Graduate School ETD Form 9
(Revised 12/07)

PURDUE UNIVERSITY
GRADUATE SCHOOL
Thesis/Dissertation Acceptance

This is to certify that the thesis/dissertation prepared

By Ketaki Abhay Pradhan

Entitled MDE-URDS-A Mobile Device Enabled Service Discovery System.

For the degree of Master of Science

Is approved by the final examining committee:

Dr. Rajeev R. Raje
Chair

Dr. Mihran Tuceryan

Dr. James Hill

To the best of my knowledge and as understood by the student in the *Research Integrity and Copyright Disclaimer (Graduate School Form 20)*, this thesis/dissertation adheres to the provisions of Purdue University's "Policy on Integrity in Research" and the use of copyrighted material.

Approved by Major Professor(s): Dr. Rajeev R. Raje

Approved by: Dr. Shiaofen Fang 12/10/2010
Head of the Graduate Program Date

Graduate School Form 20
(Revised 9/10)

PURDUE UNIVERSITY
GRADUATE SCHOOL

Research Integrity and Copyright Disclaimer

Title of Thesis/Dissertation:

MDE-URDS-A MOBILE DEVICE ENABLED SERVICE DISCOVERY SYSTEM.

For the degree of Master of Science

I certify that in the preparation of this thesis, I have observed the provisions of *Purdue University Executive Memorandum No. C-22,* September 6, 1991, *Policy on Integrity in Research.**

Further, I certify that this work is free of plagiarism and all materials appearing in this thesis/dissertation have been properly quoted and attributed.

I certify that all copyrighted material incorporated into this thesis/dissertation is in compliance with the United States' copyright law and that I have received written permission from the copyright owners for my use of their work, which is beyond the scope of the law. I agree to indemnify and save harmless Purdue University from any and all claims that may be asserted or that may arise from any copyright violation.

Ketaki Abhay Pradhan

Printed Name and Signature of Candidate

12/10/2010

Date (month/day/year)

*Located at http://www.purdue.edu/policies/pages/teach_res_outreach/c_22.html

MDE-URDS-A MOBILE DEVICE ENABLED SERVICE DISCOVERY SYSTEM

A Thesis

Submitted to the Faculty

of

Purdue University

by

Ketaki A. Pradhan

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science

May 2011

Purdue University

Indianapolis, Indiana

To,

Aai and Baba

ACKNOWLEDGMENTS

Being a Graduate student of the Department of Computer and Information Science at IUPUI is one of the most memorable experiences of my life. It has taught me many things in life and I will always cherish my memories of being a part of this Institution. I always thank God for giving me this wonderful opportunity. In addition, there are a couple of people I would like to thank who have been very supportive throughout my Graduate studies.

First and foremost of I would like to thank my advisor, Dr. Rajeev Raje for his guidance and encouragement throughout my Thesis and Graduate studies. I cannot thank him enough for being so patient and supportive. I would also like to thank Dr. Mihran Tuceryan and Dr. James Hill for agreeing to be as a part of my Thesis Committee and providing their valuable feedback on the Thesis revisions.

I would especially like to thank my colleague and good friend Lahiru for his support and being there for me whenever I needed help. My special thanks to Alfredo who helped me a great deal in my initial days of research.

Thank you to all my friends and well wishers for their good wishes and support. And finally, I would like to thank my grandmother, my parents and my brother Kunal for their unconditional love and support.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

ABSTRACT

Pradhan, Ketaki A. M.S., Purdue University, May 2011. MDE-URDS-A Mobile Device Enabled Service Discovery System. Major Professor: Rajeev Raje.

Component-Based Software Development (CSBD) has gained widespread importance in recent times, due to its wide-scale applicability in software development. System developers can now pick and choose from the pre-existing components to suit their requirements in order to build their system. For the purpose of developing a quality-aware system, finding the suitable components offering services is an essential and critical step. Hence, Service Discovery is an important step in the development of systems composed from already existing quality-aware software services. Currently, there is a plethora of new-age devices, such as PDAs, and cell phones that automate daily activities and provide a pervasive connectivity to users. The special characteristics of these devices (e.g., mobility, heterogeneity) make them as attractive choices to host services. Hence, they need to be considered and integrated in the service discovery process. However, due to their limitations of battery life, intermittent connectivity and processing capabilities this task is not a simple one.

This research addresses this challenge of including resource constrained devices by enhancing the UniFrame Resource Discovery System (URDS) architecture. This enhanced architecture is called Mobile Device Enabled Service Discovery System (MDE-URDS). The experimental validation of the MDE-URDS suggests that it is a scalable and quality-aware system, handling the limitations of mobile devices using existing and well established algorithms and protocols such as Mobile IP.

CHAPTER 1. INTRODUCTION

This thesis describes the architecture of Mobile Device Enabled – UniFrame Resource Discovery System (MDE-URDS), a service discovery architecture using mobile devices. This chapter explains the motivation behind the thesis and an introduction to the overall approach taken.

1.1. Problem Statement and Motivation

Component-Based Software Development (CSBD) has gained widespread importance in recent times, due to its wide-scale applicability in software development. Such techniques have made advancements in software development in modern times to a large extent. From taking years to finalize software, now software can be developed and deployed in a matter of a few days. Component Based Software Development [1] and Generative Programming [2] provide the luxury of the use of already existing components in the software development process. System developers can now pick and choose from the pre-existing components to suit their requirements in order to build their own system. It is then the task of developing a system using these found components with the specific and user-desired quality of service (QoS) requirements. Components usually belong to a specific domain and offer services, such as tracking components offer tracking services or healthcare components offering health-related services. As a result, the terms components and services are used interchangeably in this thesis. For the purpose of developing a system, finding the suitable components offering services is an essential and critical step. Hence, Service Discovery is an important step in the development of systems composed from already existing quality-aware software services.

A lot of emphasis is laid on finding the right services to meet the requirements of the system under development. This process is iterative and the developer may look for newer components to meet those requirements. Service Discovery, if not appropriately

done, may delay the overall process of software development and may even as well lead to incorrect development of the required system. Researchers in the past have proposed several discovery architectures, such as Jini [3], UPnP [4], UDDI [5], and Service Location Protocol [6]. These, and similar efforts, are referred to as first generation service discovery architectures in a comprehensive report from the researchers at the National Institute of Standards and Technology (NIST). A comprehensive list of related work in this field can be seen in Chapter 2.

Currently, there is a plethora of new-age devices, such as PDAs, and cell phones that automate daily activities and provide a pervasive connectivity to users. The special characteristics of these devices (e.g., mobility, heterogeneity) make them as attractive choices to host services and hence, they need to be considered and integrated in the service discovery process. Many context-aware applications, such as distributed tracking, require the inclusion of such mobile devices that host relevant services. Also, there are many real-time applications such as disaster management, traffic monitoring, benefiting by the use of mobile devices and services, giving rise to many service discovery architectures. Hence, the inclusion of these mobile devices into the discovery framework enhances the discovery process by making it suitable to the category of context-aware applications. The first generation of discovery systems however, did not consider including the mobile devices into their service discovery architecture. The problem with these devices is the additional level of support for wireless technology, their intermittent connectivity, battery life, limited memory capability and additional hardware support that is needed for correct functionality.

NIST in their survey [7] of first generation of discovery systems have identified this particular challenge of resource discovery with mobile devices. This research therefore addresses the challenge of including resource constrained devices by handling a subset of the challenges (such as Intermittent Connectivity, Mobility, memory and processing capabilities) by enhancing the UniFrame Resource Discovery System (URDS) architecture. This enhanced architecture is called Mobile Device Enabled Service Discovery System (MDE-URDS).

The MDE-URDS proposes solutions to handle the limitations of the resource-constrained devices using algorithms such as Mobile IP [8], Ad-hoc On Demand Vector routing [9], Optimized Link State Routing [10] and the use of near-by proxies in the discovery process. The quality of the results obtained from the discovery process is improved using the multi-level matching semantics proposed by [11][12].

1.2. Objectives of the Thesis

- To design an architecture, called MDE-URDS, for service discovery that incorporates mobile devices.
- To empirically validate the proposed approach by performing extensive experimentation with the prototype of MDE-URDS.
- To compare the performance of MDE-URDS with an existing prototype of URDS that does not include mobile devices.

1.3. Contributions of the Thesis

- This Thesis has tried to address the problem of incorporating mobile devices into service discovery architecture.
- The limitations of limited connectivity, limited battery life, memory capacity and other constraints have been addressed using different algorithms such as Mobile IP and changes to the existing URDS architecture.
- A comparative study of the discovery process with and without the inclusion of mobile devices with respect to the discovery performance has been studied and suggestions for the enhancement have been proposed.

1.4. Organization of the Thesis

The Thesis is organized as follows: Chapter 2 presents the related efforts; Chapter 3 explains the architecture of the proposed MDE-URDS and associated algorithms. Chapter 4 describes the experimentation carried out using the prototype of MDE- URDS. Chapter 5 presents the conclusions of this study and future work for the enhancement of MDE-URDS.

CHAPTER 2. RELATED WORK

In Chapter 1, the motivation behind service discovery architectures in recent times and the need of including mobile devices in these discovery architectures was explained in detail. Service oriented architectures [13][14] have led to the flourishing of Discovery Services (DS). Starting with the earliest architectures, there has been a significant improvement not only in the technology but also in different aspects of discovery, from simple text-based search to multi-level matching to semantic and ontology based approaches. This chapter provides a survey of the existing service discovery approaches for mobile and resource constrained devices and identifies their strengths and weaknesses.

A few prominent first generation DS are: Jini [3], UPnP [4], SLP [6], CORBA Trader Service [15] and UDDI [5]. The NIST report [7] classifies DS into two main groups:

a. **Lookup Services:** These lookup services are registry-based discovery services, where a service provider typically registers his services with a central registry. The client consults the registry to search for services that he is interested in. This group includes Jini, UPnP, CORBA Trader Service, and UDDI.
b. **Discovery Services:** This category includes specially designed architectures for resource discovery, such as the Service Location Protocol (SLP) which provides the service discovery with the help of the User Agents, Service Agents, and Directory Agents forming three-party architecture.

However, none of these first generation discovery systems consider the need of including resource constrained devices into their discovery setup. This limitation of first generation of discovery systems is also highlighted by NIST report [7]. Thereafter, several architectures have been proposed by researchers in the past to tackle this problem

of incorporating resource constrained devices into the service discovery setup. A few prominent of these architectures are described below.

The Framework for Robust and Resource-aware Discovery (FRODO) [16] is a service discovery architecture mainly designed for mobile and resource constrained devices. In FRODO, the resource constrained devices are grouped into different classes depending on their memory and processing capabilities. The devices are allotted different tasks based on the classes they belong to. Although FRODO is one of the first solutions proposed, it does not account for limitations such as the intermittent connectivity or frequent migration of the mobile devices from their home domain. This has been suitably handled by incorporating the principles of Mobile IP protocol in the proposed MDE-URDS architecture. Similar to the FRODO approach, the devices in MDE-URDS are also classified based on their processing speed and memory and accordingly tasks are delegated to them explained in Section 3.1 of Chapter 3.

M-URDS [17], an earlier work from the UniFrame group, is also an extension of URDS that deals with the mobility of the components by incorporating mobile agents into URDS. The main task of these mobile agents is to discover new components. However, M-URDS did not employ the use of mobile devices as a part of the service discovery framework. Another similar agent-based architecture is presented in [18] which introduces the collaborative searches wherein the agents gather information from different sources and present a more efficient searching method. They make use of the collaborative search wherein the agents share resources and also perform periodic update of their resources. The query propagation and delegation techniques in the MDE-URDS use similar principles as these agent-based approaches, explained in Section 3.2 of Chapter 3.

[19] and [20] make use of offloading and surrogates for the purpose of including mobile devices by offloading complex tasks to the surrogate. This helps in achieving both tasks of using mobile devices as well as providing support for complex tasks that need more processing power and speed, that cannot be handled by them. [19] also provides elaborate experimentation results for offloading under specific conditions of memory and processing limitations. This similar idea has been used in the MDE-URDS in using the

proxy for providing different matching capabilities and offloading to a nearby proxy in case of heavy load or multi-level matching approaches, explained in Section 3.2.1 of Chapter 3.

[21] defines a middleware framework called MARKS that is designed for mobile devices and it provides resource discovery, knowledge usability and self-healing. Resource discovery in MARKS is done using the cluster based hash algorithm wherein the devices perform a peer-to-peer discovery of services. However, the approach chosen is not scalable and efficient for resource discovery as the main focus of the MARKS project is dealing with Knowledge usability and self-healing aspects of mobile devices. [22] defines a secure service discovery architecture wherein the security and privacy of the clients is considered as an integral part of the discovery process. The user can browse the service portal from the Web and get access to the services. These services are prevented from attacks by malicious users by the use of the Direct Anonymous Attestation (DAA) scheme along with the Diffie-Hellman key exchange algorithm. Currently, MDE-URDS does not consider security and self-healing in its architecture.

The Daidalos project [23] designs, develops and validates a blueprint beyond the 3G framework and supports secure and pervasive services built on heterogeneous network and service infrastructures for the mobile user. They have used context-aware services in their implementation. Their main focus is on providing services rather than the mobile devices and therefore they have not considered tackling mobility limitations of the devices into their framework.

[24] is another work introducing a new discovery protocol that works on the push-based and pull-based resource information dissemination that can handle the dynamicity and the quality-of-service requirements of the software services. This framework is designed to support survivability and information assurance by migration of components to safe locations in case of any emergencies. Some of their principles can be incorporated into the MDE-URDS as a future work.

Researchers have done a lot of work in Context-Aware Service Discovery with mobile devices in the past. These context-aware architectures incorporate contextual information include service location, QoS parameters as a part of the service description.

They have been separated into a special group of context-aware discovery services. Context-aware discovery approaches are similar to ontology-based architectures wherein the former case the context is represented in an ontology. Examples of these include architectures such as [25][26][27]. In [25], contextual information plays a major role in selecting entities for discovery participation. In [25], the researchers have included contextual information at different levels of Infrastructure, Application and Component Discovery and provide its evaluation. For these service discovery layers, the contextual information is created which is used according to the matching requirements. Typically, a contextual schema has a user profile, personal user profile and the service profile. Depending on the service level, this contextual information is used, e.g., a device profile is used in infrastructure service discovery.

Another work [28] uses the contextual information mainly with respect to the QoS parameters of the services and only those services that match the required QoS are then selected as the most relevant results. They create application profiles that are used in matching of the user requirements. Matching is thus with respect to only the QoS aspects and they claim that it does not add any overhead on the device. In [27], a middleware named AIDAS (Adaptable Intelligent Discovery of context-Aware Services) tackles the contextual information and provides the semantic-based matchmaking between the available and requested services. This also matches with MDE-URDS's idea of providing multi-level matching, however it only deals with two-levels of match of syntax and semantics, whereas MDE-URDS is also similar to them as it provides the type and QoS match, but is capable of providing all five levels of matching of type, syntax, semantics, synchronization and QoS. AIDAS also provides support for the management of ontology repositories for the mobile devices.

Context-aware architectures that include location as the main contextual information include [26] and [29]. The location-based architecture [26] tackles the mobility of the devices with the help of location-based activity using scope and provides a secure access to the devices. They use location based information using IDs as used by people in real world and authorization information such as administrative policies defined for certain scopes based on standards such as Geopriv [30], in addition to accepted public

and private-key encryption. [29] provides a secure service architecture, Splendor, that allows an access to public services by location while maintaining privacy and security of the system. Splendor makes the use of tags that emit location information for keeping track of the entities in the system. Security being the main concern of this architecture, they use different authentication and communication mechanisms such as public and symmetric key encryption and also public key certificates for two-party authentication among services, clients and the servers. As mobile devices cannot handle the load of the public encryption techniques effectively, they include proxies for handling of other tasks of the mobile services, while the devices handle the authentication and authorization. MDE-URDS does not make use of the location aware service discovery approach, but with the help of Mobile IP principles, it can keep a track of the mobile devices on which the components are deployed. However, the current security mechanisms in the MDE-URDS are not very complex and using some of these techniques from above mentioned location-aware systems, the MDE-URDS can be further improved as a part of the future work.

Another example from the location-aware discovery is [31]. This architecture describes ways of handling the discovery of resources that are constantly moving. They use different location-aware metrics and vectors to track the moving resources and provide with the current resources in a particular location. However, their system is less scalable and the performance degrades due to frequent migration of resources. MDE-URDS makes use of the Mobile IP principles (explained in Section 3.2.4 of Chapter 3) and as a result, the maintenance of location-aware metrics and vectors is eliminated and as a result the scalability of the system is better. However, frequent migration of the resources is also a concern in the architecture as it causes an increase in the response time.

Ontology-based matching and semantic-based discovery are also popular fields in service discovery where the user requirements are matched with the services using a described ontology tree. [32] defines ontology as a body of formally represented knowledge is based on a conceptualization: the objects, concepts, and other entities that are assumed to exist in some area of interest and the relationships that hold among them.

A conceptualization is an abstract, simplified view of the world that we wish to represent for some purpose. Ontology-based approaches make use of ontology for description of services and the search takes place by traveling through this ontology tree. This ontology is helpful for performing searches that give rise to better quality of results. [33] enhances the Bluetooth Service discovery protocol by the inclusion of a semantic layer and discovering services in a m-commerce scenario. The matching is performed considering ontology-based descriptions.

Similar work is presented in [34] and [35] using ontology for the service discovery purposes. [36][37] also describe semantic service discovery on mobile devices using an ontology-based approach. [38] describes a global architecture for service discovery, called GloServ, for local and wide area networks. GloServ uses the Web Ontology Language (OWL) for automated registration and querying of services. They create a hierarchy for the services based on ontology and the query propagation takes place through this hierarchy. Ontology-based search can be used in the MDE-URDS by including the ontology information into its knowledgebase, that contains matching and system generation rules mentioned in addition. However, the maintenance of ontology-based information is difficult as it needs more memory requirement for saving this information. Also for processing as there is a need of traversing through the ontology tree for service matching and may lead to an unacceptable overhead in the MDE-URDS.

VOLARE [39] is a middleware that provides adaptive interfaces that match the user requirements with the web services available and adapts the requests depending on the surroundings (example traffic scenario). In the MDE-URDS, such a middleware framework can be useful for making the services adaptive and saving resources when the demand goes down or in low power mode.

Mobile Ad-Hoc Networks (MANETS) have been popular in the network domain and there are a lot of similarities between them and the mobile devices due to the similar characteristics they share such as limited power, mobility, and wireless connectivity. They communicate and operate in a close setup of devices and through the different routing protocols such as Ad-hoc On Demand Vector routing [9], Optimized Link State Routing [10] and Dynamic Source Routing [40]. [41] proposes a scalable service

MANET discovery by reducing traffic, distributed directory mechanisms and providing caching mechanisms. Local discovery is followed by the collaboration of the results leading to global service discovery. [42] also proposes a scalable service discovery protocol for MANET called CARD (Contact- based Architecture for Resource Discovery) which improves the scalability of the network by maintaining contacts or group information instead of just neighbor information, improving query routing mechanisms. [43] proposes a bandwidth preserving discovery approach for MANETS by the use of one-dimensional structures called tracks instead of zones. [44] discusses a new programming language called SpatialViews for resource constrained devices and ad-hoc networks that is used for specification of virtual networks with nodes providing services. This model uses best-effort semantics and guarantee discovery of the nodes with user-defined time constraints and quality. The MDE-URDS uses some of the principles of MANET routing protocols of maintaining a neighbor list, and on-demand routing of queries to neighbors, for its collaborative query processing.

From above mentioned related works, it is seen that all these architectures typically focus on one specific aspect of the mobile devices. The MDE-URDS, on the other hand, tackles more than one limitation (mobility, intermittent connectivity, heterogeneity, and processing capabilities) by using widely accepted principles and protocols, and highlighting the openness and heterogeneity that are inherent in a distributed system. Chapter 3 describes the architecture of MDE-URDS in detail.

CHAPTER 3. MDE-URDS DESIGN

Chapter 2 described different service discovery architectures that included the mobile devices into their frameworks. In this chapter, the architecture of the MDE-URDS is explained along with the associated algorithms. The MDE-URDS is designed to make the process of discovery a seamless approach so that any device (stationary, mobile, or resource-constrained) can be a part of the discovery framework.

Mobile devices are prevalent everywhere now and to include them in the discovery architecture requires handling the main limitations. Mobile devices have certain limitations compared to a resourceful device. They mainly include: **Mobility**, **Intermittent Connectivity**, **Device Limitations (Memory and Processor capabilities), Battery Life and Heterogeneity.** In this thesis, the first three limitations of mobile devices have been addressed while proposing the architecture of the MDE-URDS.

3.1. MDE-URDS Architecture

The architecture of the MDE-URDS is an enhancement of the URDS [45] architecture. The URDS, as indicated in the previous chapter, was designed mainly to incorporate heterogeneity of entities and to have a proactive and hierarchical service discovery system. The architecture along with the algorithms is explained in detail in [45]. It is described briefly below. Figure 3.1 shows the architecture of the URDS from [45].

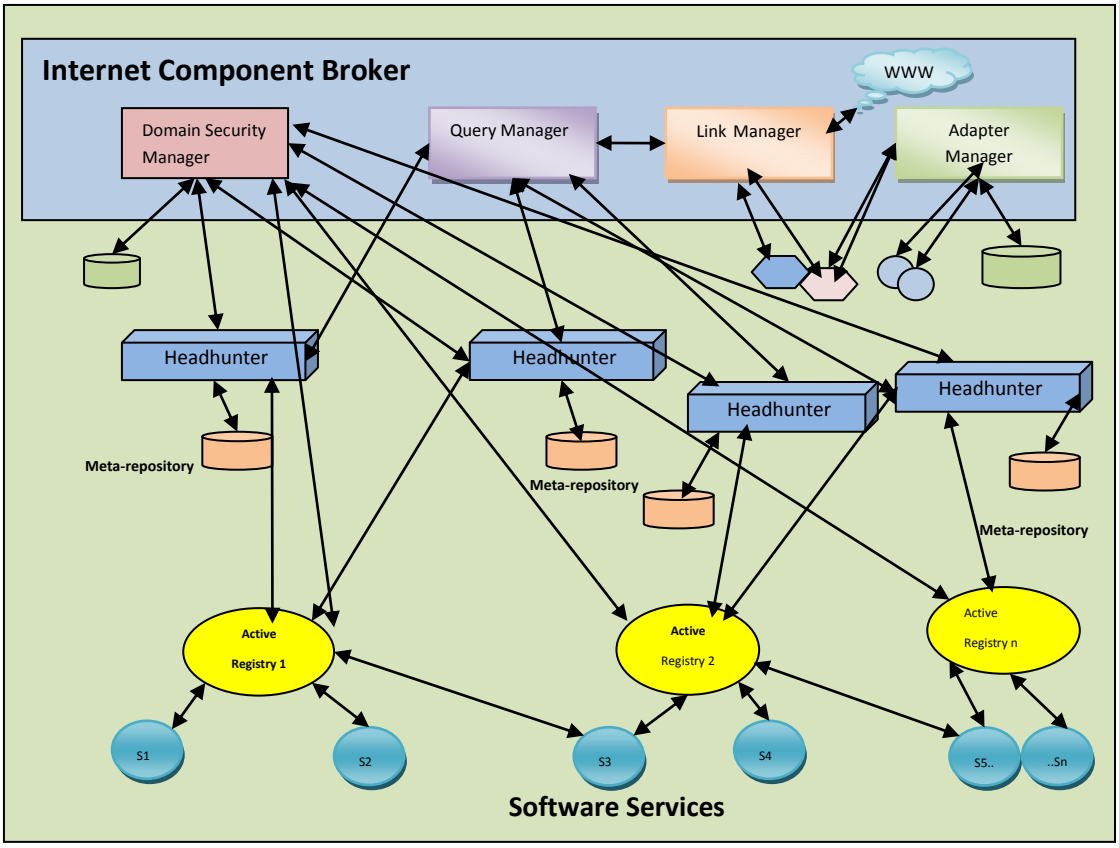


Figure 3.1 URDS Architecture

The main entities of URDS include:

1. **<u>The Internet Component Broker (ICB)</u>:** It is the main building block of URDS. It is similar to the Object Request Broker in CORBA and contains:
   a. **<u>Domain Security Manager (DSM)</u>:** It is responsible for maintaining a list of all registered entities of the URDS along with their authentication credentials. All the entities in the URDS need to authenticate themselves with the DSM before they can perform any task (such as joining the system, and getting information about new entities).
   b. **<u>Query Manager (QM)</u>:** This is responsible for querying the headhunters (described below) for the incoming queries. It authenticates with the DSM and gets

reference to all registered headhunters to whom it queries for the client requested services and returns results, if any, to the clients.

c. **Link Manager (LM):** The LM is responsible for linking many ICBs together to form a federation of discovery services.

d. **Adapter Manager (AM):** The AM deals with the heterogeneity of the entities by providing necessary adaptive bridges.

