DESIGN, DEVELOPMENT AND IMPLEMENTATION OF THE REGISTRATION SERVER FOR NETTRAVELER MIDDLEWARE SYSTEM

By

José F. Enseñat-Acevedo

A thesis submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE
in
COMPUTER ENGINEERING

University of Puerto Rico
Mayagüez Campus
2004

Approved by:

Wilson Rivera-Gallego, Ph.D.	Date
Member, Graduate Committee	
Pedro Rivera-Vega, Ph.D.	Date
Member, Graduate Committee	
Manuel Rodríguez-Martínez, Ph.D.	Date
President, Graduate Committee	
Lev Steinberg, Ph.D.	Date
Representative of Graduate Studies	
Jorge Ortiz Alvarez, Ph.D.	Date
Chairperson of the Department	

ABSTRACT

DESIGN, DEVELOPMENT AND IMPLEMENTATION OF THE REGISTRATION SERVER FOR NETTRAVELER MIDDLEWARE SYSTEM

By

José F. Enseñat-Acevedo

In recent years, mobile devices (e.g. cell phones, PDAs, and notebooks) have had a high increase in their networking, computational and storage power. The mobile nature of these devices causes a new set of problems not seen in conventional wired networks. Mobile devices can go off-line, experience network outages, have batteries with limited power-life, and require energy efficient components. As users start moving their application to these devices, it becomes to have a middleware infrastructure that should integrate the different data held by mobiles users. Unfortunately, existing middleware technologies are ill-equipped to overcome these new set of problems since they are designed for application that reside in workstations and mainframes. To address these issues, we are developing NetTraveler, a database middleware system to integrate mobile devices reliably in a computer network. In this thesis, we present the Registration Server for NetTraveler. Registration Servers are directory managers, located at cooperative local area networks (LANs) that work together to keep track of moving data sources and route query request as appropriate.

RESUMEN

DISEÑO, DESARROLLO E IMPLEMENTACION DEL SERVIDOR DE REGISTRACION PARA EL SISTEMA MIDDLEWARE NETTRAVELER

Por

José F. Enseñat-Acevedo

En los ultimos años los dispositivos móbiles (ej. celulares, PDA y computadoras portatiles) han tenido un aumento significativo en sus capacidades de comunicación, procesamiento y alamacenamiento de datos. La naturaleza mobil de estos dispositivos causa un conjunto nuevo de problemas jamas vistos en las mas convencionales redes de computadoras cableadas. Los dispositivos móbiles pueden irse de linea, pueden experimentar fallas de comunicacion, tienen baterias con cargas limitadas y requieren de componentes que hagan uso eficiente de energia. Segun los usuarios empiezen a mover sus aplicaciones a estos dispositivos se va a necesitar una infraestructura de "middleware" que pueda integrar la data de estos dispositivos y sus usuarios. Desafortunadamente la tecnologı́a "middleware" existente no esta preparada para sobreponer esta serie de problemas nuevos ya que estan mas bien diseñadas para aplicaciones que residan en estaciones de trabajos y "mainframes". Para dirigir estos problemas estamos desarrollando NetTraveler, un sistema "middleware" de bases de datos para integrar dispositivos móbiles confiablemente en una red de computadoras. En esta tesis presentamos el servidor de registración para NetTraveler. Los servidores de registración son manejadores de directorios, localizados en areas cooperativas de redes locales que trabajan juntas para mantener conocimiento de la ubicación de los dispositivos móbiles y redireccional pedidos apropidamente.

To Jessie, Jaise, Mami, Papi, Beba †, Kiko †, Mamá y Papá † for their unconditional support.

ACKNOWLEDGMENTS

First, I would like to thank the Lord for giving me enough strength to finish this. Next, I must thank my family to whom this work is dedicated, my fiancee, Jessenia Laboy Morales (Jessie), more than my fiancee, she is my best friend, the person that makes me continue looking further than my horizon sometimes can reach. My little sister, Jaise, she is my biggest inspiration. My Parents, Pedro Enseñat and Elsie Acevedo for their love, support, encouragement and patient through all these years. Without them, this work could not have been possible, and I would not be the person who I am today, they believe in me and give me the trust to continue ahead. To my grandparents, Amelia Rodríguez (Beba), Federico Acevedo (Kiko), Eugenia Irizarri (Mama) and Jaime Enseñat (Papa). I was blessed with the oportunity to know and shared good moments of my life with all of them, and I am sure that their blessings and good wishes are always with me to help me with all the goals that I intend in my life. Beba, Kiko and Papa are not longer with me, but they always will be in my heart and I will remember them and the good times that I shared with them as one of my most valuable treasures. Also I would like to thanks my Advisor, Manuel Rodríguez-Martínez for giving me the opportunity to work with him and for his friendship and support on the last two years. Thanks to my thesis committee members Wilson Rivera-Gallego and Pedro Rivera-Vega for their comments and suggestions on my research work. As a special mention, I would like to give my most sincere thanks to the ECE Network administrators, Pablo Rebollo, Edgard Vélez, Kennie Cruz and Iván David, their friendship, dedication and helps were a great support to finish this work and without them, I am completely sure this would not have been possible. Also I would like to thank Prof. Vilma López for her friendship during all my years of study and to Bienvenido Vélez, Daniel Mcgee and the PRECISE project for the financial support they gave me for three and a half year. Finally I would like to thank all my colleagues and friends at the PRECISE Lab and at the university in general, to all of them my most sincere thanks. They are in part the reason why I was able to finish this Master.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

