950378746416256
892	0.08843958245570972

893	0.0877070777096318
894	0.08754122757844435
895	0.08752740673417873
896	0.08752740673417873
897	0.08733391491446003
898	0.0873200940701944
899	0.08723716900460068
900	0.08720952731606944
901	0.08716806478327258
902	0.08714042309474133
903	0.08712660225047572
904	0.08694693127502265
905	0.08691928958649141
906	0.08690546874222578
907	0.08689164789796015
908	0.0868501853651633
909	0.08683636452089767
910	0.08678108114383519
911	0.08675343945530395
912	0.08668433523397584
913	0.08668433523397584
914	0.08650466425852277
915	0.08644938088146027
916	0.08622824737321033
917	0.08617296399614785
918	0.08615914315188222
919	0.08603475555349163
920	0.08600711386496039
921	0.08599329302069478
922	0.08597947217642915
923	0.08597947217642915
924	0.08596565133216354
925	0.08596565133216354
926	0.08595183048789791
927	0.0859380096436323
928	0.08592418879936667
929	0.08592418879936667
930	0.08592418879936667
931	0.08591036795510106

932	0.08589654711083543
933	0.08589654711083543
934	0.0858827262665698
935	0.0858827262665698
936	0.08586890542230419
937	0.08578598035671046
938	0.08571687613538236
939	0.08570305529111673
940	0.08564777191405425
941	0.085620130225523
942	0.08559248853699176
943	0.08559248853699176
944	0.08557866769272614
945	0.08556484684846052
946	0.0855510260041949
947	0.08552338431566366
948	0.08550956347139804
949	0.08550956347139804
950	0.08549574262713242
951	0.08546810093860117
952	0.08544045925006993
953	0.0854266384058043
954	0.0854266384058043
955	0.08541281756153869
956	0.08532989249594497
957	0.08530225080741372
958	0.08473559619252326
959	0.08473559619252326
960	0.08473559619252326
961	0.08472177534825763
962	0.08468031281546078
963	0.08440389593014835
964	0.08437625424161711
965	0.08436243339735149
966	0.08434861255308587
967	0.08429332917602338
968	0.08421040411042965
969	0.08419658326616404
970	0.08415512073336717

971	0.023481614407290328
972	0.023440151874493467
973	0.023398689341696602
974	0.02337104765316536
975	0.02326048089904039
976	0.02326048089904039
977	0.023149914144915424
978	0.0231360933006498
979	0.02309463076785294
980	0.02308080992358732
981	0.02290113894813424
982	0.022859676415337378
983	0.022845855571071754
984	0.022832034726806134
985	0.022818213882540513
986	0.02277675134974365
987	0.022762930505478028
988	0.022735288816946787
989	0.022721467972681167

Average Member load: 7315.923154701719

Diameter: 0

APPENDIX C

Publication Records

Chapter 3 is partially derived from the following publications:

- **Hassan, M.I.** and A. Abdullah (2010). Semantic-Based Grid Resource Discovery Systems: A literature review and taxonomy. Proceedings The 4th International Symposium on Information Technology (ITSim 2010), IEEE Computer Society, vol 3, pp 1286-1297, Kuala Lumpur - Malaysia.

Chapter 4 is partially derived from the following publications:

- **Hassan, M.I.** and A. Abdullah (2009). "A Semantic Description and Registration Framework for Large Grid Resource Discovery Systems." Journal Computer Science Letters, vol 1, issue 1.
- **Hassan, M.I.** and A. Abdullah (2009). Semantic-Based Scalable Decentralized Grid Resource Discovery. Proceedings The 2009 International Conference on e-Technology (e-Tech 2009), pp 3316-3324, Singapore.

Chapter 5 is partially derived from the following publications:

- **Hassan, M.I.** and A. Abdullah (2011). "A New Resource Discovery Framwork." The International Arab Journal of Information Technology (IAJIT) , vol 8 , issue1, pp 20-28.
- **Hassan, M.I.** and A. Abdullah (2009). Semantic-Based Scalable Decentralized Resource Discovery. Proceedings National Postgraduate Conference 2009, NO. 194, ID CIS042, Tronoh - Malaysia.

- **Hassan, M.I.** and A. Abdullah (2008). Self-organizing Grid resource discovery. Proceedings The International Symposium on Information Technology (ITSim 2008). IEEE Xplore, vol 1, pp 1-4, Kuala Lumpur-Malaysia,
- **Hassan, M.I.** and A. Abdullah (2008). Scalable Self-Organizing Model for Grid Resource Discovery. International Conference on Network Applications, Protocols and Services 2008, No.23, Kedah Darul Aman-Malaysia.