2. **Active Registries (AR):** These are proactive registries are constantly seeking new services and register these services with themselves. There can be many such registries present in the URDS and they can also be heterogeneous.
3. **Headhunter (HH):** The HH is one of the most important entities of the URDS. It performs the matching process of the query. The HHs are constantly looking for new services across the ARs and register these services with their own local meta-repository.
4. **Clients:** The clients of the URDS pass the user queries to the QM which then sends back the results from the headhunters suitable for this query.
5. **Services:** These user-defined services, heterogeneous in nature, register themselves with the ARs and are invoked by the clients to perform certain functions.

However, the URDS only considered the deployment of the entities on resourceful devices. Some preliminary experiments on mobile devices were performed [44] with the URDS, however, these devices were not considered as first-class entities in the URDS. With an increase in the number of mobile devices used today, it is difficult to find an application which does not use such devices. In an attempt to address this issue, the architecture of the MDE-URDS is proposed.

For the URDS to include mobile devices some architectural changes are needed. First, similar to the FRODO [16] approach, the resources are divided into two categories: resourceful and resource-constrained. A mapping scheme is identified that is based on the functionality of each URDS entity and associated empirical evaluations necessary for placing the URDS entities into these two types of resources. Table 3.1 indicates the result of the mapping process.

Table 3.1 Mapping of URDS Entities on the different categories of devices

URDS Entity	Resourceful device	Resource-constrained device
Domain Security Manager	Yes	No
Query Manager	Yes	Yes
Link Manager	Yes	No
Adapter Manager	Yes	No
Headhunter	Yes	Yes
Active Registries	Yes	Yes
Clients	Yes	Yes

In this thesis, among the URDS entities, mainly the mobility of the headhunters and clients is addressed. The mapping provided the basis for porting the selected entities of the URDS onto the mobile devices. As decided from the mapping, the headhunters and clients were deployed on the mobile devices.

Based on the above mapping schemes, the URDS architecture was modified to include the mobile headhunters and clients. Even, services on mobile devices could be a part of the architecture; however, as the services need to be active all the time, it was decided for them to be deployed on resourceful devices. The modified architecture is shown in Figure 3.2 and serves as the starting point for developing the comprehensive architecture of the MDE-URDS.

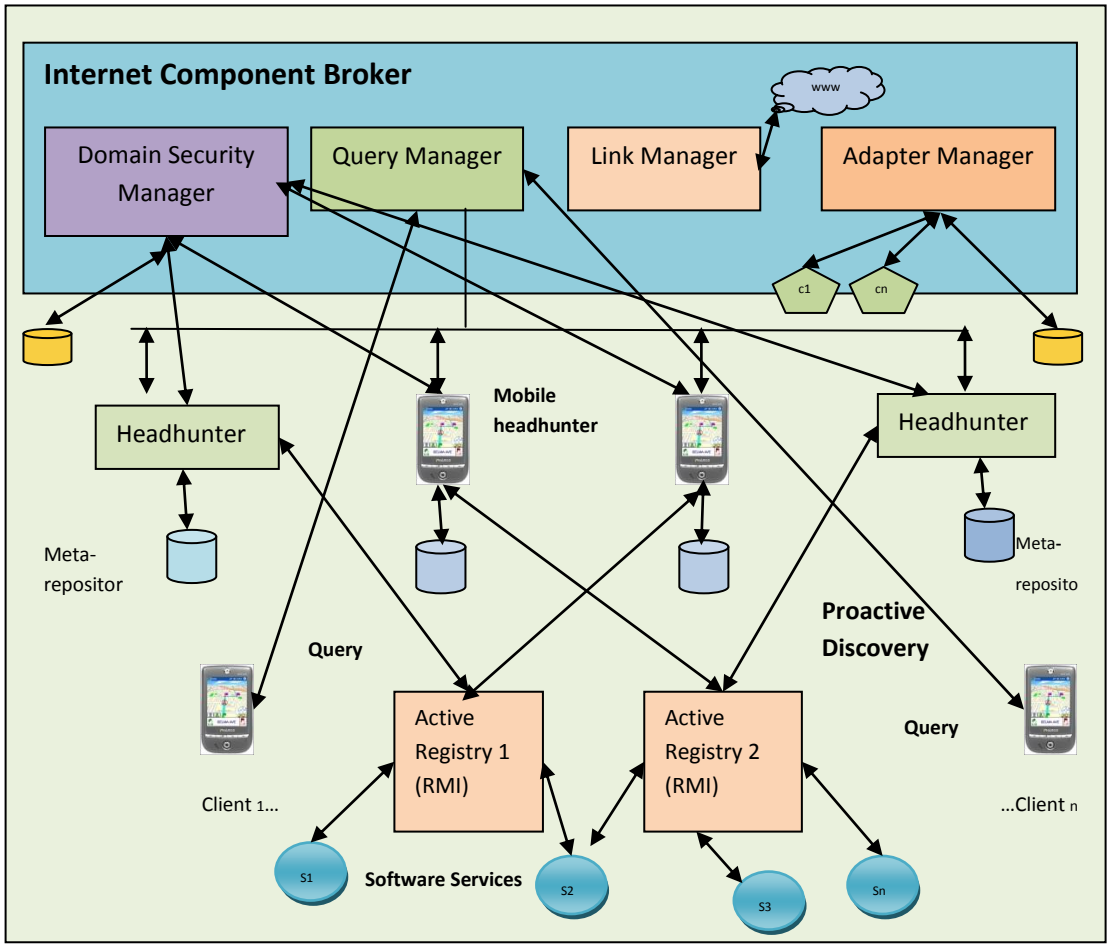


Figure 3.2 Basic MDE-URDS Architecture

The process of discovery in the MDE-URDS is similar to the URDS, however, as mobile devices are considered; the underlying algorithms vary to some extent. The algorithms for various entities in the basic MDE-URDS architecture (Figure 3.2) are discussed below.

3.1.1. Headhunter

The following are the algorithms for the headhunter start, migration and query execution functions.

a. HH Start

START.

//HH REGISTERS AS A RMI ENTITY WITH A SPECIFIC NAME WITH THE RMI ENTITY IN THE DEVICE (DESKTOP OR PDA).

Naming.bind (name, IP);

//JINI LOOKUP AND GET ACTIVE REGISTRIES CONTACT //FROM JINI.

lookup = new LookupLocator("jini://jinihost");

Jiniregistrar = lookup.getRegistrar( );

componentListObject = Jiniregistrar.lookup(template).

//UPDATE ITS META-REPOSITORY BY CONTACTING THE //ACTIVE REGISTRIES.

contractList = componentListObject.getContractList();

END.

b. HH Query Execution

START.

//HH RECEIVES QUERY

hh.receive (query, timestamp).

//EXECUTE THE QUERY

results = hh.executeQuery( ) ;

//PERFORM TYPE MATCHING AS

if(query.componentname =contract.name)

results.add(contract);

if (results not NULL)

return results.

```
        else
            return "No Components Found"
    END.
```

3.1.2. Query Manager

The following Section explains algorithms for various QM functions.

a. Query Manager Start

```
START.
//REGISTER WITH RMI REGISTRY
    Naming.bind("QueryManager",this);
//SET THE QUERY QUEUES AND HHLISTS.
    setqueryList( );
    setHHlist( );
//CONTACT DSM TO OBTAIN ALL REGISTERED HH REF LIST FROM
//DSM.
    dsm. getDSMHHRef( );
END.
```

b. Query Processing

```
START.
//PLACE NEW QUERY IN THE QUERY QUEUE.
    recieveQuery(query,timestamp);
    qList[curr]=query;
//SELECT THE NEW QUERY FROM THE QUEUE, SPAWN A NEW
//THREAD.
HH SELECT: SEE HH-SELECT ALGORITHM
    RunQuery run=new RunQuery( );
    run.setParameters(hhRef,timestamp);
    run.start( );
```

//Thread processing:

results= hh.executeQuery ();

//send results to QueryManager

receiveResults(results,ID);

//Calculate time taken

time taken = System.currentTimeInMillis( )- qtimestamp;

return results.

END.

c. Query Manager Refresh

START.

//CONTACT DSM TO UPDATE HHREFLIST FROM DSM.

dsmControl.getDSMHHRef( );

//RESET THE QUEUE.

setqList( );

END.

3.1.3. DSM

The following algorithms explain various functions of the DSM.

a. DSM Start

START.

Naming.bind("DSM", testdsm);

//INITIALIZE THE DSM LISTS FOR HHS

testdsm.sethhList( );

//WAIT FOR NEW ENTITIES TO REGISTER.(EVENT DRIVEN

//APPROACH)

END.

b. DSM HH Register

START

//REGISTER THE NEW HH

//If any entry matches with the new registrant i.e.

if (DSMList[i].name = = new registrant.name AND DSMList[i].IP = = registrant.IP) *//Already Registered*

status = true. *//make status true as HH is now active*

else

//add new HH to the list

DSMList[i].HHName= HHName.

DSMList[i].HHIP= HHIP.

status = true.

END.

c. DSM givRef() to QueryManager

START.

//SEND THE DSM HHREFLIST TO THE QUERYMANAGER

return DSMHHList[ ];

END.

d. DSM Deregister

START.

//CHECK IF ENTITY(HH) IS ALREADY REGSITERED OR NOT

if(DSMList[i].HHIP = = HHIP AND DSMList[i].HHName = = HHName)

DSMList[i].status = false.

else

return "Not Registered"

END.

e. DSM Authenticate( )

START.

//CHECK IF ENTITY(HH) IS ALREADY REGSITERED OR NOT

if (regName[i] = NAME AND regIP[I] == IP AND

regPass=password)

//AUTHENTICATE

END.

3.1.4. Active Registries (AR)

Algorithms to achieve various tasks of the ARs are described below.

a. AR Start

START.

//AR REGISTERCOMPONENTS

//find jini instances running

ServiceRegistrar[] registrars = evt.getRegistrars();

registrar = registrar[n];

//register the components with Jini.

reg = registrar.register(item, Lease.FOREVER);

END.

3.1.5. Link Manager

The algorithms for the Link Manager are taken from the URDS [45] and are unchanged in the MDE-URDS.

3.1.6. Adapter Manager

The algorithms for the Adapter Manager are taken from the URDS [45] and are unchanged in the MDE-URDS.

The further Sections explain the different enhanced architectures of MDE-URDS. Chapter 4 describes the different experiments performed along with their results on the MDE-URDS architecture described above and its further enhanced architectures.

3.2. Enhancing the MDE-URDS Architecture

The basic MDE-URDS architecture (Figure 3.2) contains mobile devices that host headhunters and clients. This basic architecture is further refined to incorporate various features (e.g., multi-level matching) in it. These enhancements are described in the following sections.

3.2.1. MDE-URDS with Multi-level Matching (MLM)

Typically in service discovery approaches, only type level matching is performed. The URDS [45] has also suggested multi-level matching (MLM) [11] for obtaining better quality of the results. In the MDE-URDS, the MLM consists of five levels of Type, Syntax, Semantics, Synchronization and QoS. Also, at every level, the matching operators can be exact or relaxed. With respect to the MDE-URDS, the exact and relaxed match semantics for different types of match are defined as.

Table 3.2 Exact and relaxed match Operators

Level	**Exact**	**Relaxed**
Type	Synonym (Exact)	Inheritance (Relaxed) Coercion (Relaxed)
Syntax	Synonym (Exact)	Inheritance (Relaxed) Coercion (Relaxed) Default Parameters (Relaxed) Parameter Order (Relaxed)
Semantics	Equivalence (Exact)	Implication (Relaxed) Reverse Implication (Relaxed)
Synchronization	Exact values	Compatibility
QoS	Exact values for attributes.	Comparability

The quality metrics used are the precision and recall [48] of the results. Precision is the total number of relevant services retrieved by a search to the total number of services retrieved by the search. Recall is the number of relevant services retrieved to the total number of relevant services present in the system.

It is seen from the results (explained in Section 4.2.1 of Chapter 4) that the precision of the results becomes better when MLM is used. In order to incorporate MLM in the HHs of MDE-URDS, following modifications are needed.

3.2.1.1. Headhunter MLM executeQuery( )

START

//HH RECEIVES QUERY

receive (query, timestamp).

// IFMLM QUERY

//PERFORM MLM MATCHING

//PERFORM TYPE MATCHING

if(query.componentname =contract.name)

resultsList.add(contract);

if (syntax matching enabled)

//PERFORM SYNTAX MATCHING AS

if(query.functionname=contract.functionname AND

query.parameters=contract.parameters AND

query.returntype=contract.parameters)

if (semantics matching enabled)

//PERFORM SEMANTICS MATCHING BY

//CALLING THEOREMPROVER.

if (QoS matching enabled)

//PERFORM QOS MATCHING

do for all attributes

if(contract.QoSval=query.QoSval)

QoSresultsList.append(contract)

//RETURN RESULTS DEPENDING ON QUERY MATCH LEVEL.

return resultList.

END.

However for the MDE-URDS to include multi-level matching, the HHs require the incorporation of a new entity called the HeadhunterProxy (HHProxy) for each HH. This is because the mobile HHs cannot handle the MLM processing due to the limited processing and memory capability of the mobile devices. Also, these multi-level specifications of services are written in XML and the mobile devices cannot process them due to their

JVM-related limitations. The MDE-URDS architecture therefore needs to be altered so as to include the proxies for the HHs supporting multi-level matching. However, not all HHs may support MLM in the architecture; as a result there are heterogeneous HHs in the system. Hence, the MDE-URDS architecture is now modified as follows:

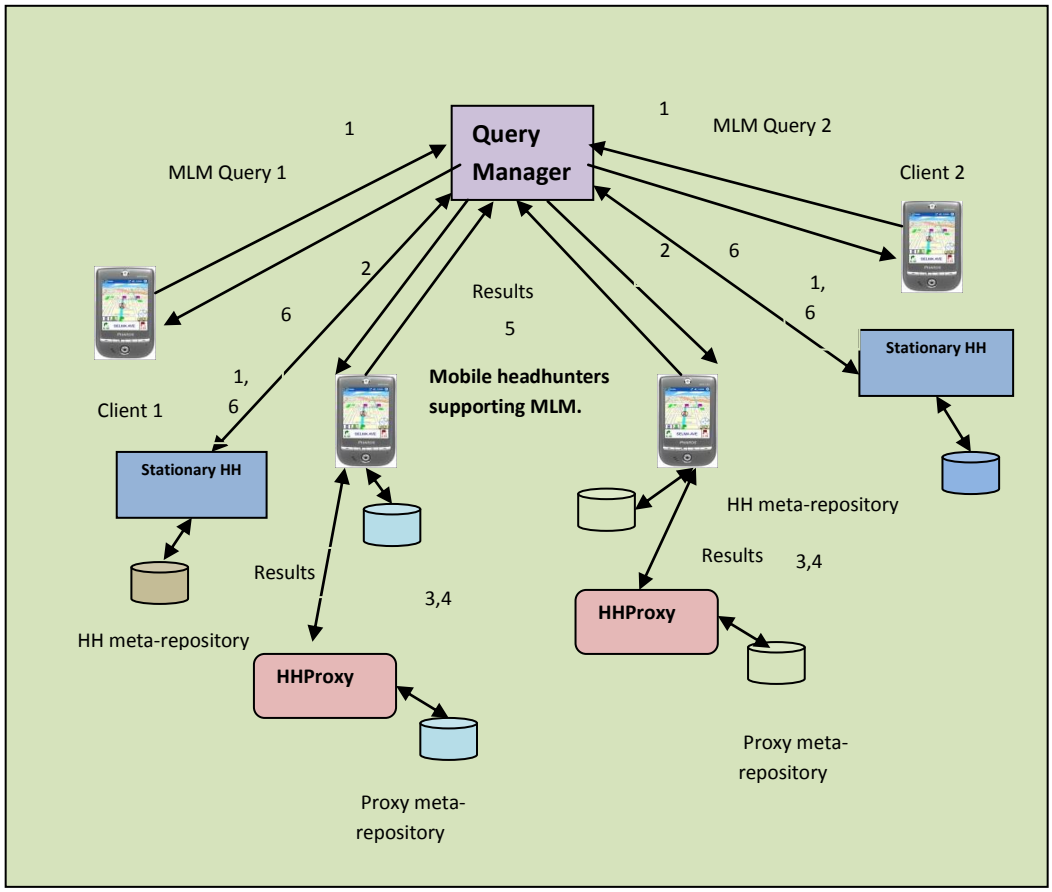


Figure 3.3 Use of proxy and Query Propagation of multi-level queries in MDE-URDS

A typical multi-level query processing scenario is depicted in Figure 3.3. The HH on checking if the query requires MLM, passes the query to its proxy who then handles it, evaluates it and sends back the results. This change requires the following modified algorithms for mobile HHs – these modifications are indicated in boldface. A new entity called as HHProxy is introduced and its algorithms are also described below.

3.2.1.2. HH Start

START.

//HH REGISTER AS A RMI ENTITY WITH A SPECIFIC NAME //WITH THE RMI ENTITY IN THE DEVICE (DESKTOP OR PDA).

Naming.bind (name, IP);

//JINI LOOKUP AND GET ACTIVE REGISTRIES CONTACT //FROM JINI.

lookup = new LookupLocator("jini://jinihost");

Jiniregistrar = lookup.getRegistrar( );

componentListObject =Jiniregistrar.lookup(template).

//UPDATE ITS META-REPOSITORY BY CONTACTING THE //ACTIVE REGISTRIES.

//HH invokes its Proxy to initialize itself.

contractList = componentListObject.getContractList2( );

hhProxy.initializeProxy();

END.

3.2.1.3. HH Query Execution

START.

//HH RECEIVES QUERY

receive (query, timestamp).

//IF QUERY IS NOT A MLM QUERY

if(query.matchlevel = = 0)

results=hh.executeQuery( ) ;

else

//pass the query to its Proxy

results= hhProxy.executeQuery(query, timestamp)

if (results not NULL)

return results.

else

return "No Components Found".

END.

3.2.1.4. HeadhunterProxy

Headhunter proxy is an entity for every headhunter required for it to give the multi-level matching capability. It is with the proxy that the headhunter can perform multi-level matching. It functions similar to a stationary headhunter for the matching process. The following algorithms are for proxy start, initialize and execute query.

a. HHProxy start

START

// REGISTER.

Naming.bind("HHProxy",this);

END.

b. Initialize Proxy

START

//FIND THE ACTIVE REGISTRIES USING JINI.

registrar =lookup.getRegistar( );

componentListObject = registrar.lookup(template);

//UPDATE THE REPOSITORY BY PARSING THE MLM //SPECIFICATIONS OF THE CONTRACTS.

contractList = componentListObject.getContractList();

END.

c. HHProxy executeQuery( )

START.

//PROXY RECEIVES QUERY

hhProxy.receive (query, timestamp).

//*EXECUTE QUERY (SAME AS **HH MLM EXECUTEQUERY**).*

//RETURN RESULTS TO HH.

return resultList.

END.

3.2.2. MDE-URDS Incorporating Caching and Buffering mechanisms

The previous architecture of the MDE-URDS (Figure 3.3) does handle the inability of processing XML of the JVM of mobile devices well; and provides support of MLM in the MDE-URDS. However, the response time increases (as indicated in the next chapter) due to the additional level of indirection of queries to the proxy. In order to reduce this overhead, other architectural changes such as caching or buffering of results are used. As a result, every headhunter now needs the presence of a buffer to cache the results of previous queries. Least Recently Used (LRU) policy is used for maintaining the buffer queues. The hit/ miss ratio is also a point of consideration to check the quality of results obtained.

The architecture is thus modified and the query processing is as in Figure 3.4.

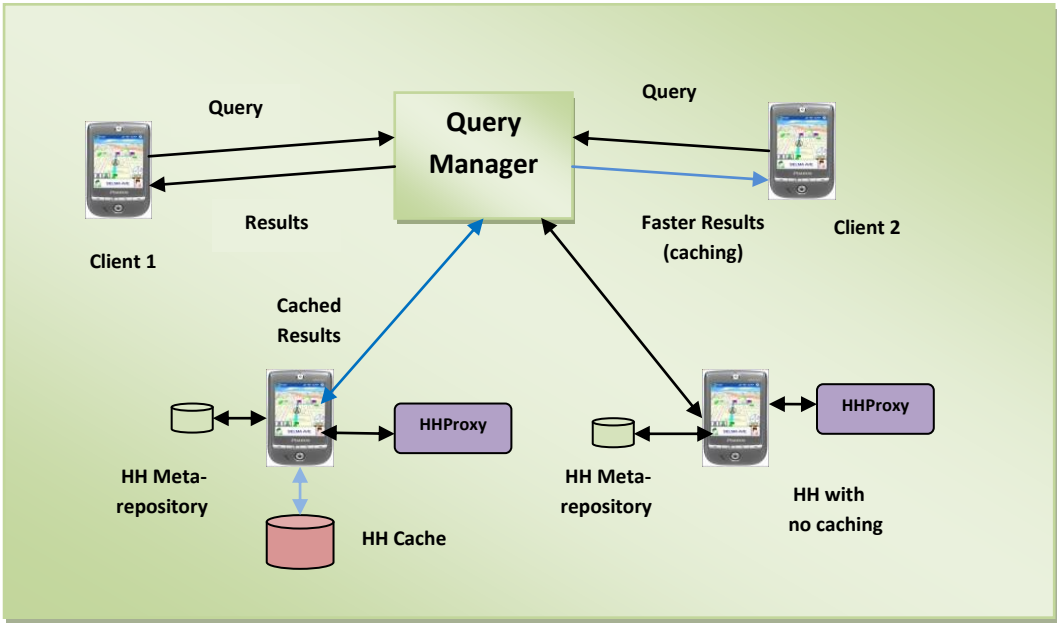


Figure 3.4 Query Propagation of multi-level queries in MDE-URDS using caching

Accordingly, the modified algorithm for query processing of a mobile headhunter is as follows.

3.2.2.1. HH Query Execution

```
START
//HH RECEIVES QUERY
        receive (query, timestamp).
//IF QUERY RESULTS PRESENT IN BUFFER LIST
        if( bufferList[].contains(query.componentName))
            results = bufferlist[curr].
        else
        //IF QUERY IS NOT A MLM QUERY
            if(query.matchlevel = = 0)
            results=hh.executeQuery( ) ;
            else
            //pass the query to its Proxy
            results= hhProxy.executeQuery(query, timestamp)
            if (results not NULL) //add to Buffer List for further queries.
                    if(bufferList NOT Full and bufferList ! contains results)
                            bufferList.add(results);
            return results.
            else
              return "No Components Found"
END.
```

3.2.3. MDE-URDS with different querying methods

The QM can follow different methods for selecting HHs for propagating incoming queries. The MDE-URDS supports three such methods of query propagation which are as follows:

a. Selective(domain specific) Search
b. Exhaustive Search
c. Random selection of HHs (random 3)

The main aim in all these approaches is to maximize one goal while compromising on the others (i.e. time vs quality of results).

- **Selective (Domain Specific HHs):** In this case, a domain-specific HH is contacted depending on the nature of the query. This is neither too time intensive nor does it suffer from poor quality. As the HH specializes in the domain whose service is needed by the query, the quality of the results is better. This is a good approach when the results are required within a given time bound.
- **Exhaustive:** This approach is an effort of delivering the best service to the user. So, the system queries all the HHs available and fetches the results. This improves the quality of results (i.e., recall) and gives the user a lot of choices for his query which was not the case in the selective search approach. However, this approach is time intensive and so; it is used when the results are not needed in a given time frame.
- **Random HHs:** This approach does not use any heuristics while contacting HHs. It queries random 3 HHs for processing the query.

  These three schemes are shown in Figures 3.5 and 3.6.