1.1 Overview

In the last twentyfive years the computer related fields have experienced a dramatic growth. With this growth, changes are inevitable; from big computers and mainframes that took full rooms to accommodate and cost millions to devices small enough to fit inside a human body without even being notice. In recent years this constant change has been even at a bigger rate with Internet, as the main factor, expanding its horizons through broadband (e.g. DSL, Cable Modem) and wireless networks (e.g. IEEE 802.11b and Bluetooth). With the emergence of mobile devices (e.g. cell phones, PDAs notebooks and Table PCs among others) as reliable computational devices, existing technologies have to adjust to accommodate these continuous changes that are arising. Traditional approaches are not longer sufficient to solve all problems emerging from the union of these new technologies and devices.

1.2 Problem Statement

Computational resources, such as Data sets, processor time, disk storage, and devices such printers and LCDs can be accessed from virtually anywhere with the proper communication protocols, network and devices. A resource is an entity that can be discovered and used by applications. It may be a service, (server application) willing to respond to requests from applications. Traditionally these resources have resided on computers and mainframes that are online on a 24x7 basis, on a fixed location, and with continuous power source. Middleware systems have relied on these assumptions to provide interoperability between devices in this kind of distributed and heterogeneous environment.

In recent years mobile devices have had a high increase in their computational, communicational and storage power. Mobile phones and PDAs are the best examples to this, but even there are hand watches running fully feature operating system [1]. Most of these mobile devices are Internet-ready and have enough capabilities to run popular applications such as Web browsers, word processors, spreadsheet editors, e-mail applications, e-books readers, video games and multimedia (audio and video) players. However the truly revolutionary impact of these devices will come when they enable their users to have ubiquitous access to the Internet. With new network technologies arising this once utopian dream is a tangible reality.

Given this scenario, we can expect mobile computing devices to have a dual role for their users. First they will act as the clients sites that run the application used to obtain content from the Internet. Hence, users will demand their client applications, for example e-mail, to keep working regardless of changes in location and network connectivity. An e-mail client that works fine at office but stop working when the user goes home or in a business trip will soon become an unacceptable proposition. Second, mobile devices will become valuable sources of information that users need to carry out with them as they move across geographic locations. In this scenario we will find data sources containing very diverse and important data sets in mobile devices. Therefore, it will be necessary to build distributed systems to support seamless data access for these client applications, and to efficiently harvest their data sets. In particular, we will need integration systems to organize mobile sites into a coherent wide area information system, which also incorporates more traditional data sources (e.g. relational databases) that are residing on enterprises servers as shown in Figure 1.1