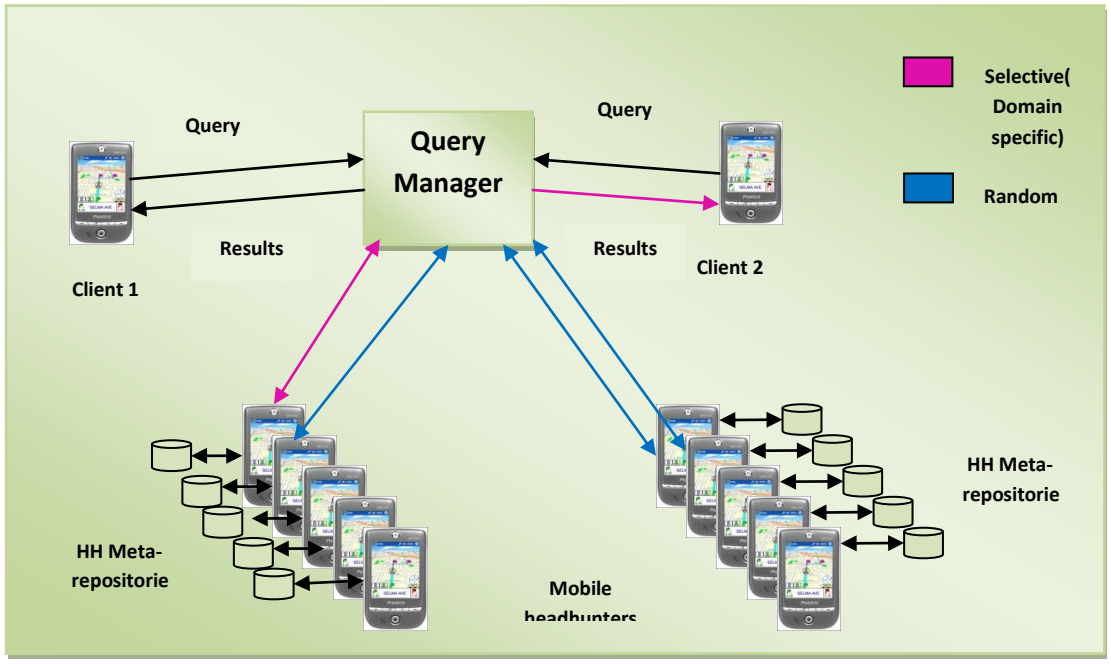


Figure 3.5 Random, Selective (Domain Specific search) of Query Manager

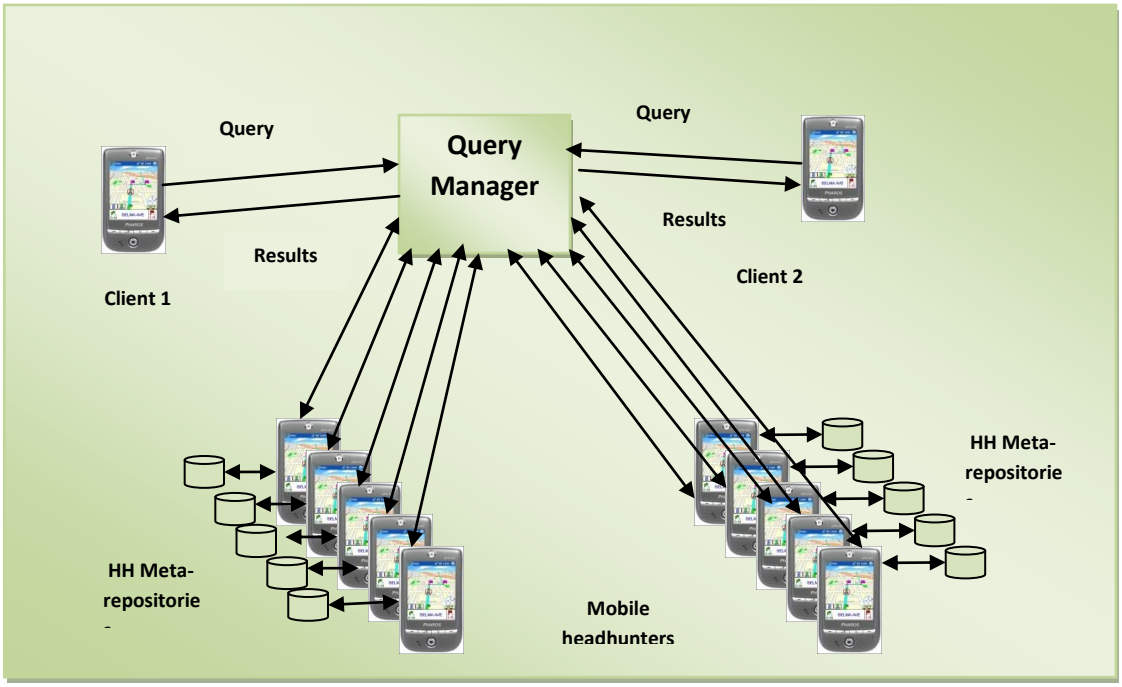


Figure 3.6 Exhaustive search of Query Manager

Hence, the algorithms for query processing are suitably changed for incorporating these schemes and are described below.

3.2.3.1. Query Manager:Query Processing

START

//PLACE NEW QUERY IN THE QUEUE.

recieveQuery(query,timestamp);

qList[curr]=query;

//SELECT THE NEW QUERY FROM THE QUEUE, SPAWN A NEW THREAD

RunQuery run=new RunQuery( );

*//SELECT A HH (BASED ON THE SCHEME USED) SEE //**HEADHUNTER-SELECT ALGORITHM.***

run.setParameters(hhRef,timestamp);

run.start( );

Thread processing:

results= hh.executeQuery ( );

//send results to QueryManager

QM.receiveResults(results,ID);

//Calculate time taken

time taken = System.currentTimeInMillis( )- qtimestamp;

return results.

END.

3.2.3.2. Headhunter-Select Algorithm

START

//QM DECIDES A SCHEME PROVIDED BY THE SYSTEM USER.

//CHECK THE SCHEME CHOSEN

if (Selective scheme)

//Pick a random HH from the available list of HHs //obtained from the DSM.

```
r = random.nextInt(hhCount).
HHIP= DSMList[r].HHIP
if(Exhaustive Search)
        while(hhcount ! = 0)
            HHIPList[hhcount] = DSMList[hhcount];
if(Domain Specific)
 //QM passes the query to the headhunter such that
     if(query.componentName IS A PART of hh.domain )
        HHIP = hhIP;
  if (Random Search)
   //QM passes query to random 3 HHs.
  while( i <3)
        r = random.nextInt(hhCount).
        HHIP= DSMList[r].HHIP.
  return HHIP[ ];
```

END.

3.2.4. MDE-URDS with Mobile IP (Incorporation of Mobility)

In the MDE-URDS, as mobile devices traverse from one location to another, their intermittent connectivity poses a challenge that needs to be addressed. To handle the mobility and network connectivity of these devices and to provide a transparent access to them whenever needed, the principles of Mobile IP [8] are used in the MDE-URDS.

Using Mobile IP, every HH in MDE-URDS has a home world, where it starts and registers with an agent called as “Home Agent” which is responsible for forwarding the queries to it appropriately. There is also a “Foreign World” that is an external world(domain) where the HHs can migrate to and register with an entity called as “Foreign Agent” which is responsible for appropriately routing of queries to the HHs from their respective Home Agents.

The query propagation with Mobile IP in MDE-URDS takes place as follows:

- If the mobile HH is in its home world, then the query is sent directly to it,
- When the mobile HH is in any foreign world, then the query is routed from its Home Agent to its Foreign Agent and then to the mobile headhunter.

Also, the problem of intermittent connectivity is solved by the node's Home Agent, which can buffer the queries or pass it on to its neighbors, depending upon the client's requirements, when the mobile device is not accessible. Experiments dealing with the performance of Mobile IP implementation can have different scenarios for propagating queries to the mobile headhunters such as: when headhunters are in their home world, when the headhunters are connected to an external foreign world, when the headhunters are not connected to any world or are in transit. The results related to these experiments are explained in Chapter 4.

With the inclusion of Home and Foreign Agents, the MDE-URDS architecture is modified as shown in Figure 3.7.

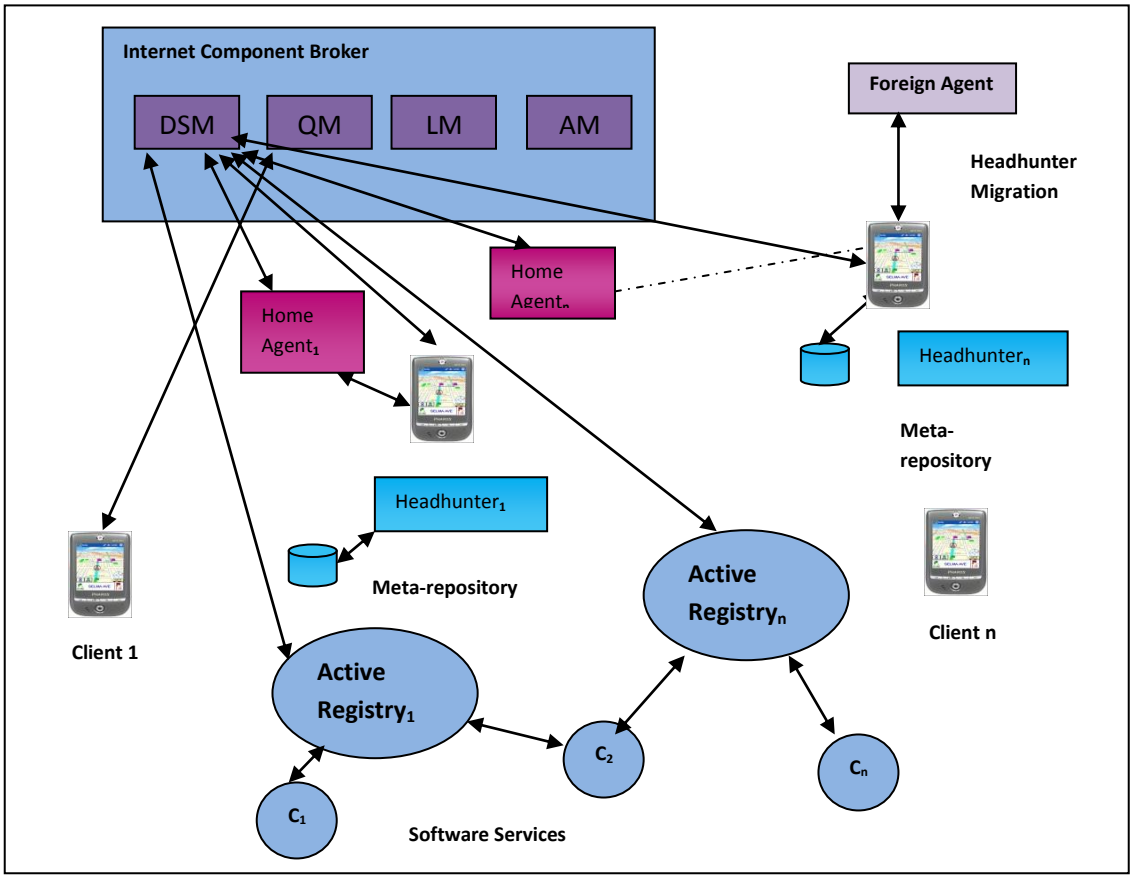


Figure 3.7 MDE-URDS architecture with Mobile IP implementation

The modified algorithms for the HH, QM, and DSM are shown below along with algorithms for the Foreign Agent and Home Agent.

3.2.4.1. HH Start with Headhunter Migration: Mobility of headhunters

START.

//HOMEAGENT LOOKUP AND REGISTER

Naming.lookup("//"+homeAgentIP+"/HomeAgent");

haControl.registerMobileNodes(myName, IP);

//JINI LOOKUP AND GET ACTIVE REGISTRIES CONTACT FROM JINI.

//UPDATE ITS META-REPOSITORY BY CONTACTING THE ACTIVE

REGISTRIES AND INITIALIZE PROXY

//DO THE FOLLOWING

if (HH wants to move to other Domain and already registered)

//deregister with Foreign Agent.

faControl.deregisterMobileNodes(myName,IP);

else

//inform Home Agent

haControl.setMobileNodeStatus(1,myName,IP);

//register with new Foreign Agent.

faControl.registerMobileNodes(myName,IP,homeAgentIP;

//call to Foreign Agent to inform Home Agent.

faControl.informHA( );

WHILE (REPLY=="YES");

END.

3.2.4.2. Query Manager: Query Processing

START

//PLACE NEW QUERY IN THE QUEUE.

//SELECT THE NEW QUERY FROM THE QUEUE, SPAWN A NEW THREAD

SELECT A HH (BASED ON THE SCHEME USED) SEE ***HEADHUNTER-SELECT ALGORITHM.***

Thread processing:

//Pass the query to the HH's Home Agent by

results = haControl.returnResults ( );

return results.

END.

3.2.4.3. DSM HH Register

START.

//If any entry matches with the new registrant entity i.e.

if (DSMList[i].name = new registrant.name AND DSMList[i].IP = = registrant.IP)

else

//ADD NEW HH TO THE LIST WITH HOMEAGENT DETAILS

DSMList[new]=hhDetails.

END.

3.2.4.4. DSM getHAIP, HHIP, HHName, HAName

START.

//CHECK IF HH REGISTEREDOR NOT

if(DSMList[i].HHIP = = HHIP AND DSMList[i].HHName = = HHName)

return the suitable name or IP.

else

return false;

END.

3.2.4.5. Home Agent

Home Agent is the entity that is present one for every headhunter. It keeps a track of the headhunter location in any domain it registers to. Thus, algorithms such as register headhunters, update headhunter location and query passing is a part of the HomeAgent functioning.

a. HomeAgent start

START.

Naming.bind("HomeAgent", this);

//WAIT FOR ENTITIES TO REGISTER(EVENT DRIVEN).

END.

b. HomeAgent Register

START.

//ADD THE MOBILE ENTITY AS ITS REGISTERED MOBILE DEVICE

HHname= mobilenodename;

HHIP=mobilenodeIP;

//TAKE ITS REFERENCE FOR QUERYING

HHRef = mobilenodeRef

END.

c. HomeAgent UpdateMobilenodestatus

START.

//UPDATE THE NEW IP AND FOREIGN ADDRESS

if (HHname= mobilenodename AND HHIP=mobilenodeIP)

HHIP= newIP ;

FAIP = foreignIP;

END.

d. Home Agent passQuerytoHH

START.

//RECEIVE QUERY FROM QM

receiveQuery(query, timestamp);

//CHECK MOBILE NODE STATUS

if(status= =0)

//pass query directly to the HH

results = hh.executeQuery( );

else if (status= =1) *//registered with a foreign agent.*

//pass query to the FA

results = faControl.receiveResults(query, timestamp);

else

//default store the pending queries in the pending query list.

pqList[i]=query;

//RETURN RESULTS TO THE QM.

return results;

END.

3.2.4.6. Foreign Agent

Foreign Agent provides a way for headhunters to register with a foreign world and route their queries appropriately. The algorithms described are for FA start, register and pass query to headhunter.

a. Foreign Agent Start

START.

//FOREIGN AGENT REGISTERS WITH THE DSM.

Naming.bind("ForeignAgent", this);

//INITIALIZE ITS MOBILENODE LIST AND WAIT FOR NODES TO

//REGISTER(EVENT DRIVEN).

setMNVariables( );

END.

b. Foreign Agent register

START.

//CHECK AND REGISTER THE HH.

if (hhList[].IP ! = hhIP AND hhList[].name ! = hhName)

//then add the HH to the list

hhList[i].IP = HHIP,

hhList[i].name = HHName

END.

c. Foreign Agent passQuery

```
START.
//CHECK IF REGISTERED AND PASS QUERY
    if (HH registered)
            //pass query to the HH
            results= HH.executeQuery( )
//RETURN RESULTS TO THE HOMEAGENT.
    return results.
END.
```

For the mobile HHs, there are scenarios of pending queries when the headhunters are not registered (in transit). As a result, waiting for the headhunter (solution 1) or querying other headhunters (solution 2) also impacts the architecture. For the first case there is wait time (in case of quality aware query, where it wants to wait for that particular headhunter for some specific services, domain specific headhunters are a good example of this), whereas in the second case there is no wait time associated, but may not guarantee quality.

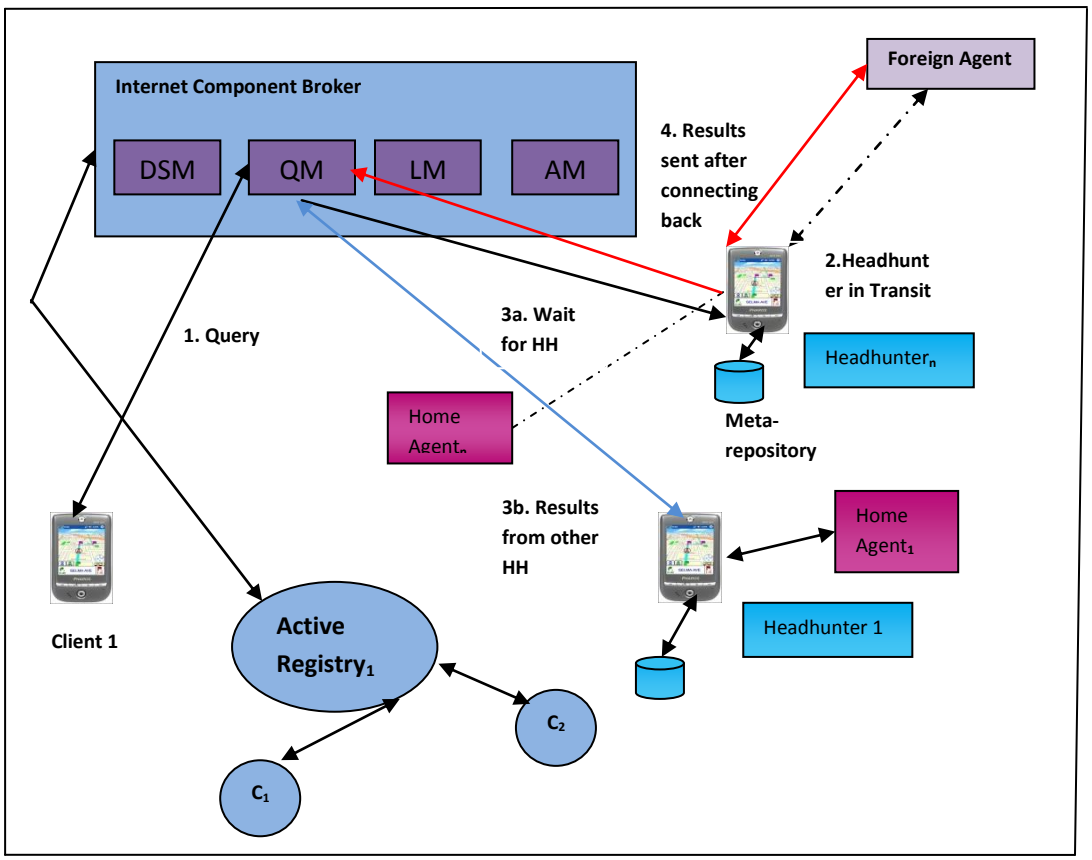


Figure 3.8 Query propagation in Mobile IP scenario, headhunters in transit

In Figure 3.8, two different approaches to query propagation are seen – when headhunter is in transit, the query manager decides to send the query to another headhunter or decides to wait for the respective headhunter to connect back and query it later. In both the approaches, the response time and the quality of the results vary and so the selection of the approach must be according to the application. This is more detailed in Chapter 4.

The query processing algorithms accordingly vary as shown below.

3.2.4.7. Query Manager: Query Processing for headhunters in transit (Solution 1)

START.

//PLACE NEW QUERY IN THE QUEUE.

//SELECT THE NEW QUERY FROM THE QUEUE, SPAWN A NEW THREAD

//SELECT A HH BASED ON THE RANDOM SCHEME

Thread processing:

//Pass the query to the HH's Home Agent by

results= haControl.returnResults ( );

return results.

//IF HH IS UNAVAILABLE

if (hhstatus= = false)*//HH status is false, buffer the queries in a pending query list.*

pqList [count++] = query.

return false.

END.

3.2.4.8. Query Manager: Query Processing for headhunters in transit (Solution 2)

START.

//PLACE NEW QUERY IN THE QUEUE.

//SELECT THE NEW QUERY FROM THE QUEUE, SPAWN A NEW THREAD

//SELECT A HH BASED ON THE RANDOM SCHEME

Thread processing:

//Pass the query to the HH's Home Agent by

results= haControl.returnResults ( );

//IF HH IS UNAVAILABLE

if (hhstatus= = false) *//HH status is false, query another headhunter.*

//SELECT ANOTHER HH BASED ON THE RANDOM SCHEME

Thread processing:

//Pass the query to the HH's Home Agent by

results= haControl.returnResults ( );

return results.

END.

3.2.5. MDE-URDS with Collaborative Approaches (Incorporation of Collaborative Approaches)

In the above architectures (Figures 3.2 to 3.8), the query manager is responsible for propagating the queries to the respective headhunters. In order to avoid the query manager from being the bottleneck for query processing, the task of selecting headhunters for handling the queries can be assigned to the headhunters by propagating them among its neighbors. Also, every headhunter has a set of services registered with it. Getting response from a single headhunter for a query makes the resultant set often limited and may not provide the most relevant services for a query. In order to tackle this drawback, concepts from the domain of Mobile Ad-Hoc Networks are employed in the MDE-URDS. Mobile Ad-Hoc Networks (MANET) is a self-configuring mesh network where the individual mobile nodes (sensors) are connected by wireless links and coordinate with its neighbors for a particular task. There are different routing protocols used by the nodes that take the reactive as well as proactive approach of maintaining routing information. Amongst them, Ad-hoc On Demand Vector (AODV) Routing [9] and Dynamic Source Routing (DSR) [40] are reactive MANET protocols where the nodes maintain each other's information; however the path of communication is decided when needed. When information has to be passed to a node, the sender node decides a path looking at its routing table and sends the information. This path can be decided by the sender node itself (as in DSR) or it can change at the intermediate nodes depending on the availability of the nodes (as in AODV). For the proactive approach, the On-Demand Link State Routing (OLSR) protocol is used wherein any change in the routing table is propagated to every node and they change their routing entries to suit the changes.

In order to incorporate the principles of these protocols, a topology of headhunters was created, which was that of an irregular graph of the headhunters. For every headhunter in the graph, the headhunters that are connected to it by a direct link are called its

neighbors. As a result, in the MDE-URDS, every headhunter maintains its neighboring HH information and propagates the query to its neighbors when needed. They propagate the messages to its neighbors depending on whether suitable results were achieved or not. It, however, maintains neighbor's information and uses that to send it across and does not depend upon the other intermediate nodes for determining the next path, similar to the DSR protocol. It is still reactive as the information may not be up-to-date.

However, unlike multi-hop paths used in MANETs, only one-hop paths, i.e., the HHs pass the queries only to its immediate neighbors are used in this approach due to time limitations. This also helps to reduce consumption of power and communication in case of mobile devices. The three MANET protocols mentioned above are selected and their principles are adapted and incorporated in the MDE-URDS for the collaboration among HHs. These three collaborative approaches are: "I do it", "We do it", and "You do it".

"I do it" approach is similar to querying a single headhunter and the headhunter does not involve results from other headhunters. This cannot be actually termed as a collaborative approach but since the headhunter decides on the query processing approach, instead of the query manager, this experiment is included in this category. For this approach, the HHs are queried individually and they do not propagate the queries to any other headhunters, unless they have no results to a query or is busy with other queries. The algorithm for "I do it" approach is as follows:

3.2.5.1. HH Query Execution: No Collaboration (I do it)

```
START.
//HH RECEIVES QUERY
        hh.receive (ContractQueryImpl query, long timestamp).
//EXECUTE THE QUERY
          results = hh.executeQuery( ) ;
          if (results not NULL)
                  return results.
          else
                  //else pass it to its neighbor.
                          return hhNeighbor[i].executeQuery( );
END.
```

"We do it" approach includes results from the queried headhunter and its neighbors. In this, the headhunters collaborate their results to obtain better recall for a query. These techniques are useful when the HH graph is not fully connected and there is a need to improve the quality of results of the system. The two approaches used here from the MANET domain- the reactive and the proactive approach using the AODV and OLSR protocols respectively. Similar to the protocols, HHs maintain neighbor information and propagate the queries to them for collaborative responses.

In MDE-URDS, the headhunters when moving to a foreign domain update their current location information to its neighbors and as a result the headhunters always have the updated information about neighboring headhunters. As only one-hop paths are considered, the updated information is propagated only to immediate neighbors.

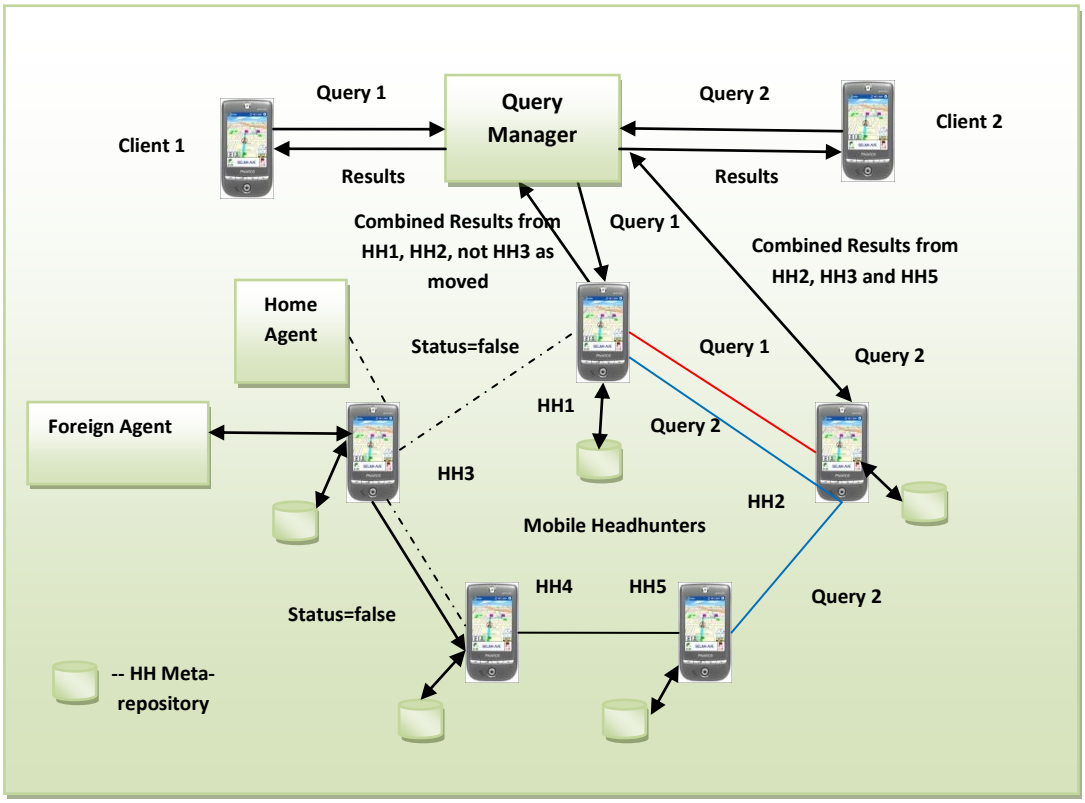


Figure 3.9 Collaborative approach: We do it using AODV and OLSR protocol

In Figure 3.9, the "We do it" approach is depicted. Here, the query 1 is routed to the HH1, which then passes it to its neighbors, HH2 and HH3. However, by using the OLSR protocol, the HH3 sends message to its neighbors indicating it is moving to another domain and as a result, the results sent back that is a combination of results from HH1 and HH2, not HH3. Similarly for query 2, HH2 is selected by the query manager; HH2 passes the query to its neighbors, HH5 and HH1 and sends back the combined results back to the query manager.

The algorithms for query processing at HH are as follows.

3.2.5.2. HH Query Execution: Collaboration (We do it)

START.

HH RECEIVES QUERY

hh.receive (ContractQueryImpl query, long timestamp).

//DEPEND ING ON LEVEL OF QUERY, EXECUTE OR PASS TO PROXY.

//PASS QUERY TO OTHER HHS IN ITS NEIGHBORS LIST CALL

while(i!=hhNeighborCount)

results=HHneighbour[i++].executeQuery(query).

// Combine all the results its result list.

resultList.append(results);

return resultList.

END.

The "You do it" approach includes delegation of the query processing from the queried headhunter to its neighbors as shown in Figure 3.10. When a headhunter is busy, it passes the query to its neighbors and sends back the response. The response does not include the queried headhunter's results.

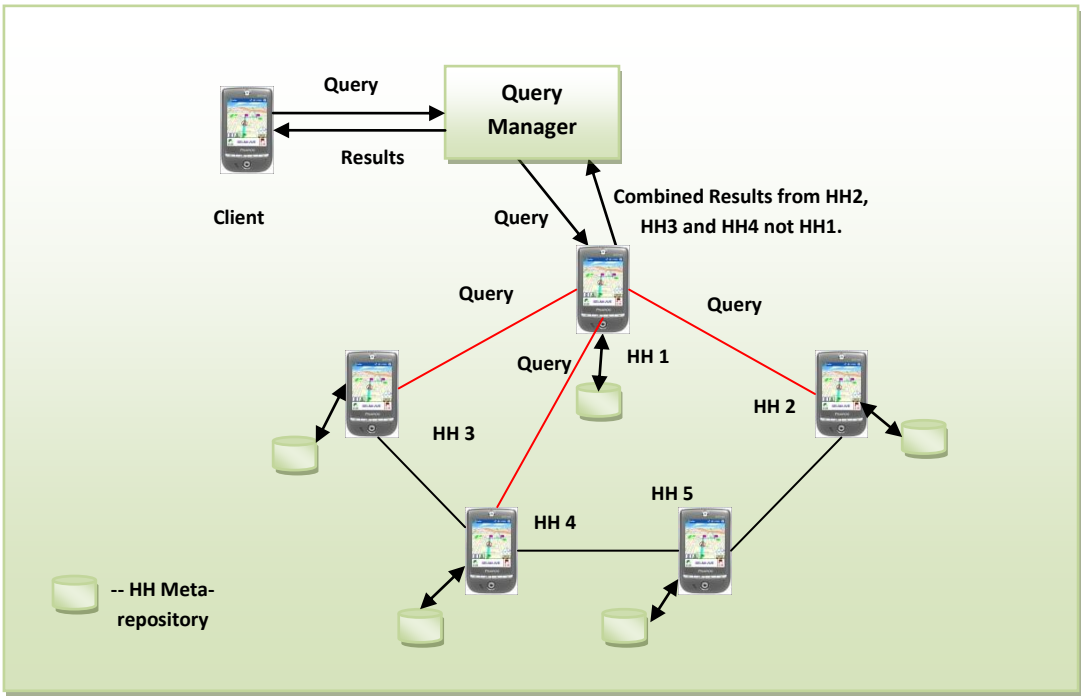


Figure 3.10 Collaborative approach: You do it

3.2.5.3. HH Query Execution: Collaboration (You do it approach)

START.

HH RECEIVES QUERY

hh.receive (ContractQueryImpl query, long timestamp).

IF HH IS BUSY WITH QUERIES OR QUERY IS DS AND HH HAS NO ANSWER

if(hhstatus=busy)

//pass query to other HHs in its neighbors list call

while(hhNeighborCount ! = 0)

results=HHneighbour[hhNeighborCount - -

].executeQuery(query).

// Combine all the results its result list.

resultList.append(results);

return resultList.

END.

The results from these searches are included in Chapter 4 in detail. Thus, the collaborative approaches are a way to improve the quality of the results and are reliable as they are based on the widely accepted MANET protocols.

Thus, the final MDE-URDS architecture which contains all these enhancements is shown in Figure 3.11.

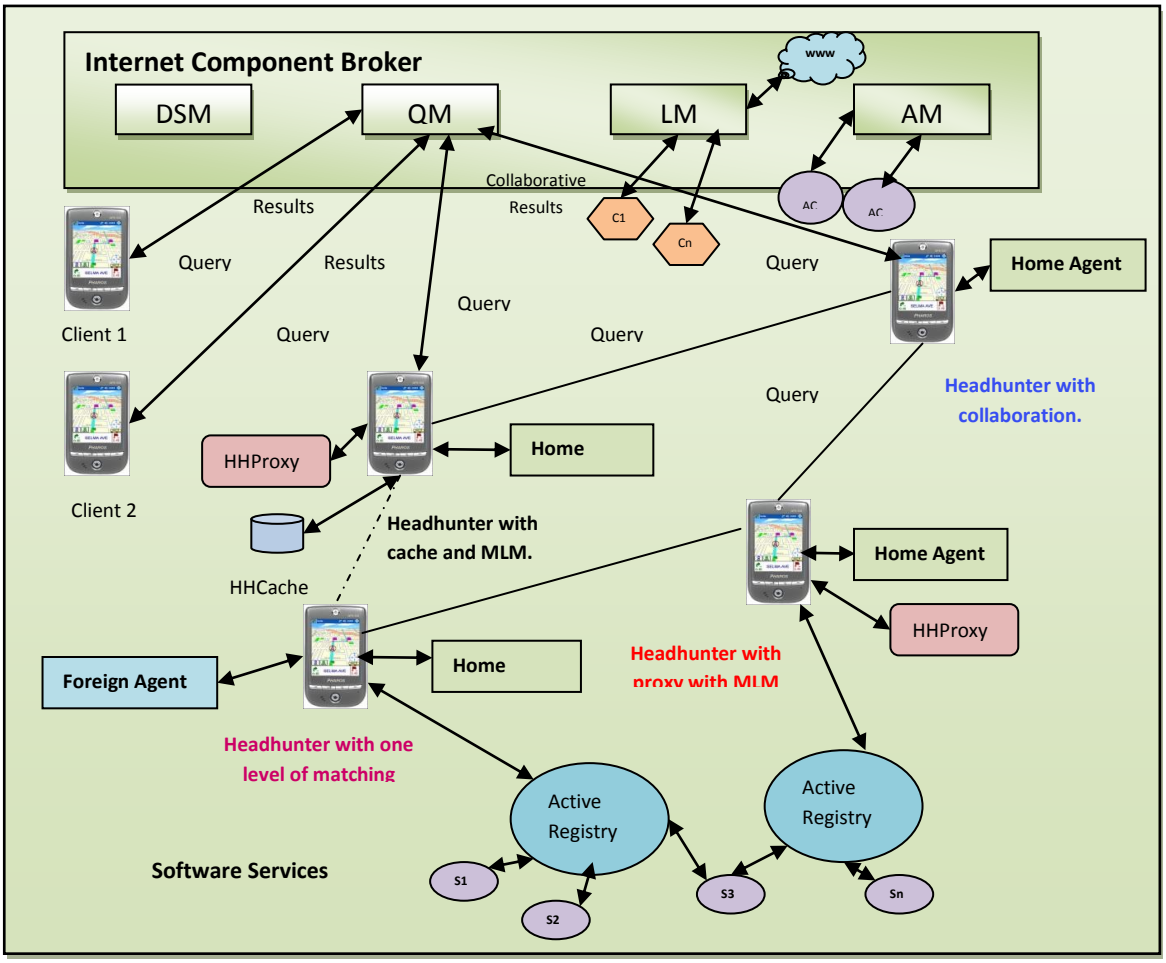


Figure 3.11 Enhanced MDE-URDS Architecture

This final architecture includes Mobile IP for handling the intermittent connectivity and mobility, HHProxies for helping in offloading of MLM queries, HH caches for buffering of results and different query processing mechanisms of selective,

random and domain specific querying. The HHs shown are heterogeneous where some are mobile, some stationary, some handling MLM using a proxy, some using collaborative approaches for query processing.

Chapter 4 describes the empirical validation carried out with each version of the MDE-URDS architecture (Figures 3.2 to 3.11) along with the suggestions for future enhancements.

CHAPTER 4. EXPERIMENTATION AND VALIDATION

This chapter describes an empirical validation of various MDE-URDS architectures that were presented in Chapter 3.

The experimentation setup was as follows- the mobile devices used were Pharos Traveler GPS 525 PDAs running Windows Mobile 5 operating system and twelve desktop machines that use Windows XP operating system. The wireless connectivity was provided by the different wireless networks on campus and also by a private access point for a controlled experimental setup, whereas the desktop machines are connected by a local area network (LAN). The entire MDE-URDS implementation is created using Java RMI, and J9 JVM [49] is used for running the Java classes on the Pharos PDAs. Services from the publically available QWS dataset [50] were used in the experiments. The current version of the QWS dataset consists of around 5,000 Web services, out of which 365 are made available for public usages. Each service contains nine QoS attributes (throughput, reliability, response time) measured using commercial benchmark tools. A few services from these were selected and their multi-level specifications, consisting of type, syntax, semantics, and QoS levels, were manually created. The synchronization level was not used in the matching process of the MDE-URDS as the synchronization contracts for these services could not be created due to the unavailability of their source code. An example of multi-level specification for a service is shown in the Figure 4.1.

```
<?xml version="1.0" encoding="UTF-8"?>
<proURDSContract name=FastWeather" type="Weather">
       <ComponentAttributes>
              <property name="DomainName" value="weather"/>
              <property name="SystemName" value="DOTSFastWeather"/>
              <property name="Discription" value="For more information on our
              web services, visit us"/>
              <property name="auther" value="www.serviceobjects.com"/>
       </ComponentAttributes>
       <ComputationalAttributes>
              <InherentAttributes>
              </InherentAttributes>
              <FunctionalAttributes>
                     <SyntaxAttributes>
                            <ContractAttributes>
                                   <Contract>
                                          <property name="methodName"
                                          value="GetWeatherByZip"/>
                                          <property name="ret_Weather"
                                          value="string"/>
                                          <property name="param_postalcode"
                                          value="string"/>
                                   </Contract>
                                   <Contract>
                            <property name="methodName"
                            value="GetWeatherByCityState"/>
     <property name="ret_Weather" value="string"/>
     <property name="param_state" value="string"/>
    </Contract>
                            </ContractAttributes>
                     </SyntaxAttributes>
                     <SemanticAttributes>
                            <PreConditon>
                            </PreConditon>
                            <PostCondition>
                            </PostCondition>
                     </SemanticAttributes>
              </FunctionalAttributes>
       </ComputationalAttributes>
       <QOSAttributes>
              <property name="Availability" scale="percentage" value="94"/>
              <property name="Reliability" scale="percentage" value="96"/>
              ……
              <property name="Security" scale="percentage" value="99"/>
       </QOSAttributes>
       <SynchronizationAttributes>
              <property name="N/A" value="N/A"/>
       </SynchronizationAttributes>
       <CooperationAttributes>
              <property name="N/A" value="N/A"/>
       </CooperationAttributes>

       <DeploymentAttributes>
              <property name="N/A" value="N/A"/>
       </DeploymentAttributes>

       <AuxillaryAttributes>
              <property name="N/A" value="N/A"/>
       </AuxillaryAttributes>
</proURDSContract>
```

Figure 4.1 Multi-level XML Specification of a Service

Figure 4.1 shows the multi-level specification of a Weather Service. As described in Chapter 3, the MLM description consists of four levels- Syntax, Semantics, Synchronization and Quality of Service (QoS). The above description includes the general service attributes (short description, version, etc.) and also the four levels of Syntax (that include its methods, method parameter types, method return types), Semantics (the preconditions, post conditions of the methods), Synchronization (use of synchronization methods, etc.), and the Quality of Service (QoS) attributes (Reliability, Availability, etc.) for the Weather Service.

In Chapter 3, various versions of the MDE-URDS architecture, from basic to the final, were described. Each of these versions was experimented with as described in following sections.

4.1. Study of performance of the basic architecture of the MDE-URDS

This category of experiments is carried out on the basic architecture of MDE-URDS which includes the HHs deployed on mobile devices supporting basic type matching. It is described in Section 3.2.1, Figure 4.2. The purpose of this set of experiments was to check how the mobile headhunters of MDE-URDS fare with respect to the stationary headhunters in terms of response time and the quality of results obtained. The stationary headhunters that are used for comparison are similar to the ones of the URDS architecture (MDE-URDS being an extension of URDS) and have already been tested for performance in terms of response time and the quality of results against the publicly available jUDDI [51] in [52]. It shows that these stationary HHs fare well when compared with the matching semantics of the jUDDI, in terms of both response time and the quality of results.

The following configuration is used for the set of experiments (4.1.1- 4.1.4).

- Number of components: 100, 10/HH.
- Matching used: Type matching.
- Distribution of Components: Random.
- Number of HHs: 20, 10 S-HHs and 10 M-HHs.

In this category of experiments, the response time for every query is an average over 10 readings.

4.1.1. To study the performance of the mobile headhunters in comparison to stationary headhunters with respect to average response time

This experiment was carried out to compare the response time taken by the stationary headhunters and the mobile headhunters of the MDE-URDS architecture. This response time and also the quality of the results is necessary for the system developer who composes the system from the discovered services and needs to adhere to the timing restrictions. Also, for applications that are time critical, such as real-time tracking, these response time will determine if mobile headhunters would be suitable for such applications.

This experiment is carried out by querying the mobile and stationary headhunters for a random set of queries for observing the difference in their response times.

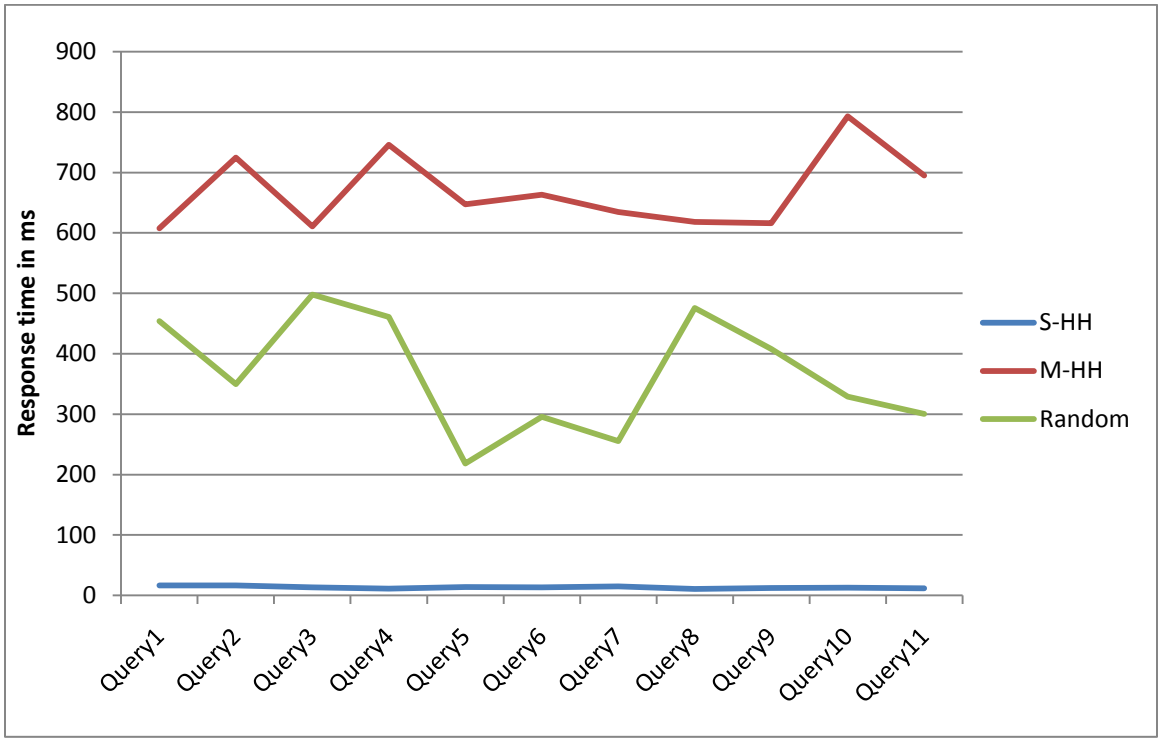


Figure 4.2 Average time taken by Mobile Headhunters vs Stationary Headhunters

The above graph (Figure 4.2) shows the average response time of the MDE-URDS for a given set of queries from mobile headhunters (M-HH), stationary headhunters (S-HH) and a randomly (Random) chosen headhunter. The average time taken by a mobile headhunter is around 700 ms whereas that taken by a stationary headhunter is 32 ms. The random selection of headhunters, gives an average response time of 350 ms.

The average time taken by mobile headhunters is more due to factors such as the involved wireless communication and the processing capacity of the mobile device (PDA). However, the graph for M-HHs does not show any predictable trends due to the fluctuating nature of wireless connectivity. This observation is reinforced by the ping experiment with these PDAs as shown in Figure 4.3.

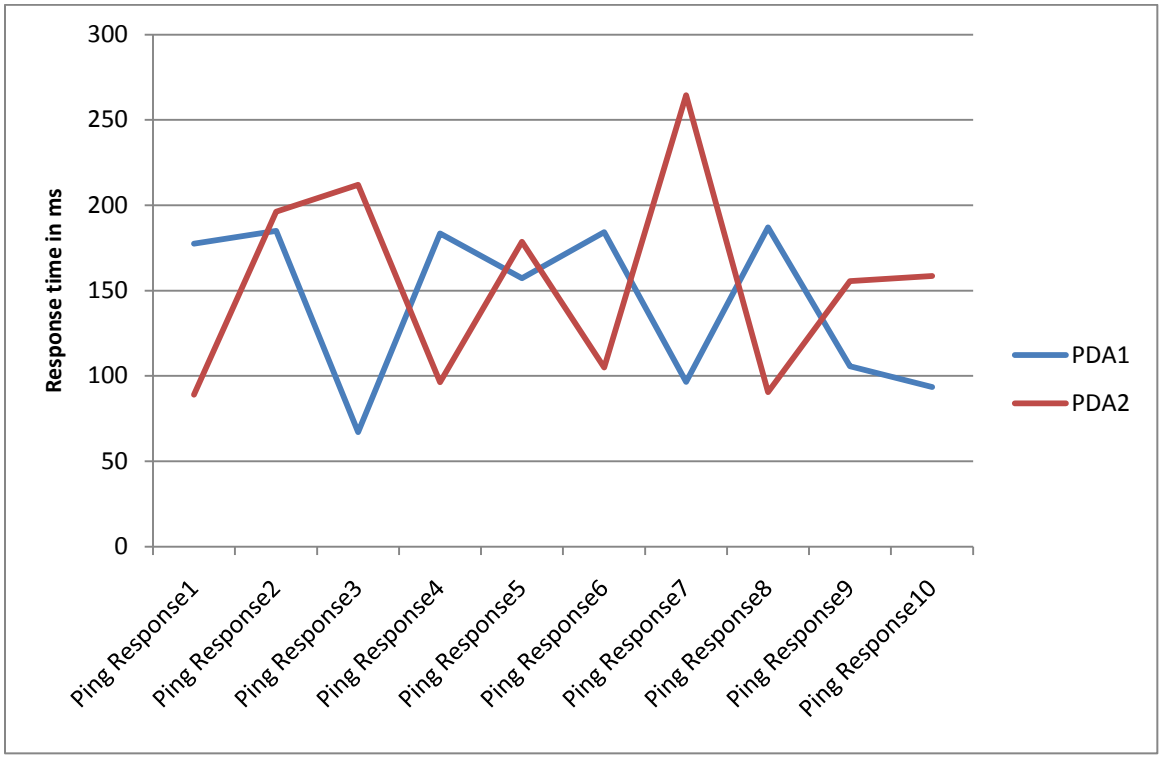


Figure 4.3 Changes in Ping Response time for a wireless HH

4.1.2. Calculation of individual times

The previous experiment showed that the response time of mobile headhunters is significantly higher than that of stationary headhunters. The purpose of this experiment was to identify the precise reason for this high response time of the mobile headhunters by breaking down their total time into individual parts required for processing a query. This would be helpful in improving the response time by exploring the opportunity of minimizing any of the individual times.

On analysis of the response time for the mobile headhunter, it was seen that the total time was divided into two main parts of: End-to-End Response Time and Processing Time.

- **End-to-End Communication Time:** This is the time taken for making a remote call to the headhunter and getting back the results from it.

- **Processing time:** This is the actual time taken by a headhunter for processing a query, from the time it arrives to the time it obtains the results for the query.

These time calculations were under consideration of a single query response.