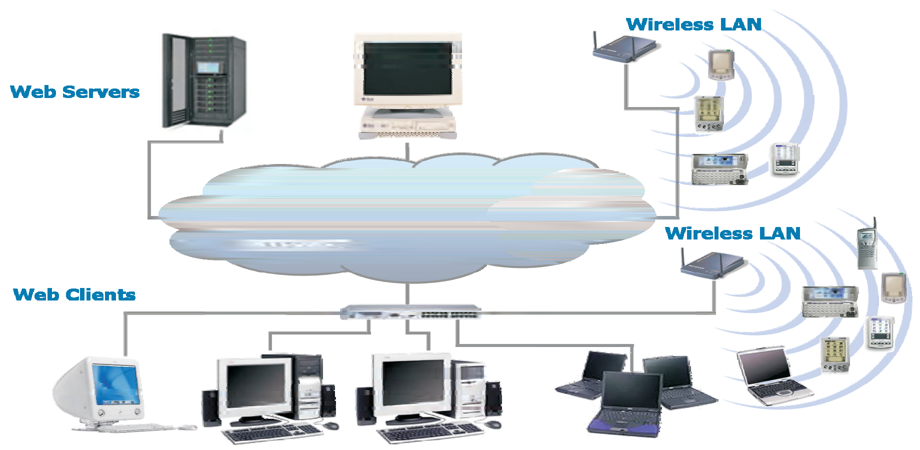

Figure 1.1: Dynamic wide-area environments

However, the mobility advantage of these low capacity devices [2] causes a new set of problems not seen in common networks. Lost of connectivity, location-dependant configurations, and limited power sources are some of the new challenges presented to developers working in these area. Database Middleware Systems were developed to integrate heterogeneous collections of distributed data sources. Database gateways [3, 4] and mediators [5, 6, 7] are the best known examples of database middleware. A fundamental assumption in these systems is that the data sources reside on heavy-weight enterprises servers that form slow changing federations of cooperative sites. Meanwhile, client sites are viewed as “couch potatoes” whose only role is to help users digest the data produced by the sources. Therefore we must realize that existing database middleware solutions for data integration are inadequate and ill-quipped to cope with environments that host mobile devices as data sources. The main reason is that they are designed for rather static systems, based on enterprise server that are on fixed location, with 24x7 access, continuous power source, configured and administered by teams of professionals, and organized into slow-changing federations of cooperative sites.

1.3 Solution Approach

The proposed solution from the Electrical and Computer Engineering Department (ECE) in the University of Puerto Rico at Mayagüez is to develop an innovative framework for building database middleware systems that support ubiquitous data access on wide-area environments. This framework will be realized in an open-source prototype system called NetTraveler. NetTraveler is a database middleware system designed to cope with dynamic wide-area environments where data sources go off-line, change location, have limited power capabilities and form ad-hoc federations of sites that works together to complete a given task and then go about their business independently. NetTraveler is addressing four research problems: 1) Self configuration of clients and servers applications, 2) Resource discovery and self-organization of federated sites, 3) Client-side context aware query processing with differentiated services, and 4) Server-side context aware query processing with differentiated services. Web Services, XML-base profiles, and Peer-to-Peer search protocols form the heart and soul of NetTraveler.

1.4 Research Objectives of this Thesis

The main objective is to design, develop and implement one of NetTraveler main components: the Registration Server (RS). The RS is a web service used to coordinate access to a federation and the resources it provides. The RS acts as a services router, which directs clients applications to the services they need to consume (use). RSs are directory managers located at cooperative local execution environments (e.g. LANs) that work together to keep track of moving data sources and route query request as appropriate. The RS also will provide client authentication and authorization to federated sites.

This thesis works with the first two NetTraveler research problems. By self configuration of client and server applications we mean that devices that are previously configured to use NetTraveler can use the RS to stay informed of new resources appearing on the network or changes in the current available resources. If a new resource appears or a current one changes its connectivity information, the RS will provide the client with these information. In this way the client can reconfigure itself with the new parameters and start or continue using the desired resource. As an example a user with a notebook arrives at a new network and after all the required validations ask the RS for a printer service. The RS will provide the client with all the parameters and information needed to connect and use this service. If at any moment this printer information changes the RS can provide the new information to the client so it can reconfigure itself with the new information and continue using the printer services as always. All this can be done in a trasparent way to the user. Also when the printer change its connectivity information, it can send all the new configuration to the RS, in this way the resource discovery and self organization of federated sites can be done without the intervention of system administrators and can be reflected immediately after the printer notifies the RS. By self organization we mean that computer devices can decide autonomously to join or leave a federation based on resources or computer needs.

In order to achieve our objective, we have created an XML-base protocol and schema specification to encode the logical operations supported by the RS. XML stands for Extensible Markup Language, and is use to encode metadata about resources over the internet. This will allow the implementation of the protocols in other programming languages (e.g. C++, C#) since XML is completely plataform and programming language independant. We have implemented an application program interface (API) for the Java programming language. The logical operations supported by the RS are described as follows:

- Creation, Dissolution and Maintenance of ad-hoc federations, Three classes of federations will be defined.
- Association, Disassociation and Re-association of members within a given federation
- Registration Servers from different LANs will work together to keep track of moving data sources and route data requests as appropiate.

We created a series of test to verify our RS implementation. These test were ran as real time simulations. In the tests, we had resources moving from one RS to another, and clients trying to access them. We verified with the test the RS performance and the ability to keep track of the devices as they move.

1.5 Summary of Contributions

The main contribution of this thesis is the design, development and implementation of a prototype of one of NetTraveler main components; the Registration Server. The RS was developed as a Decentralized Peer to Peer Architecture, this allow us to deploy them in different Local Area Networks, so they can work together for a common goal; keeping track of mobile devices as they move across different networks. The RS provided authentication, organization and search capabilities of mobile devices and the resources they have. As a result of our research we presented a paper [8] at the Seven IASTED International Conference on Internet and Multimedia Systems and Applications, realized in Honolulu, Hawaii on August 13-15, 2003.