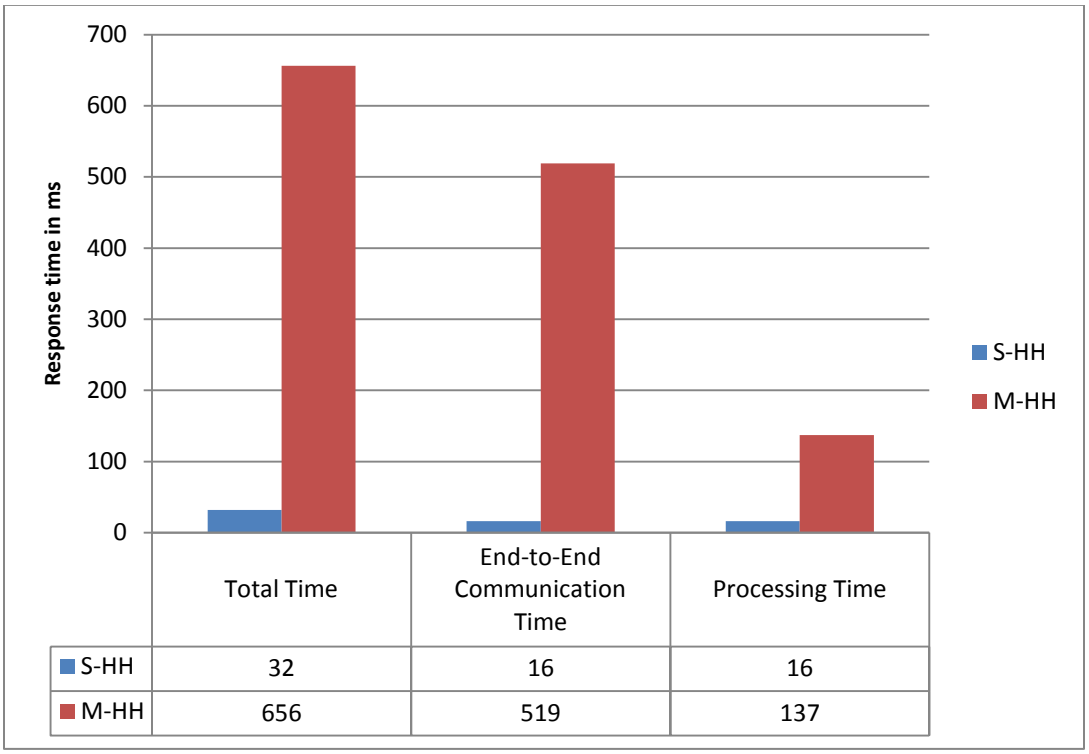


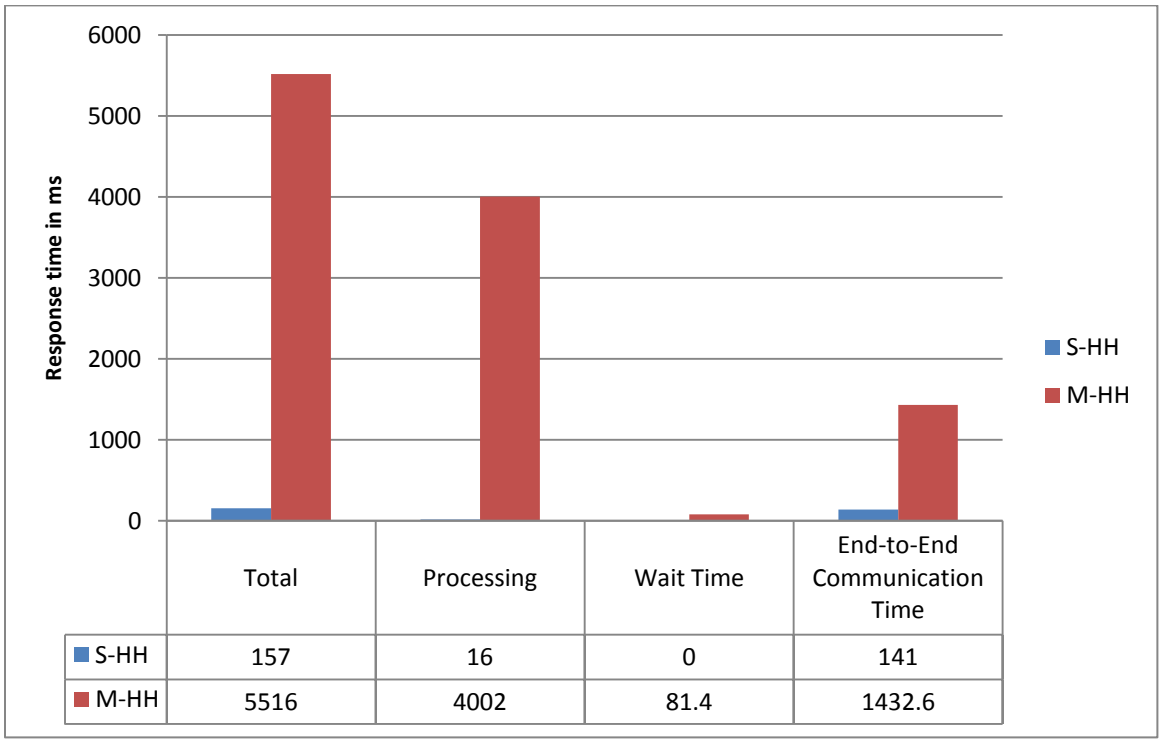
Figure 4.4 Division of response time for mobile and stationary HHs

Figure 4.4 show that the RMI calls dominate the time taken by the mobile HHs. This is again because of the wireless calls and the limited processing power of the PDA that makes the processing in the device to take more time than a stationary headhunter deployed on a resourceful device.

4.1.3. Studying the wait time for the headhunters

In the Section 4.1.2, the individual time taken by a headhunter was evaluated. However, when several queries are sent to the headhunter at a time, the wait time also plays a significant role in the overall response time. Wait Time calculation was also done

for a selected number of queries (here 10) sent at a time to a headhunter. Wait time calculation is important for improvement in response time. If the wait time for a query exceeds more than a threshold value (decided by the application), then other techniques such as passing of queries to other headhunters, caching or dropping of queries can be followed. This would help in maintaining a consistent response time for the queries. The typical wait time at the headhunter for 10 queries was found and the results are noted in Figure 4.5.



	Total	Processing	Wait Time	End-to-End Communication Time
S-HH	157	16	0	141
M-HH	5516	4002	81.4	1432.6

Figure 4.5 Calculation of wait time at a headhunter

It is seen from Figure 4.5 that Avg. Wait time: S-HH: 7.5 ms, M-HH: 240 ms. A typical response time is considered for the S-HH and M-HH and is noted in Figure 4.5. The graphs show that the Wait Time for S-HH is negligible and so is the processing time. M-HH has significant processing and RMI time due to which every query has an additional wait time, waiting for the previous queries to be processed.

Another noteworthy time calculation was that of the Matching Time. The Matching Time is the time taken by a headhunter to match a specific query to the components registered with it, meeting the different attributes of the query. This is more significant when considering multi-level queries. For the initial setup, where matching is only based on the type, the matching time was found to be negligible in comparison with the overall processing time.

4.1.4. To study the scalability of the system

For the basic MDE-URDS architecture described in Section 3.2.1 of Chapter 3, a scalability study was carried out to check the limitations of the basic MDE-URDS architecture. This study was performed with respect to: several queries in the system, several queries at a single headhunter and the number of HHs in the system.

In these experiments, the HHs maintain a buffer for the incoming queries, S-HHs maintain a queue size of 100 while the M-HHs maintain a size of 10 due to their limited memory capacities.

4.1.4.1. With respect to several queries in the system

In this experiment, the scalability of the basic MDE-UDRS architecture (described in Section 3.2.1) is tested by passing random simultaneous queries at a uniform rate to only ten stationary headhunters, then to ten mobile headhunters and also to a hybrid configuration consisting of stationary and mobile HHs. The HHs are selected on a round robin basis for load balancing. The results of this experiment are shown in Figure 4.6. As seen from Figure 4.6, with increasing the number of queries, the response time increases non-linearly in the case of mobile HHs; while in the case of stationary HHs, the time increase is uniform. This is due to the higher wait time (which is dependent on the processing of earlier queries, the arrival rate of the incoming queries and the processing rate) in the case of mobile HHs. For the given setup of passing random queries at a uniform rate, it is seen that the threshold for the number of queries in the system is around 200-250, after which the queries are dropped.

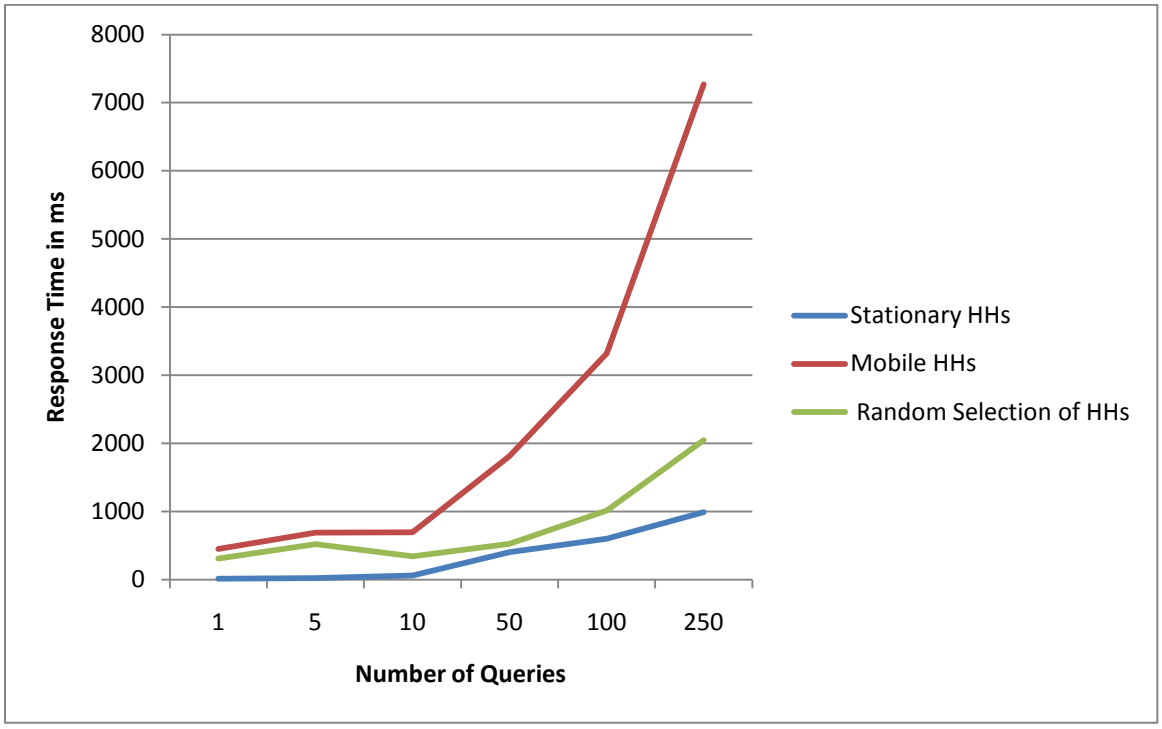


Figure 4.6 Scalability w.r.t several queries in the system

4.1.4.2. With respect to several queries at a single headhunter

This experiment helps understand the limitation of every headhunter in terms of the number of queries it can process within a certain upper bound of the response time. This is useful for load balancing of the headhunters.

Figure 4.7 shows the number of queries at a given headhunter along with the associated response times.

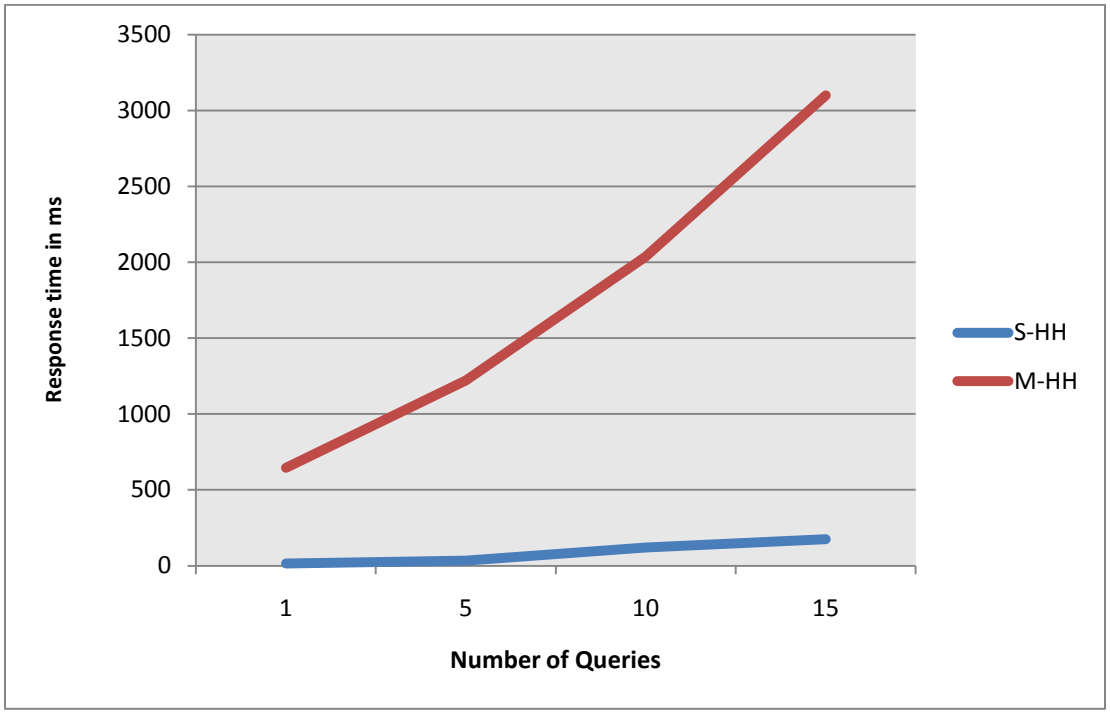


Figure 4.7 Scalability w.r.t several queries at one HH

It is seen, from Figure 4.7, that for a S-HH for a queue size of 100, the queries start dropping at 500 i.e. the processing rate of queries is comparable to the arrival rate of incoming queries. However, for a M-HH, however, it is close to 20 i.e. for a queue size of 10. This is due to the limited processing capability of the PDA due to which it cannot process the queries at a rate faster than the arrival rate. As a result, it cannot handle the number of queries over a certain value and the buffer size becomes full.

4.1.4.3. With respect to the number of HHs

This study was carried out to check if the response time varies if the number of HHs is increased in the system. If that would be the case, then increasing the number of headhunters would guarantee a better response time and the overall system performance would also be better.

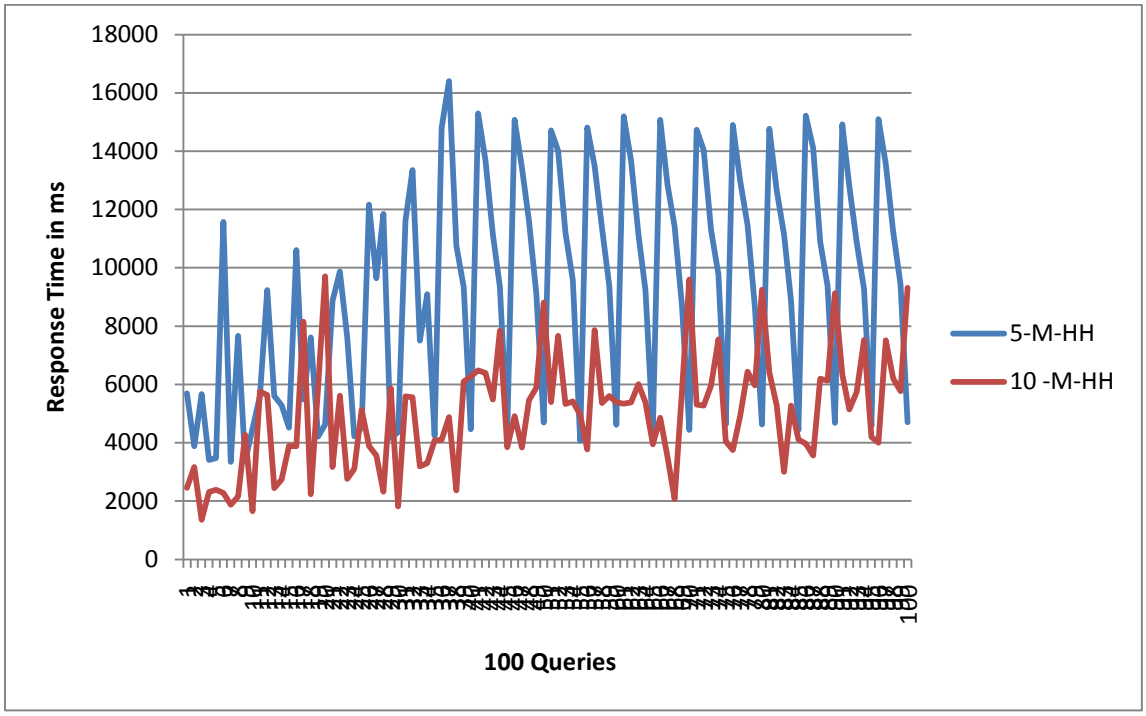


Figure 4.8 Increasing number of M-HHs in the system

The Figure 4.8 shows the response time when the mobile HHs are queried in a round-robin fashion for a selected number of queries. For M-HHs, when the number is 5 in the system, the response time is higher than when the number is 10. It is seen that for most of the queries, the response time is better when there are more resources (in terms of HHs) in the system.

4.2. Enhancements to the MDE-URDS architecture

This set of experiments is performed on the architectures of MDE-URDS with multi-level matching, MDE-URDS with buffering and caching mechanisms and MDE-URDS with different querying mechanisms such as selective, exhaustive searches that are focused towards improving response times, explained in Sections 3.2.1, 3.2.2 and 3.2.3 of Chapter 3.

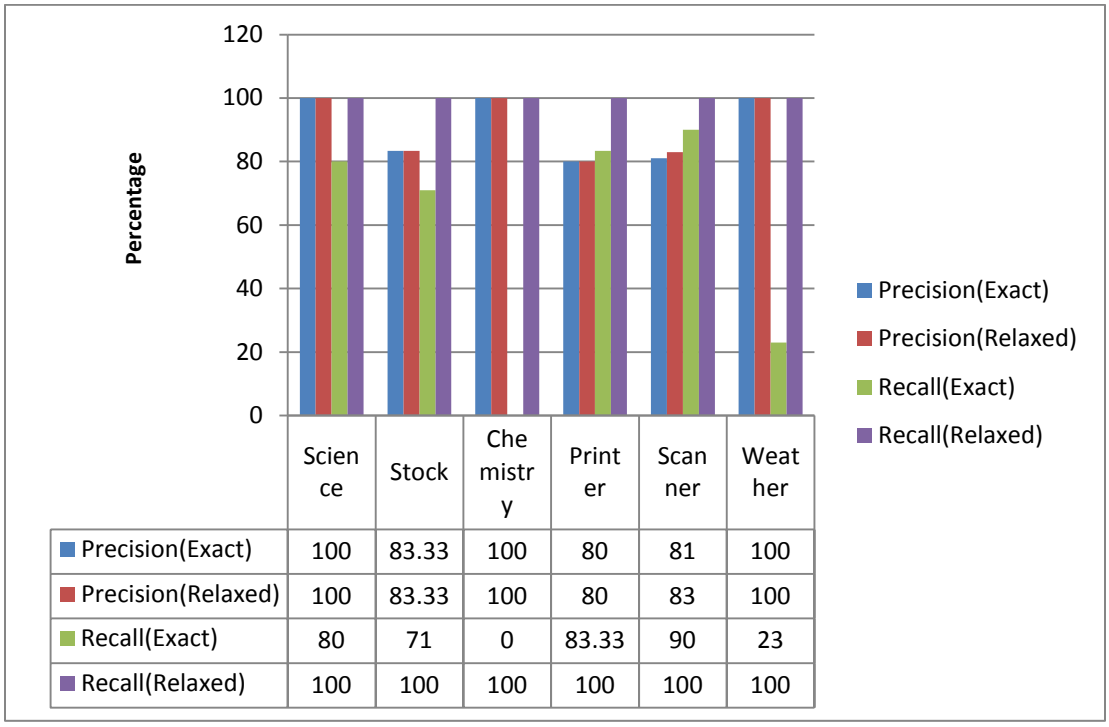
As seen from previous Sections 4.1.2 and 4.1.3, mobile headhunters take more time for returning a response compared to a stationary headhunter. The enhanced MDE-

URDS architectures explained in Sections 3.2.2 and 3.2.3 are an attempt to improve the response time and the quality of results of the mobile headhunters using different techniques of buffering, caching of results, domain specific and exhaustive search mechanisms. The following set of experiments describes the performance of these architectures.

4.2.1. MDE-URDS with multi-level matching (MLM)

This set of experiments focuses on the quality of the results obtained from the headhunters of the basic architecture of the MDE-URDS with multi-level matching scheme with the use of a proxy. It is explained in Section 3.2.1. The results obtained from the HHs are not only tested for their response times but also for their quality. The quality metrics used are the precision and recall [48] of the results as explained in Chapter 3.

For multi-level matching, the mobile headhunters were queried using different semantics (exact and relaxed) for the multi-level matching, as explained in Chapter 3 and the precision and recall were calculated for the results obtained. This first experiment included only type matching of the results. This was to study the difference in quality (precision and recall) of the results obtained when exact and relaxed match semantics are used.



	Science	Stock	Chemistry	Printer	Scanner	Weather
Precision(Exact)	100	83.33	100	80	81	100
Precision(Relaxed)	100	83.33	100	80	83	100
Recall(Exact)	80	71	0	83.33	90	23
Recall(Relaxed)	100	100	100	100	100	100

Figure 4.9 Precision and Recall calculation for a set of queries

Figure 4.9 indicates that with relaxed match semantics, the recall improves whereas the precision may or may not improve (depending on the user's requirements), as every application may have different requirements with the services obtained as results

Experiments were also conducted to observe the improvement of quality in terms of better precision while considering matching with multiple levels of type, syntax, semantics and QoS. With every level, the matching becomes more accurate as it matches more of the user's requirements. However, the response time for evaluation of these multiple levels is high. There is thus a tradeoff between obtaining the best quality services and obtaining a quick response. It is seen that increasing levels of matching improves precision and relaxing the matching criteria improves the recall at the cost of matching time.

For such MLM queries, the precision and recall values were evaluated when type and QoS matching were used; the results of which are shown in Figure 4.10.

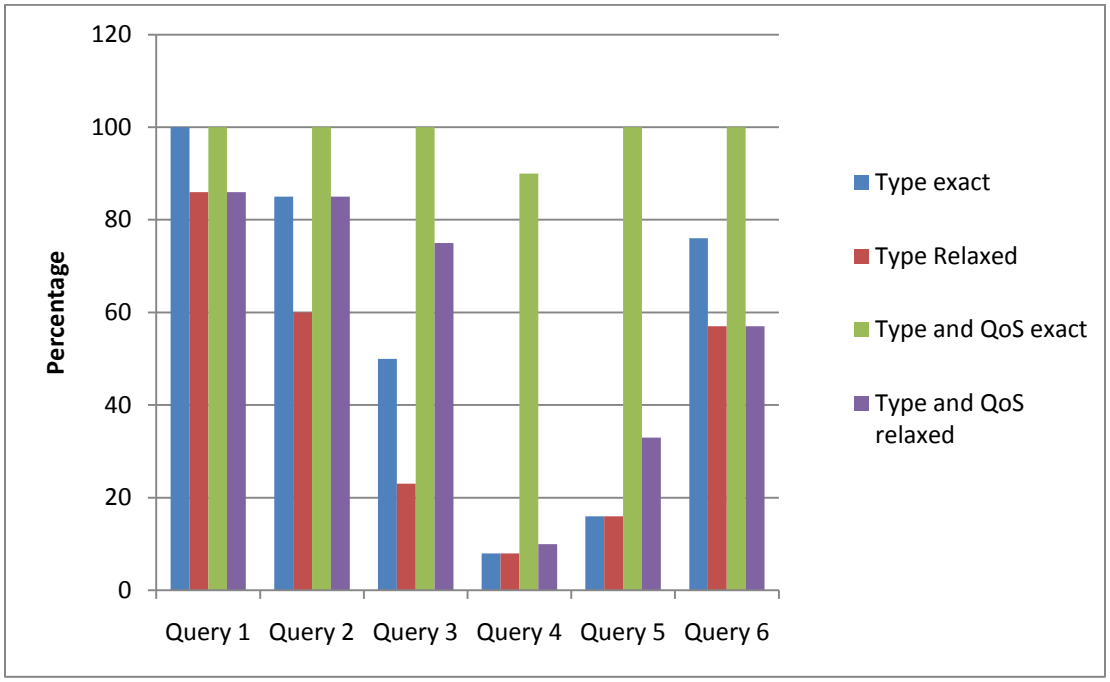


Figure 4.10 Precision of results with different levels of matching

From Figure 4.10, it can be concluded that with an exact type and QoS match, the precision is better as compared to other approaches. This is because exact type match have stricter semantics than relaxed matching and gives more relevant results. Type only matching has low precision and relaxed match also does not improve the precision for this approach. However, the recall of type matching becomes better with type only and relaxed matching, as relaxed matching also includes results that are obtained from synonym, coercion and inheritance based type matching.

However, with increasing the levels of matching and relaxing the matching criteria, the query processing time increases. Figure 4.11 shows the increase in matching time (processing time) for a query that uses only one level of match (type) when compared with using type and QoS match.

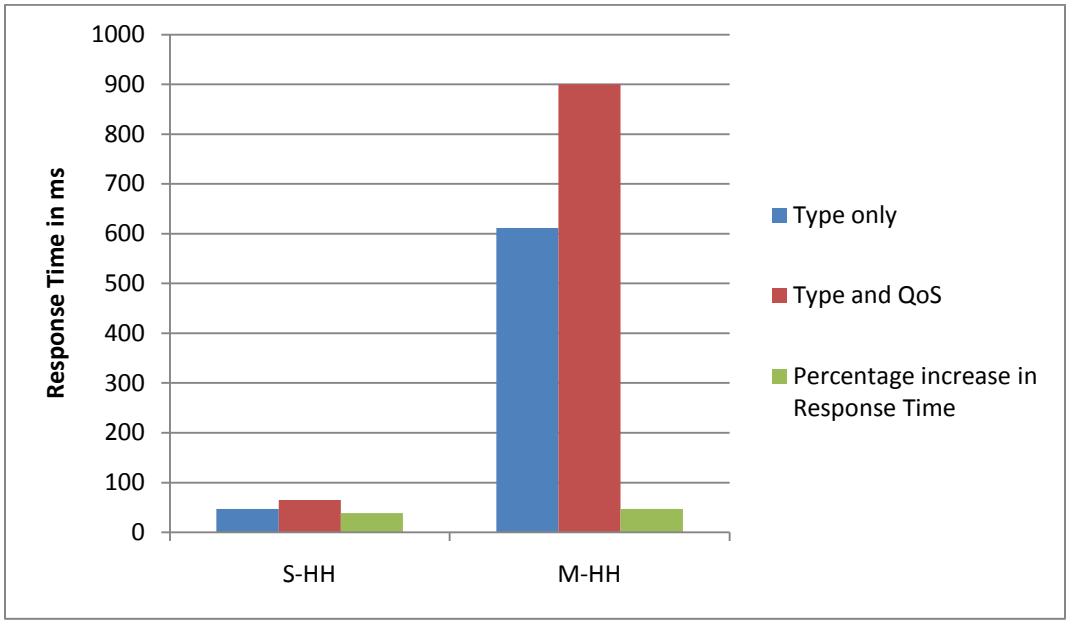


Figure 4.11 Comparison of Response time for Type only and MLM query

It is seen that the response time is more as the levels of matching increase. For a S-HH this increase in time is around 38% and for a M-HH, it is close to 47%.

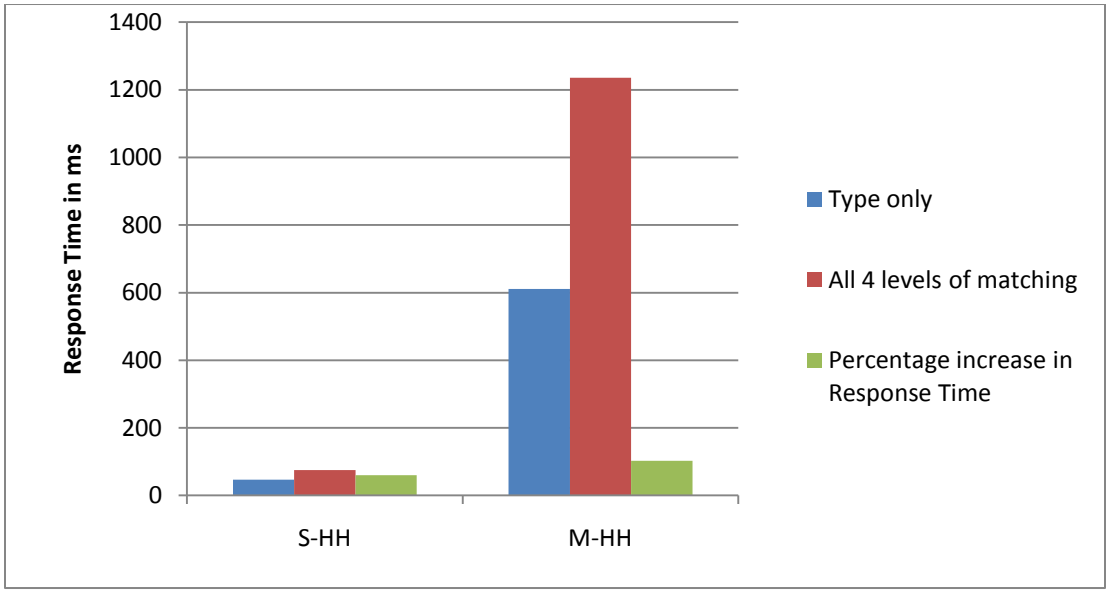


Figure 4.12 Comparison of Response time for Type only and all Four levels of Matching

For all levels of matching, the response time is very high, but the precision is also the highest. It is 1 for the set of queries used in the above experiments (Figure 4.12).

4.2.2. MDE-URDS with caching and Buffering mechanisms (To improve the response time (especially with multi-level matching))

This experiment is performed on the architecture suggested in Section 3.2.2. The results from Sections 4.1.3 and 4.1.4 show that since RMI communication is used, the response time increases of the system by mobile headhunters. To improve this response time, the use of buffering mechanisms is suggested at the HH. The buffering/caching is of the recent query results of the HH. The least recently used (LRU) policy is used for maintaining the HH buffer.

It is seen, from Figure 4.13, that the buffering improves the response time considerably, as expected.

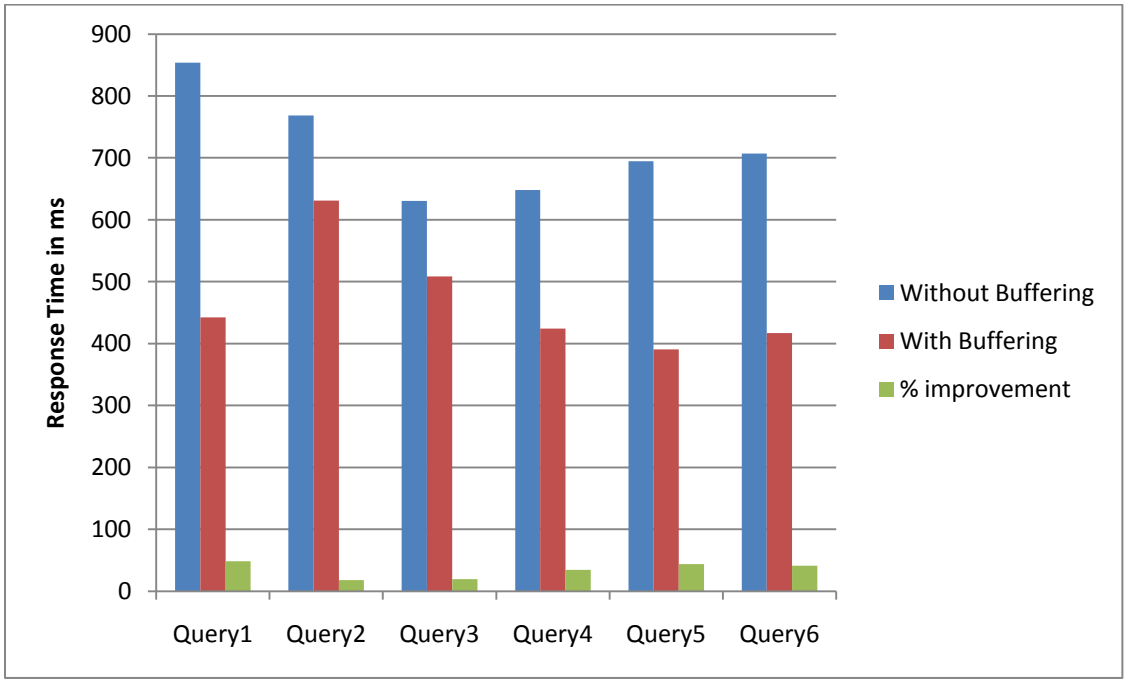
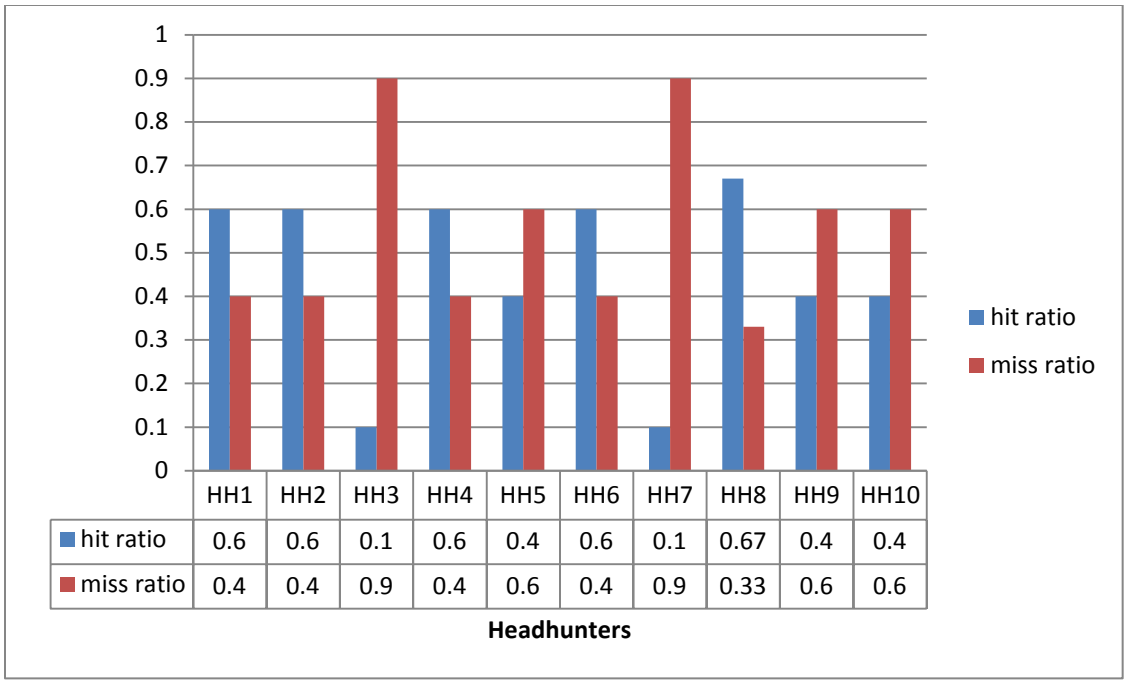


Figure 4.13 Comparison of response times with and without buffering of components

Figure 4.13 shows the improvement in the response time when buffering/caching is used. However, the hit/miss ratio is a major concern for caching mechanisms. It is seen that for the LRU policy used, the hit ratio (Figure 4.14) is more than the case where no replacement policy, as the HH maintains the results of the most recently queried services Figure 4.14 shows the hit ratio for a number of HHs individually.



	HH1	HH2	HH3	HH4	HH5	HH6	HH7	HH8	HH9	HH10
hit ratio	0.6	0.6	0.1	0.6	0.4	0.6	0.1	0.67	0.4	0.4
miss ratio	0.4	0.4	0.9	0.4	0.6	0.4	0.9	0.33	0.6	0.6

Figure 4.14 Hit and Miss ratio for different HHs

The Figure 4.14 shows the hit/miss ratio calculated for a random set of queries with an uniform distribution of services amongst the HHs. The hit/miss ratio is different for different headhunters depending on the distribution of components and other factors such as the frequency of querying that headhunter.

The recall is calculated for a specific query (Weather domain) to check the effectiveness of buffering. Again, depending on the distribution of components among the headhunters, the recall varies, here uniform distribution is assumed.

Table 4.1 Recall for the 10 M-HHs

Query=Weather	Recall
HH1	0.2
HH2	1
HH3	1
HH4	0.66
HH5	0
HH6	0.4
HH7	0
HH8	0.25
HH9	0
HH10	0

4.2.3. MDE-URDS with different querying methods

The next two sets of experiments are performed on the MDE-URDS architecture with different querying methods, described in Section 3.2.3 of Chapter 3. The purpose of performing these experiments is to observe if the performance (response time and quality) of the system improves using different querying mechanisms. This task is achieved by comparing the different search approaches.

The three main approaches selected were as follows:

a. Selective Search (domain specific)
b. Exhaustive Search
c. Random selection of HHs (random 3).

The results of all the three approaches are presented in the following Figures (4.15 to 4.18), that show that when time is a limitation, the selective or random approach is better, with some compromise on quality; whereas when time is not an issue and the major focus is on obtaining the most relevant services, then the Exhaustive Search approach is better. The random approach performs comparably well at some times, while at other times, it reaches the time taken by the exhaustive approach.

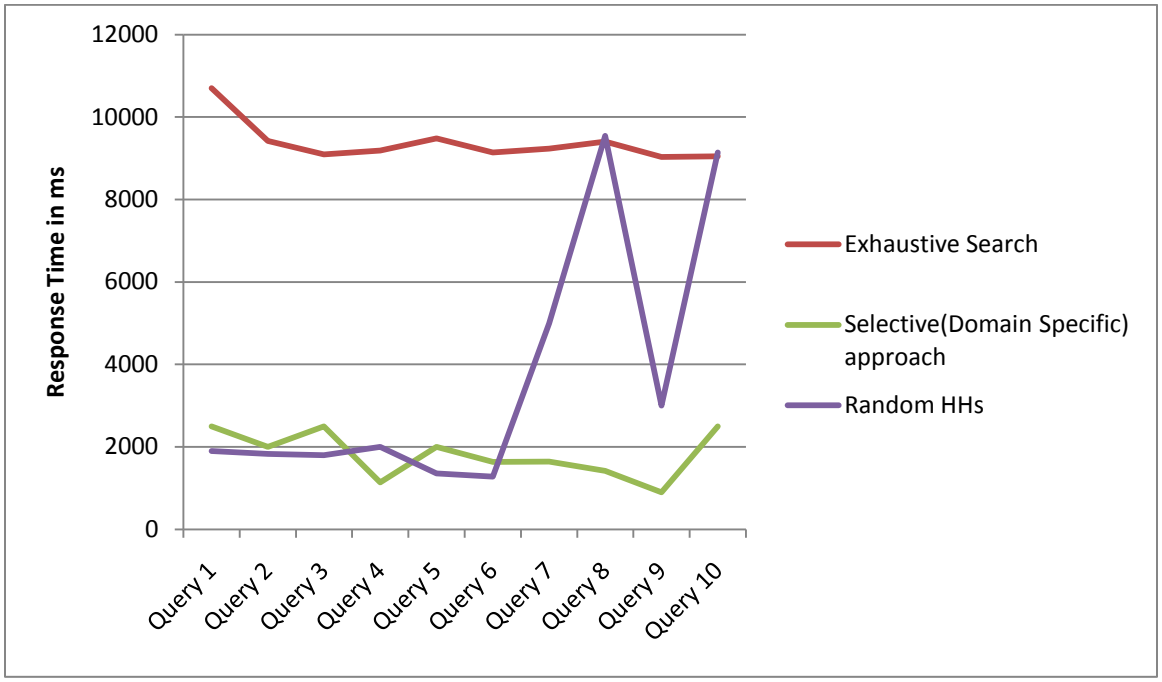


Figure 4.15 Comparison of Response Time for the three different search approaches

The above approaches are also evaluated for the quality of the results obtained using them. Figure 4.16 shows the recall values for the results for different queries.

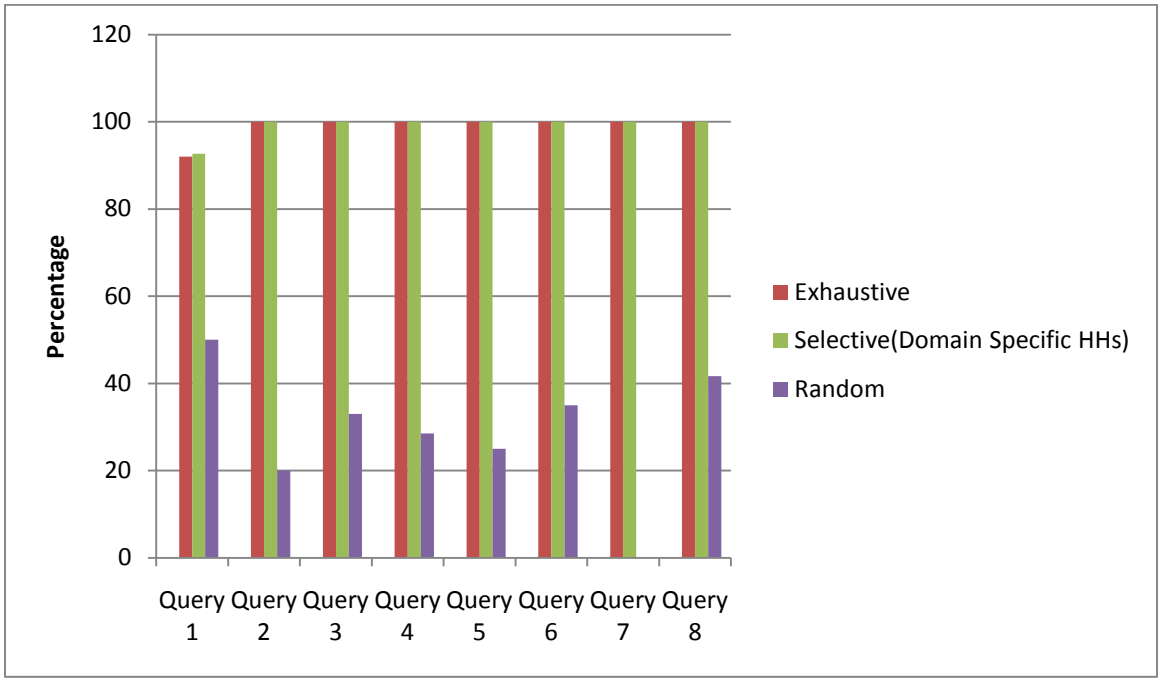


Figure 4.16 Comparison of quality (Recall) for the three different search approaches

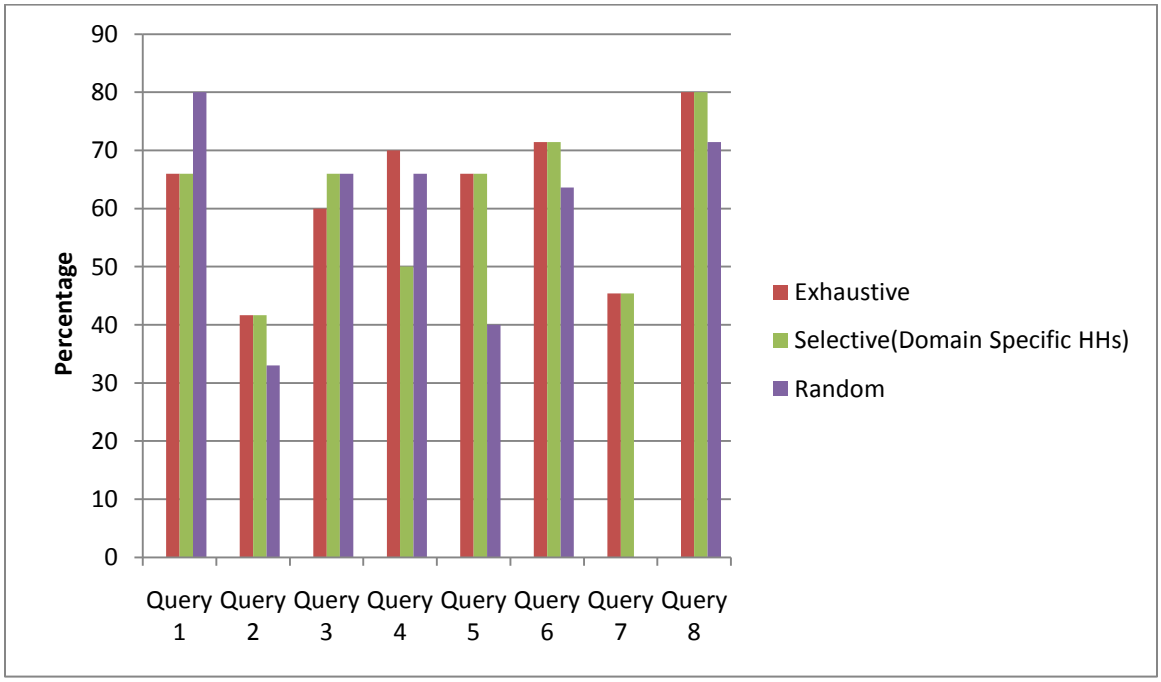


Figure 4.17 Comparison of quality (Precision) for the three different search approaches

From Figures 4.16 and 4.17, it is seen that the recall value for various queries in the domain specific approach is 1 most of the time. The random approach has a lesser value of recall amongst the three cases because of the random nature of selecting HHs as a result of which only a few relevant services can be obtained. Exhaustive approach has a recall of more than 0.5 in most cases. In terms of precision, the domain specific approach and exhaustive approach has high precision being more than 0.5 in most cases. Random approach sometimes gives high precision (Query 1) whereas sometimes gives zero precision (Query 7). This is due to the HHs queried and the matching components retrieved from them.

4.2.3.1. To check for the quality of results for a Timed Response

For some applications such as real-time tracking, the discovery time needs to be as small as possible, as these applications demand an end-to-end response time of 30 ms or less. For similar applications having time bounds, the performance of the MDE-URDS architecture with different querying methods (described in Section 3.2.3) such as selective, exhaustive search is evaluated to check if it can be suitable for these applications and how the MDE-URDS can be modified, if need be. For this purpose, a time limit was set up and the quality of the results in terms of precision and recall was calculated for a set of queries. The time limit can differ according to the client's or application's requirements. For the purpose of experimentation, it was set it to 5 seconds (5000 ms).

The Figures 4.18 and 4.19 show the quality of results obtained from a timed response scenario versus a scenario without any time bound.

Figure 4.18 Difference in quality (Precision) of results with a timed response

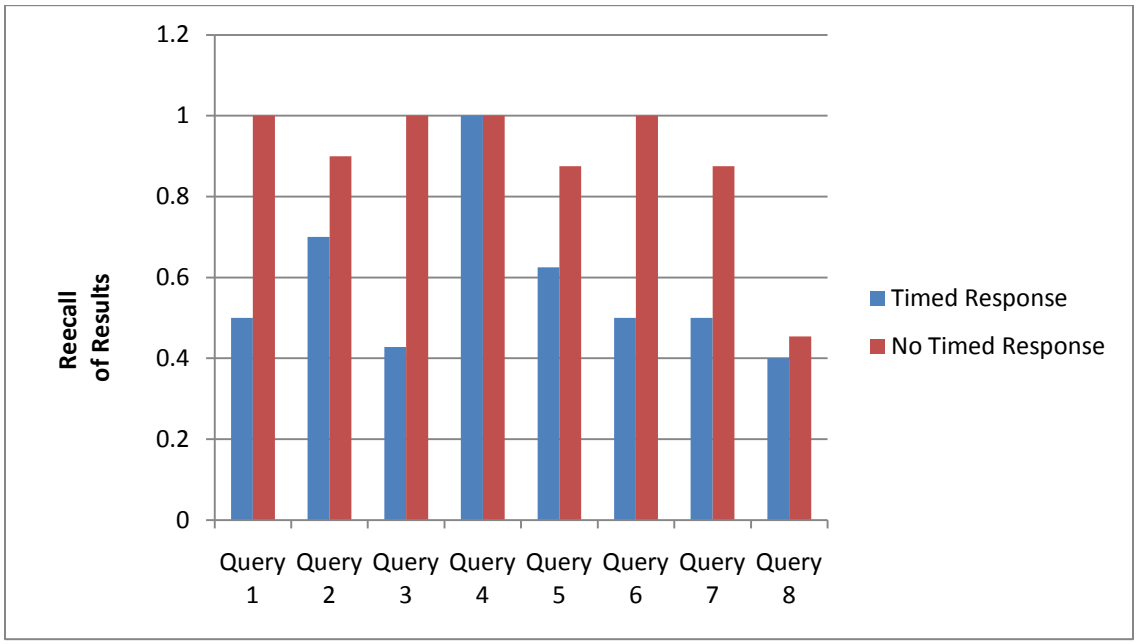


Figure 4.19 Difference in quality (Recall) of results with a timed response

It is evident from Figures 4.18 and 4.19 that with the timed response scenario the number of matching services obtained are less than the one when there are no time bounds. The recall of the latter approach is thus high. The precision is relative to the HHs selected and thus is not constant for either of the approaches. This shows that given no time bounds, the system can obtain better results whereas in case of time critical or real-time applications, obtaining more and best services may not be possible. Also, the time limit is decided by the client who can vary it according to his needs and this will affect the overall recall of the system.

4.2.4. MDE-URDS with Mobile IP (Incorporation of Mobility)

This set of experiments is the study of the performance of the implementation of Mobile IP on the MDE-URDS architecture explained in Section 3.2.4.

The configuration for this set of experiments is as follows: 10 M-HHs, 10 ARs, 100 services, 3 Foreign Worlds, MLM matching and Proxies for the HHs. The readings are an average over 10 readings and noted for 10 different queries.

4.2.4.1. Headhunters in their home domain

As seen from the algorithms described in Section 3.2.4 of Chapter 3, a headhunter registers with its Home Agent when it starts. It remains in its home domain unless it wants to move to another domain. The time taken to query these headhunters when they are still in their home domain is calculated to get an idea of the response time of the mobile headhunters in this implementation.

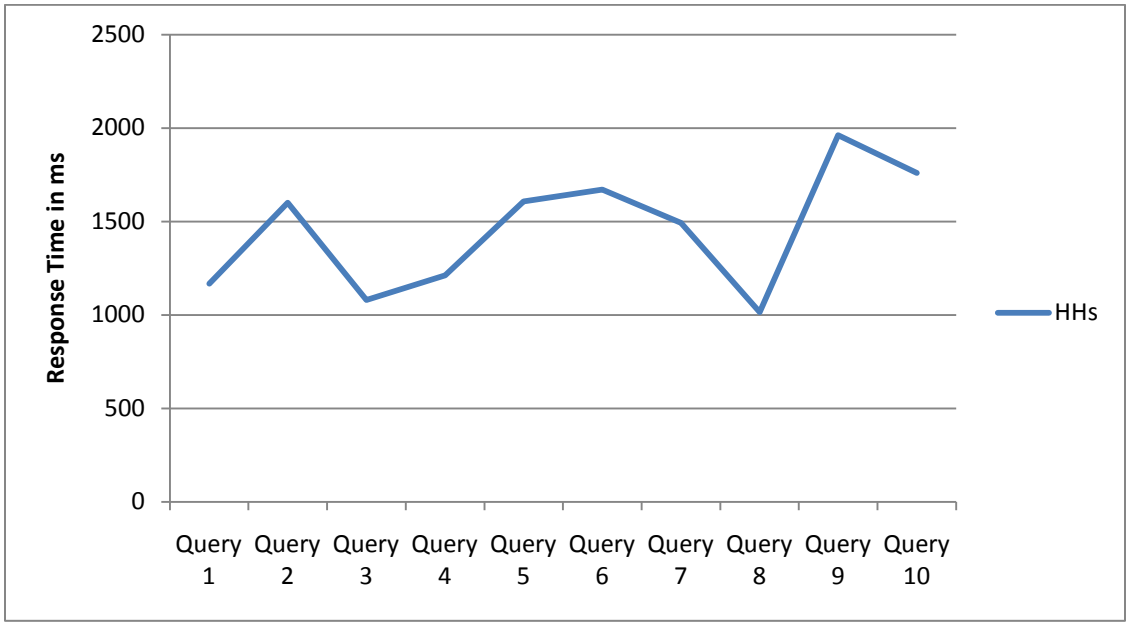


Figure 4.20 Response time of HHs when they are in their home domain

The Figure 4.20 shows the response times from the HHs while they are present in their home world. The response time varies between 1000-1600 ms. Also, if these headhunters are searched exhaustively for a particular query, then the average response time is around 4000 ms. This is shown in Figure 4.21.

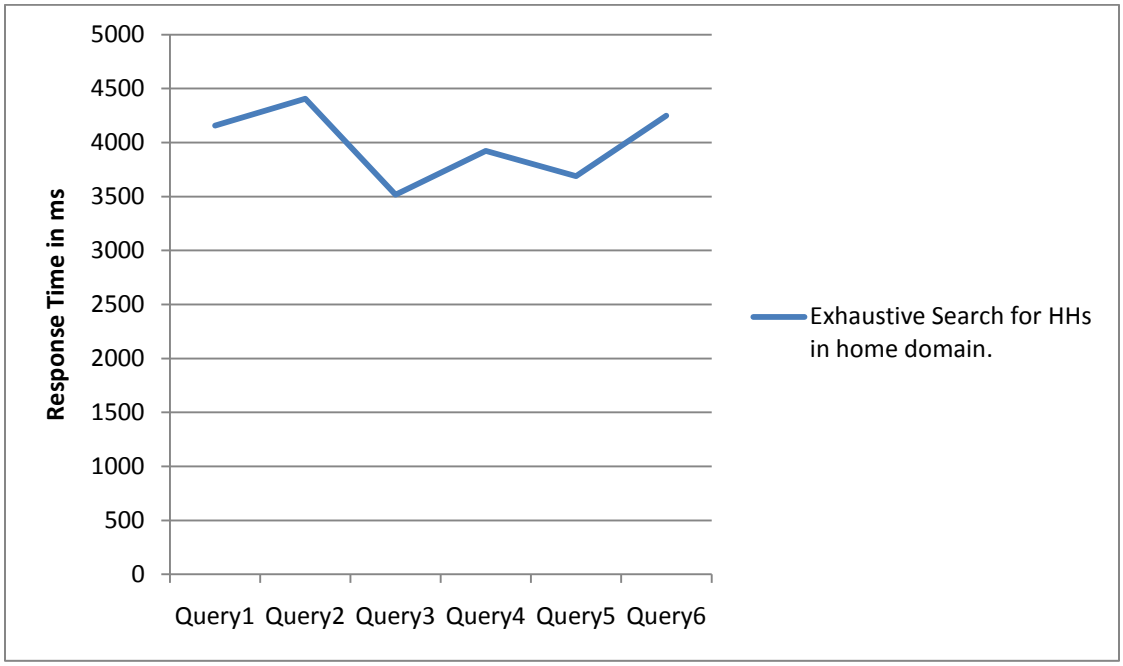


Figure 4.21 Exhaustive search Response time of HHs when they are in their home domain

4.2.4.2. Headhunters in the Foreign Domain

The response times from all headhunters who are currently in a foreign world are observed and the additional time taken due to redirection of the queries is also studied.

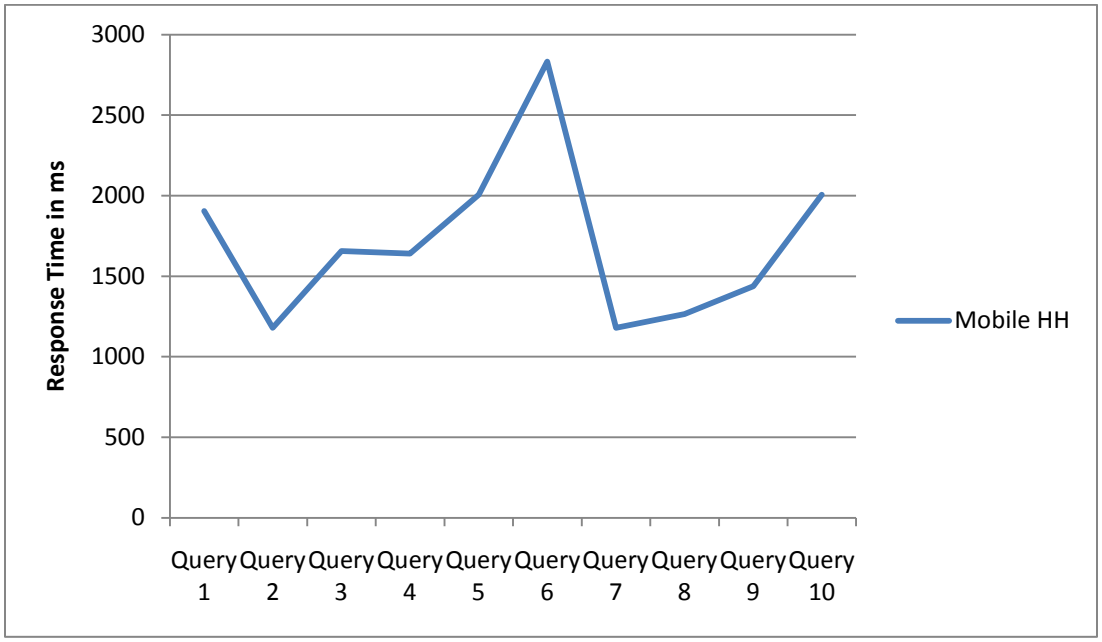


Figure 4.22 Response time from headhunters in a foreign domain

From Figure 4.22, it is seen that the response times for mobile headhunters registered with a Foreign Agent are higher due to the level of indirection associated to passing a query first to the headhunter's Home Agent. The query is then routed by the Home Agent to the Foreign Agent and then to the HH. As a result, the response time is higher. The typical response time is between 1500-2200 ms, with some queries having higher response times.

4.2.4.3. Headhunters in transit

The headhunters are said to be in transit when they are not connected to any domain or are unreachable. In such a case, the queries are either:

a. Buffered at the Home Agent and are sent to the HH when it gets connected eventually
b. Passed to another HH.

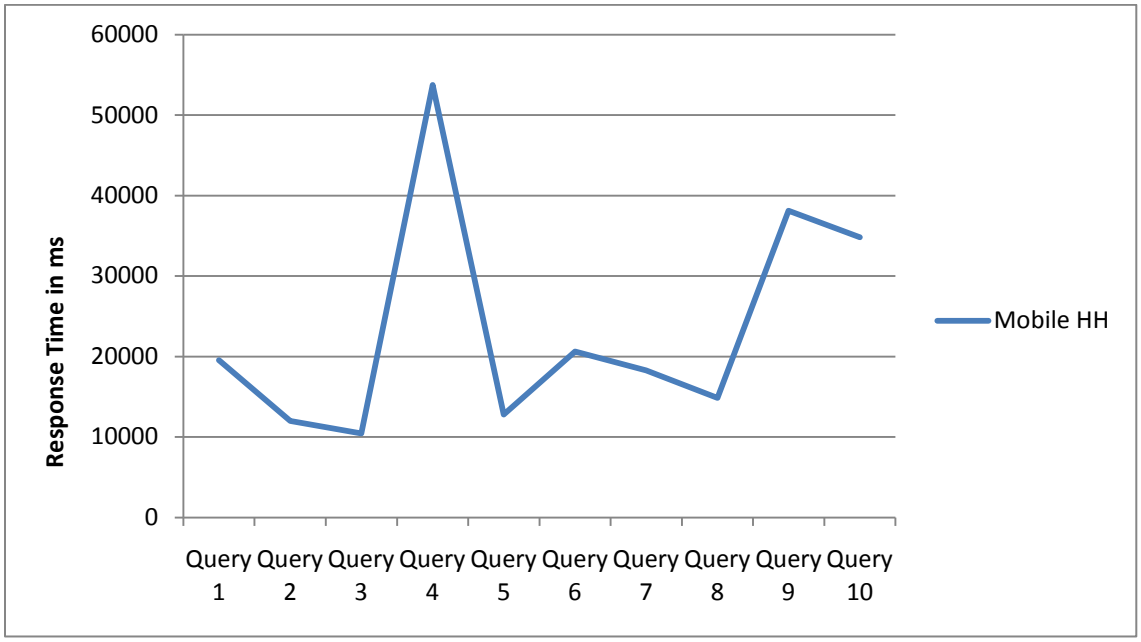


Figure 4.23 Response time calculation when headhunters are in transit and waiting for them

From the Figure 4.23, it can be seen that the response time varies according to the time taken by the headhunter to connect back to a new domain. It could be in a matter of few seconds or may even take few minutes. In such cases, wherein the time is a constraint, it is better to propagate the query to a nearby headhunter (option (b)) and obtain the results, however if the headhunter is domain specific or critical to the query and the application can afford to wait for a long response time, then the higher response time is acceptable.

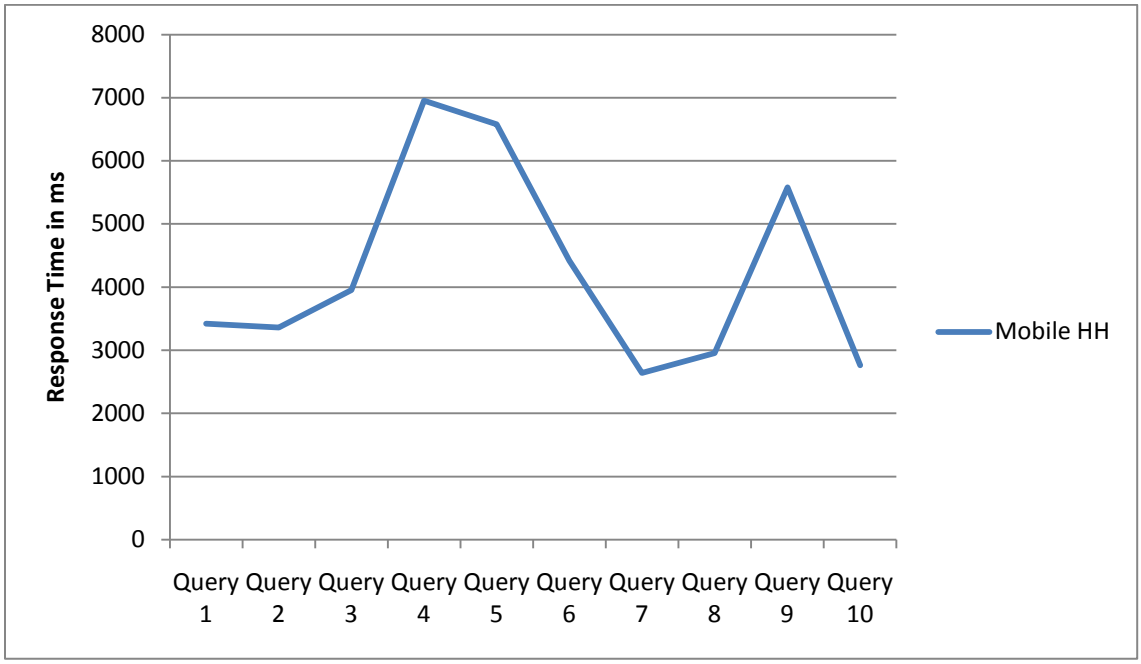


Figure 4.24 Response time calculation when headhunters are in transit and getting response from other headhunters

The Figure 4.24 shows that in case of applications where the time is a constraint, the query is propagated to another headhunter, if the current one is unavailable. This time varies according to finding the next available headhunter and also the domain where it resides (foreign domain will take more time, etc.). The peaks shown in the Figure 4.24 depict such scenarios.

For the quality evaluation of the queries for scenario 4.3.3 (a) and 4.3.3 (b) of Section 4.3, when both the headhunters are similar in their distribution of the services (i.e., uniform distribution) then the quality of the results remain more or less similar whereas when they differ in the distribution (e.g., domain specific HHs) then the quality suffers as the precision of the results reduces.

4.2.4.4. Response time increase due to Mobile IP

The increase in response time due to the implementation of Mobile IP is studied. The response time is now more with respect to time to first contact the HH's Home Agent, then Foreign Agent and then the HH itself. Every reading for a response time is evaluated for individual times and is shown in Figure 4.25. The time needed for a response from a Foreign Agent was also evaluated in terms of its constituents indicating which part dominates in that case (Figure 4.25).

These experiments are not evaluated for quality as the precision and recall for this category is similar to that of experiments of Section 4.1.

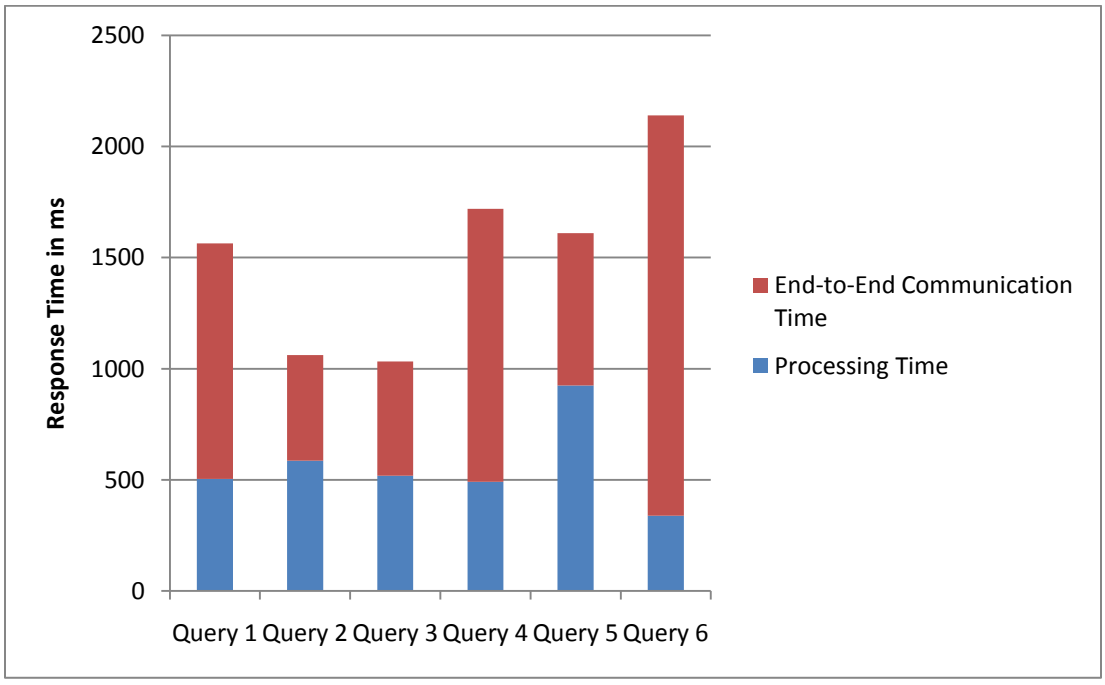


Figure 4.25 Evaluation of response time from headhunter

It can be seen, from Figure 4.25, that with the use of Foreign Agent, an additional RMI request is added which adds to the overall response time. The End-to-End communication time therefore dominates the total time.

4.2.4.5. Scalability of the system

The scalability of the system is performed to check the maximum limit on the number of queries the MDE-URDS architecture including Mobile IP can handle and how frequent the headhunters can move around.

a. **Number of queries:** This is similar to the study in Section 4.1. However, in this case, this study is for identifying the number of queries the mobile headhunters can handle. It is seen that for the 10 mobile headhunters present in the system, the limit on the number of queries is 200, i.e., as the number of queries in the system becomes more than 200, the queries start dropping.

b. **Effect of Buffering on response time:** As seen in Section 4.2, for mobile headhunters, the buffering of results improves response time by almost 48%. Thus, buffering can benefit the mobile headhunters that move around often; however, their mobility still increases the response time to some extent due to the redirection of queries and is therefore more than in Section 4.4.1.

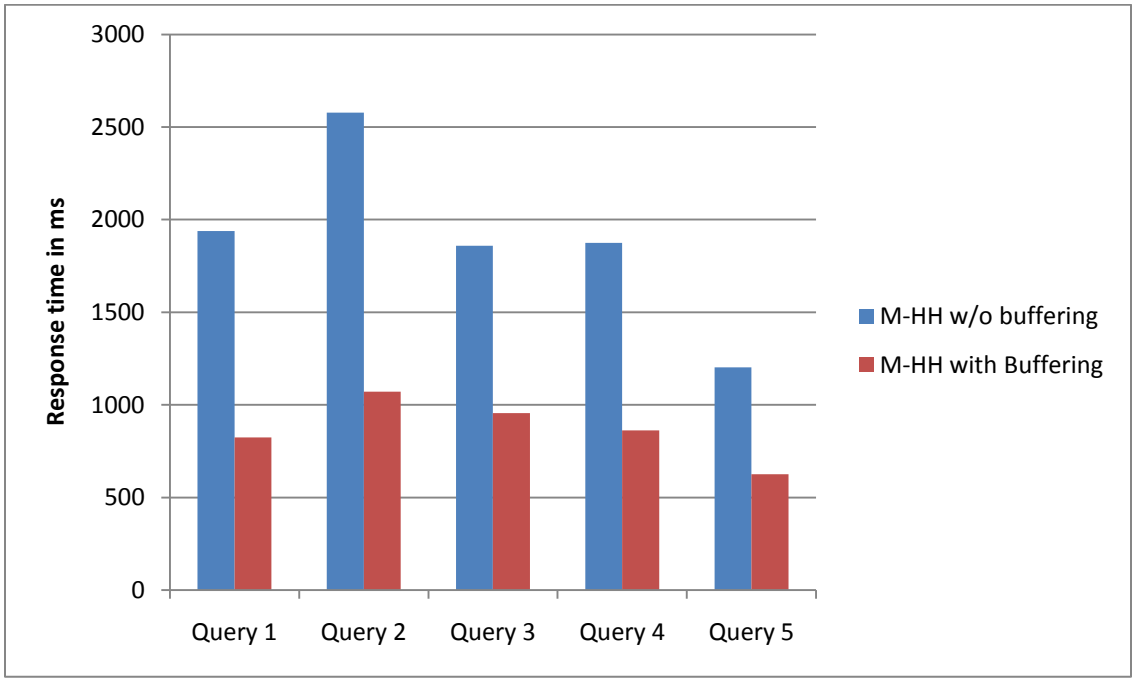


Figure 4.26 Response time variation for M-HHs w.r.t Buffering of Results

Figure 4.26 shows the improvement in response time for mobile headhunters using buffering mechanisms which is almost 50%.

4.2.5. MDE-URDS with Collaborative Approaches: I do it, We do it, You do it

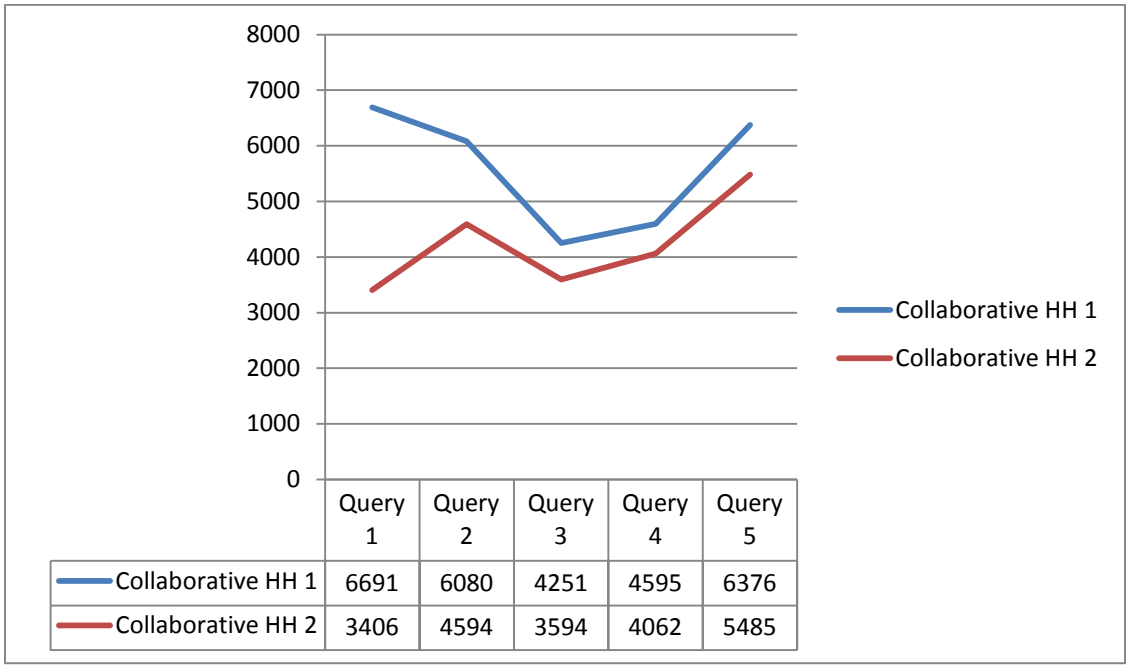
As described in the Chapter 3, in order to improve the quality of results the MDE-URDS architecture was enhanced using MANET protocols. This set of experiments studies the performance of this MDE-URDS architecture with collaborative approaches, described in Section 3.2.5.

4.2.5.1. I do it approach

As this is actually not a collaborative approach, as explained in Section 3.2.5, the results are similar to the scenarios explained in category 4.3 when the headhunters are in their home world and the results obtained were similar to them.

4.2.5.2. We do it approach

The results for the “We do it” approach using AODV technique is presented in the Figure 4.27.



	Query 1	Query 2	Query 3	Query 4	Query 5
Collaborative HH 1	6691	6080	4251	4595	6376
Collaborative HH 2	3406	4594	3594	4062	5485

Figure 4.27 We do it approach using AODV protocol

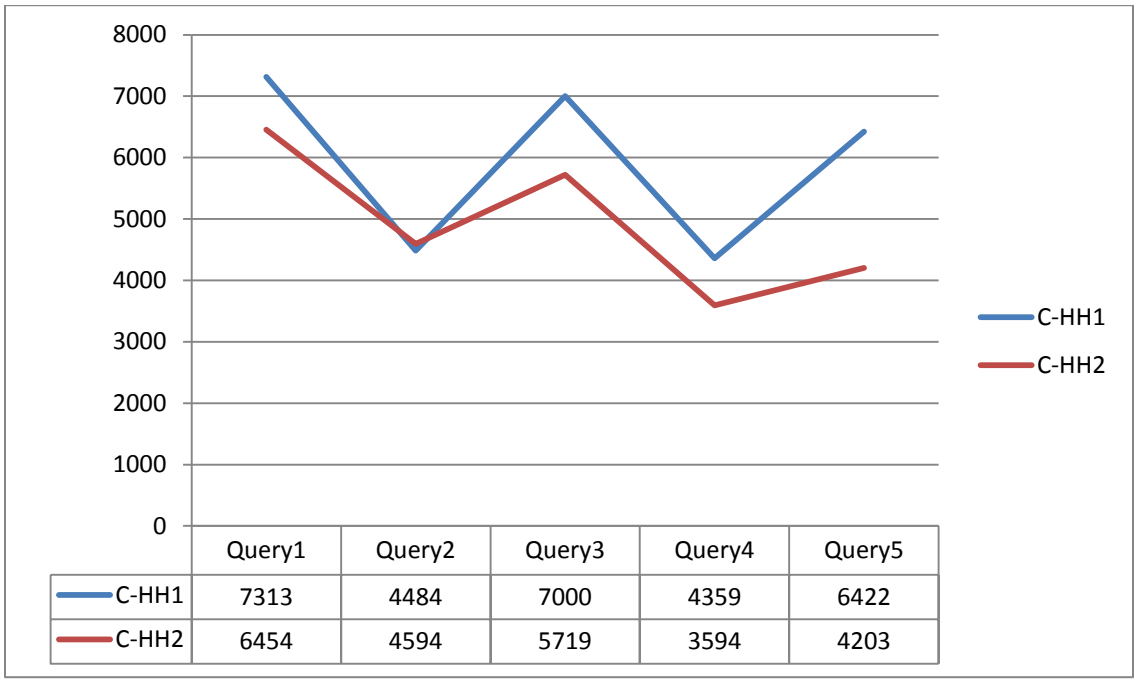
Table 4.2 Precision and Recall for We do it approach

Query	Number of Services from single HH		Number of Services with collaboration	
	Precision	Recall	Precision	Recall
Scanner	0	0	0.5	0.4
Health	0	0	0.6	0.833
Chemistry	0.5	0.2	0.3	0.4
Tracking	1	0.125	0.75	0.375
Science	1	0.166	0.727	0.66

The Figure 4.27 shows the response time for the collaborative HHs using the AODV protocol. It is seen that the response time is high compared to querying a single HH, because the results are obtained from 2 neighbors plus from the queried HH and also time is spent in accumulating these individual results.

The Table 4.2 illustrates the precision and recall when the query is processed by a single HH and when collaboration occurs. As the number of services is more in the latter case and thus the recall of the system is better. The precision varies according to the headhunter selected, but is better in the second approach as seen.

For the proactive approach, as described in Section 3.2.5 using the OLSR protocol, the results are presented in Figure 4.28.



	Query1	Query2	Query3	Query4	Query5
C-HH1	7313	4484	7000	4359	6422
C-HH2	6454	4594	5719	3594	4203

Figure 4.28 Collaborative results for We do it approach using OLSR protocol

Table 4.3 Precision and Recall comparison for We do it approach using OLSR protocol

Query	Number of Services from single HH		Number of Services with collaboration	
	Precision	Recall	Precision	Recall
Scanner	0	0	0.5	0.4
Health	0	0	1	0.66
Chemistry	0.5	0.2	0.25	0.2
Tracking	1	0.125	1	0.375
Science	1	0.166	0.66	0.5

Here, similar response times and quality of results obtained as the AODV approach in Figure 4.28 are seen, only this approach is more reactive and increases the network traffic due to constant sending of messages. It however avoids any exceptions as all nodes have up-to-date information about other nodes. Table 4.3 shows the precision and recall of the approach. It is seen as the previous case of 4.6.1 that the precision varies according to the HHs queried but the recall is better in case of collaboration.

4.2.5.3. <u>You do it approach</u>

This study is performed to check if a HH is busy with other tasks and needs help with the query processing, how well the delegation of the query processing to another HH can take place. The Figure 4.29 shows that with delegation of a query, the response time increases however, the quality of results are better.

When comparing the previous approach of “We do it”, the recall of this method is less as seen from the obtained results and it maybe because it does not include the headhunter’s own response. As a result, the response time is less.

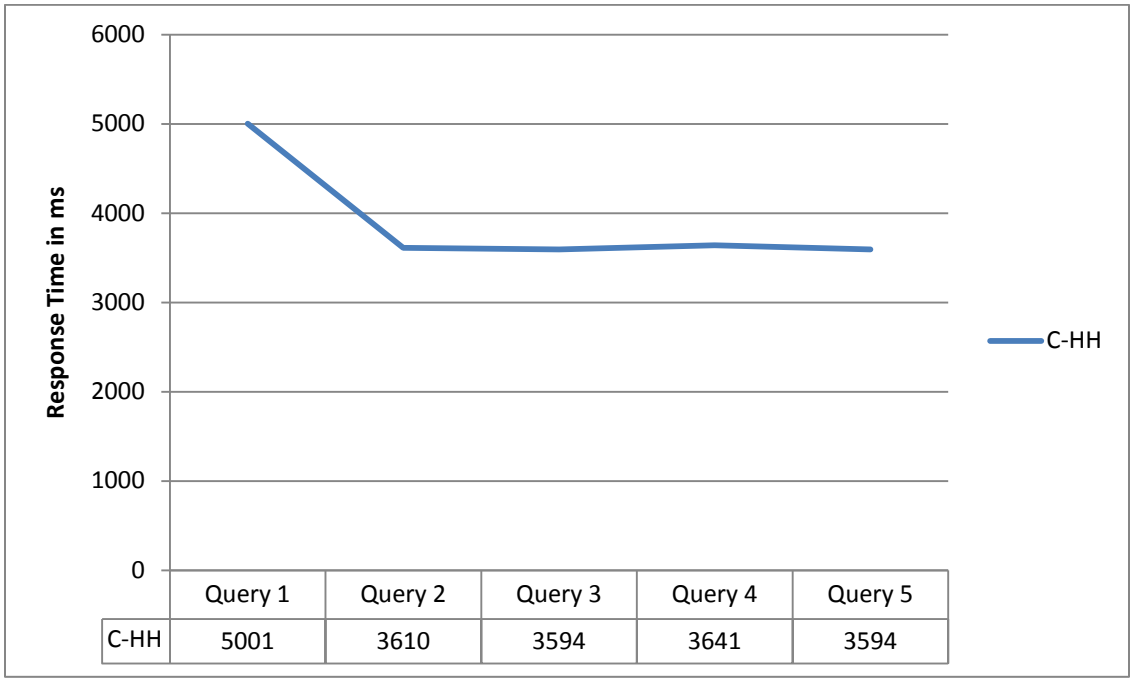


Figure 4.29 Collaborative results for You do it approach

Table 4.4 Precision and Recall comparison for You do it approach

Query	Number of Services from single HH		Number of Services with collaboration	
	Precision	Recall	Precision	Recall
Scanner	0	0	0.5	0.4
Health	0	0	0.75	0.5
Chemistry	0.5	0.2	0.5	0.4
Tracking	1	0.125	0.66	0.25
Science	1	0.16	0.66	0.5

The Table 4.4 shows that the precision and recall of the approach. As more number of HHs now process the query, the recall improves. Precision may or may not improve depending on the HHs queried and the distribution of components. In Table 4.4, it can be seen that the precision is some cases is better in collaboration (Scanner, Health)

whereas it degrades in some cases (Science, Tracking). The recall however is always better in case of collaboration.

4.3. Case Study

The case study selected is of a real-time tracking application. The application of MDE-URDS into the domain of tracking is studied for its performance effectiveness.

Real time tracking requires tracking of objects from sensors such as Cameras. These Cameras have different QoS attributes such as Camera Resolution, Frame Rate, Orientation, Relative Location, and Clock Drift(to name a few) that are distinguishing. As a result, these cameras can be treated as services that need to be discovered in order to track an object and multi-level specifications are created for these Camera Services.

An experimental setup of an existing Tracking System of [53][54] was chosen for this purpose. In this setup, around 20 cameras are deployed in a simulated lab environment. Every Camera has a service called as CamService that serves to track the visual data. These cameras track vital object information and then fusion of the visual data from the cameras in the system is performed for object tracking. In order to apply the architecture of MDE-URDS in this tracking system, first multi-level specifications for these camera services are created. These specifications are created using similar semantics of ML specifications of services (shown in Figure 4.1). However, depending on the user's requirements of choosing these services (Frame Rate, Resolution, Orientation, Clock Drift); they are discovered using the discovery architecture of MDE-URDS with MLM.

For this setup, experiments for discovering these Camera Services were performed using different ML queries (similar to Section 4.1.4) and the average response time was found to be around 1000 ms. The discovery is followed by fusion of the results that takes around 35-55 ms. The quality of results obtained by using MDE-URDS with MLM is comparable to the existing discovery service of Jini and in some cases even better according to the experiments performed for searching these Camera Services. This time is however, too high for real-time tracking and as a result, it may not be suitable for tracking in real-time. It can however be applied to mobile tracking with the use of mobile

devices and offline tracking when there are no strict time limitations. As only those services that meet the criteria are selected, the filtering of unwanted results becomes simpler and the fusion is also faster.

Thus, MDE-URDS can be applied successfully in applications such as real-time tracking as a MLM discovery service. Thus, the performance evaluations for the different enhancements to the architecture of MDE-URDS have been categorically studied. These studies show the current performance of the system and highlight some of the limitations of the architecture still present and that will be a part of the future work. Chapter 5 includes the conclusions and future work of MDE-URDS in detail.

CHAPTER 5. CONCLUSIONS AND FUTURE WORK

Chapters 3 and 4 gave an insight on the MDE-URDS architecture and its performance related experiments. This chapter concludes the experiments with conclusions and the future work of the MDE-URDS architecture.

The MDE-URDS architecture tries to tackle the problem of including mobile devices into the service discovery framework that were not considered initially in the first generation of discovery systems. The limitations of intermittent connectivity, memory and processing limitations and mobility have been tried to address by the suggested architecture. From the experimental validation, it is seen that the architecture performs well for obtaining appropriate results for services in terms of response time and quality. It also uses different searching techniques such as selective and exhaustive search and certain collaborative searching techniques using the principles from different MANET protocols such as AODV, OLSR and DSR, in order to improve the quality of the results. The response time obtained is high due to the memory and processing limitations of the device and the wireless connectivity, but the quality of the results in terms of precision and recall is good due to the support of multi-level matching and alternative searching techniques. For real-time applications like tracking, it is seen that the response time is too high for discovery using mobile headhunters and therefore it cannot be applied to real-time data. However, if timing restrictions are not strict, then, the MDE-URDS can serve as a useful discovery tool for finding appropriate services and filtering unwanted ones by using MLM of specifications.

One of the other limitations of mobile devices is the battery life that was not a part of this MDE-URDS architecture. It can serve as a potential future work by employing techniques from the MANET domain suggested to preserve the battery life.

The architecture does not involve context as a part of the discovery process, except from the domain specificity of the headhunters. The discovery process can be enhanced further

by using contextual information. This is one of the future works that will be considered. Uncertainty with respect to the context or surroundings is another area of future work that can be studied for MDE-URDS. The performance of MDE-URDS can be evaluated in such situations where the entities in the system and the surroundings are constantly changing and at times are uncertain.

The security mechanism used in the MDE-URDS architecture is basic password-based authentication. More complex security mechanisms using public key certificates, symmetric key encryption can be used to provide additional level of security and prevent unauthorized access to the system. The architecture can be made more heterogeneous using different devices that are capable of running Java components. This may give a real-time feel to the architecture where different devices are present.

Being successfully used in the real-time tracking application, the MDE-URDS performance in other real-time applications such as disaster management can be tested. If it is successful, then it may be useful for such applications.

Thus, the MDE-URDS architecture has successfully incorporated mobile devices into a service discovery framework and has tried to tackle some of the limitations of mobile devices. The future work suggested above will be helpful in making the architecture more comprehensive.

LIST OF REFERENCES

LIST OF REFERENCES

[1] Kozaczynski W., Booch G., "Component-Based Software Engineering," IEEE Software Volume: 155, Sept.-Oct. 1998, pp. 34-36.

[2] Czarnecki K., "Generative Programming: Principles and Techniques of Software Engineering Based on Automated Configuration and Fragment-Based Component Models," Ph.D. Dissertation, Department of Computer Science and Automation, Technical University of Ilmenau, October 1998.

[3] Sun Microsystems, "Jini™ Architectural Overview," http://pages.cs.wisc.edu/~lhl/cs740/papers/jini-overview.ps (accessed on September 25th, 2009).

[4] "Universal plug and Play (UPnP)," http://www.upnp.org (accessed September 25, 2009).

[5] "UDDI Technical White Paper." http://www.uddi.org/ pubs/Iru_UDDI_Technical_White_Paper.pdf (accessed September 25, 2008).

[6] Guttman E., Perkins C., Veizades J., Day M., "Service Location Protocol, Version 2," IETF RFC 2608, 1999.

[7] Dabrowski C., Mills K., Quirolgico S., "A Model-based Analysis of First-Generation Service Discovery Systems," Special Publication 500-260, NIST, National Institute of Standards and Technology, 2005.

[8] Perkins C., "Mobile IP," IEEE Communications Magazine, Volume 35, Number 5, May 1997.

[9] Perkins C., Royer M., "An implementation study of the AODV routing protocol," In Proceedings of the IEEE Wireless Communications and Networking, Chicago, IL, 2000.

[10] Clausen T., Jaqcuet P., Laouiti A., Minet P., Muhlethaler P., Qayyum A., Viennot L., "Optimized link state routing protocol," IETF Draft, 2003.

[11] Raje R., Katuri P., Kumari A., Tilak O., "Multi-level Matching of Distributed Software Components," Proceedings of the International Conference on Computer, Communication and Instrumentation, Mumbai, India, 2009.

[12] Zaremski A., "Signature and Specification Matching," Technical Report CMU-CS-93-103, Ph.D. Dissertation, 1996.

[13] Erl, T., "Service-Oriented Architecture: Concepts, Technology, and Design," Prentice Hall PTR, 2005.

[14] Huhns M.N., Singh M.P., 'Service-oriented computing: key concepts and principles," Internet Computing, IEEE, vol.9, no.1, pp. 75-81, Jan-Feb 2005.

[15] OMG, "Corba Trading Services," http://www.omg.org/technology/documents/formal/corbaservices.htm (accessed September 25, 2009).

[16] Sundramoorthy V., "FRODO High-level and Detailed Specifications," Ph.D. Dissertation, University of Twente, Enschede, the Netherlands, 2006.

[17] Gandhamaneni J., "UniFrame Mobile Agent Based Resource Discovery Service (MURDS)," M.S. Thesis, Department of Computer & Information Science, IUPUI, June, 28, 2004.

[18] Al-Jaroodi J., Kharhash A., AlDhahiri A., Shamisi A., DhaherFi A., AlQayedi F., Dhaheri S., "Collaborative Resource Discovery in Pervasive Computing Environments," International Symposium on Collaborative Technologies and Systems, 2008, CTS 2008, IEEE, 2008.

[19] Messer A., Greenberg I., Bernadat P., Milojicic D., Chen D., Giuli T. J., Gu X., "Towards a Distributed Platform for Resource-Constrained Devices," In Proceedings of the 22nd International Conference on Distributed Computing Systems, Washington, D.C., 2002.

[20] Gu X., Messer A., Greenberg I., Milojicic D., Nahrstedt K., "Adaptive Offloading for Pervasive Computing," IEEE Pervasive Computing, 2004, pp. 66-73.

[21] Sharmin M., Ahmed S., Ahamed S.I., "MARKS (Middleware Adaptability for Resource Discovery, Knowledge Usability and Self-healing) for Mobile Devices of Pervasive Computing Environments," Third International Conference on Information Technology, Las Vegas, Nevada, 2006.

[22] Zhang D., Mhamed A., Mokhtari M., "A trustworthy framework for impromptu service discovery with mobile devices," In Proceedings of the 4th international Conference on Mobile Technology, Applications, and Systems and the 1st international Symposium on Computer Human interaction in Mobile Technology, Singapore, 2007.

[23] "DAIDALOS- Designing Advanced network Interfaces for the Delivery and Administration of Location independent, Optimised personal Services," http://www.ist-daidalos.org/default.htm (accessed September 25, 2009).

[24] Choi B., Rho S., Bettati R., "Dynamic Resource Discovery for Applications Survivability in Distributed Real-Time Systems," Parallel and Distributed Processing Symposium, International, International Parallel and Distributed Processing Symposium (IPDPS'03), 2003. 122b.

[25] Raverdy P.G., Issarny V., "Context-Aware Service Discovery in Heterogeneous Networks,' Sixth IEEE International Symposium on a World of Wireless Mobile and Multimedia Networks), 2005.

[26] Jiménez L., García-Macías A.J., "Privacy and Location-Aware Service Discovery for Mobile and Ubiquitous Systems," In Mobile and Wireless Communication Networks, 2006.

[27] Tonineeli A., Corradi A., Montanari R.,"Semantic-based discovery to support mobile context-aware service access," Computer Communications, Volume 31, Issue 5, Mobility Management and Wireless Access, 25 March 2008, pp. 935-949.

[28] Capra L., Zachariadis S., Mascolo C., "Q-CAD: QoS and Context Aware Discovery Framework for Mobile Systems," In Proc. of International Conference on Pervasive Services (ICPS'05), 2005.

[29] Zhu F., Mutka M., Ni L., "Splendor: A Secure, Private, and Location-Aware Service Discovery Protocol Supporting Mobile Services," Pervasive Computing and Communications, 2003 (PerCom 2003), Proceedings of the First IEEE International Conference, 23-26 March 2003, pp. 235-242.

[30] IETF, "Geographic Location/Privacy (geopriv)", http://datatracker.ietf.org/wg/geopriv/charter/ (accessed September 25, 2009),

[31] Denny M., Franklin M.J., Castro P., Purakayastha A., "Mobiscope: A Scalable Spatial Discovery Service for Mobile Network Resources," In Proc. of the 4th International Conference on Mobile Data Management (MDM), London, 2003.

[32] Gruber T., "What is Ontology," http://www-ksl.stanford.edu/kst/what-is-an-ontology.html (accessed September 25, 2009), 1992.

[33] Ruta M., Noia T., Sciascio E., Piscitelli G., "Ontology Driven Resource Discovery in Bluetooth Based M-Marketplace," The 8th IEEEInternational Conference on E-Commerce Technology and The 3rd IEEE International Conference on Enterprise Computing, E-Commerce and E-Services, 2006.

[34] Pawar P., Tokmakoff A., "Ontology-Based Context-Aware Service Discovery for Pervasive Environments," IEEE International Work on Services Integration in Pervasive Environments, 2006.

[35] Connelly K., Liu Y., "SmartContacts: A large scale social context service discovery system," In the 3rd Workshop on Middleware Support for Pervasive Computing (PerWare 2006), Pisa, Italy, March 13, 2006.

[36] Wagner M., Noppens O., Liebig T., Luther M., Paolucci M., "Semantic-based Service Discovery on mobile Devices," In Proc. of ISWC'05 Demo Track, Galway, 2005.

[37] Chakraborty D., Perich F., Avancha S., Joshi A., "DReggie: Semantic Service Discovery for M-Commerce Applications," Workshop on Reliable and Secure Applications in Mobile Environment, 20th Symposium on Reliable Distributed Systems, 2001.

[38] Arabshian K., Schulzrinne H., "GloServ: Global Service Discovery Architecture," Mobile and Ubiquitous Systems: Networking and Services, 2004.

[39] Papakos P., Rosenblum D., Mukhija A., Capra L., "VOLARE: Adaptive Web Service Discovery Middleware for Mobile Systems," Proceedings of the Second International DisCoTec Workshop on Context-Aware Adaptation Mechanisms for Pervasive and Ubiquitous Services, EASST, 2009.

[40] Johnson D., Maltz D.A., Broch J., "The dynamic source routing protocol for mobile ad hoc networks," Mobile Adhoc Network (MANET) Working Group, IETF, 1998.

[41] Sailhan F., Issarny V., "Scalable Service Discovery for MANET," In Proceedings of the Third IEEE international Conference on Pervasive Computing and Communications, IEEE Computer Society, Washington, D.C., 2005.

[42] Helmy A., Garg S., Pamu P., Nahata N. "Contact-Based Architecture for Resource Discovery (CARD) in Large Scale MANets," IEEE/ACM Proc. International Parallel and Distributed Processing Symposium (IPDPS), Apr. 2003, pp. 219-227.

[43] Li Z., Sun L., Ifeachor E.C., "Track-based Hybrid Service Discovery in Mobile Adhoc Networks," Proceedings of the 16th Annual IEEE International Symposium on Personal Indoor and Mobile Radio Communications, Berlin, Germany, IEEE, 2005.

[44] Ni Y., Kremer U., Stere A., Iftode L., "Programming ad-hoc networks of mobile and resource-constrained devices," Proceedings of the 2005 ACM SIGPLAN conference on Programming language design and implementation, ACM, 2005, pp. 249-260.

[45] Siram N., "An Architecture for Discovery of Heterogeneous Software Components," M. S. Thesis, Department of Computer and Information Science, Indiana University Purdue University Indianapolis, 2002.

[46] Devaraju B., "Enhancement of the UniFrame Resource Discovery Service," M. S. Thesis, Department of Computer and Information Science, Indiana University Purdue University Indianapolis, 2005.

[47] Siram N., Raje R., Olson A., Bryant B., Burt C., Auguston M., "An Architecture for the UniFrame Resource Discovery Service," Springer-Verlag Lecture Notes in Computer Science, Vol. 2596, 2003, pp. 20-35.

[48] Buckland M., Gey F., "The relationship between recall and precision," Journal of the American Society for Information Science, 1994, pp. 12-19.

[49] IBM, "J9 VM Installation Guide," http://brainmurmurs.com/products/timeout/docs/wm2003/j9install.htm (accessed March 11, 2009).

[50] Al- Masri E., Mahmoud Q.,"The QWS Dataset," http://www.uoguelph.ca/~qmahmoud/qws/index.html (accessed May 2, 2009).

[51] Apache.org, "jUDDI - An Apache Project," http://ws.apache.org/juddi/ (accessed September 25, 2009).

[52] Gallege L., Pradhan K., Raje R., "Experiments with a Multi-level Discovery System," Technical Report, Number: TR-CIS-0825-10, (Accepted to ICC 2010, New Delhi 27-28 December 2010), Department of Computer and Information Science, Indiana University Purdue University Indianapolis, 2010.

[53] Talavdekar N., "e-DTS: Enhanced Distributed Tracking System," M.S. Thesis, Department of Computer and Information Science, Indiana University Purdue University Indianapolis, 2009.

[54] Rybarczyk R., "e-DTS 2.0: A next-generation of a Distributed Tracking System," M.S. Thesis, Department of Computer and Information Science, Indiana University Purdue University Indianapolis, 2010.