1.6 Thesis Structure

The remainder of this thesis is structure as follows. Chapter two discusses some related works in the covered topics that served to develop this thesis. Chapter three presents an overview of the proposed solution, NetTraveler. Chapter four presents a detailed explanation of this research contribution, the Registration Server for NetTraveler. Chapter five continues with the Registration Server, this time looking at the experiments done to evaluate the performance of the system. And finally chapter six present a summary of our contributions, conclusions and future work related to the NetTraveler Registration Server.

CHAPTER 2

Related Work

We present here relevant work upon which this thesis is based. The areas are a) Network Architectures, b) Peer-To-Peer Networks, c) Web Services, d) Computer Middlewares, e) Naming Services, and f) Resource Discovery Technologies.

2.1 Network Architectures

The term, **Network Architecture** is used to define the layout of the computing systems for a specific area. It can include the physical part (computers specifications and computer location) as well as the logic part (how the computers connects between them and how the network workload is distributed among all available resources). There are many different network architectures, but in general we can define two main architectures: Centralized and Decentralized arquitectures. An arquitecture is said to be centralized if the vast majority of the processing occurs only in one place while a decentralized arquitectures possess a distributed processing schema. Here we present the advantages and disadvantages of both architectures.

Central arquitectures rely on a single server doing all the network work, this means it is cheap and easy to maintain than a decentralized arquitecture, in which multiple servers are used to do all the work. However, using a centralized arquitecture we have the drawback of a single point of failure for the whole network. If for any reason this server is down, all network services are down too, while with a decentralized arquitecture a server can be down and still the network can offer other services that are not related to the offline server. Finally a centralized server has to process all the data for all the services that is offering, meaning that while more services and more clients trying to access those services, slower the server to respond to all services request. That lead to the main drawback from those kind of systems: they are not scalable while a well design decentralized system can be scalable up to any point as long as you keep enough nodes or servers to process the workload beign generated by the system.

2.2 Peer-to-Peer (P2P) Networks

Peer-to-Peer(P2P) computing has been a paradigm in development for as long as the Internet itself. In fact, the original Internet was fundamentally designed as a P2P system. The goal was to share computing resources around the U.S. The challenge for this effort was to integrate different kinds of existing networks as well as future technologies with one common network architecture that would allow every host to be an equal player [9]. Over the time, due to bandwidth and technologies limitations, Internet became more likely of a client-server network just for serving web pages. Now in the last five years with high speed internet access becoming a reality, powerfull home computers, applications like KaZaA, and eMule appearing more frecuently in those home computers and networks such Gnutella [10], eDonkey [11], and BitTorrent [12] arrising, P2P has started to call the attention once again.

P2P applications consist of a number of peers, each performing a specific role in the P2P network, in communication with each other. Typically, the number of peers is large and the number of different roles is small. These two factors explain why most P2P applications are characterized by massive parallelization in function. The best example are file-sharing P2P networks, in which identical peer clients share files such mp3s, movies, and applications.

The P2P network provide common protocols to authentication (if required), to search files shared by others peers and to publish your files so other peers can search them within the peer community. In P2P applications, the interesting problems lie in the interaction between the peers and, to a lesser extent, in the peers themselves. The problems to be solved in P2P computing overlap to a considerable degree with the problems faced in distributed computing - coordinating and monitoring the activities of independent nodes and ensuring robust, reliable communication between nodes [13].

In other words a P2P distributed system is one in which participants rely on one another for service, rather than solely relying on dedicated and often centralized infrastructure. Instead of strictly decomposing the system into clients (which consume services) and servers (which provide them), peers in the system can elect to provide services as well as consume them. The membership of a P2P system is relatively unpredictable: service is provided by the peers that happen to be participating at any given time [14].

In general, P2P applications need at least the following elements to be fully functional: proper security including authentication and authorization; reliable message routing and delivery; content and resource management; distributed queries; and naming. We will discuss those concepts and how the Registration Server manage and implement them over the course of this thesis.

2.3 WWW and Web Services

Internet and the World Wide Web (WWW) have been in a constant growth since the beginning. According to the NEC Research Institute there are over one billion web pages that can be accessed online through this gigantic network. In the beginning those web pages only had static content on them. With the evolution of programming languages and the creation of new ones, it was possible to start creating web pages dynamically and with interactive capabilities, changing the whole Internet concept and the original purpose. WWW relies on the HyperText Transfer Protocol (HTTP) [15] for communications, a well defined and stable protocol specification by the World Wide Web Consortium (W3C). Since this protocol is very reliable programmers has used it not only for web programming but for other networks applications as well. In order to use this technology a web container or web server is required to manage all HTTP connections. Also an encoded data system is necessary to do the data exchange between the client and the server. For web programming the most famous encoding system is the Hyper Text Markup Language (HTML) for others applications relying on this technology the most common is to use the Extensible Markup Language (XML).

One of the new types of applications relying on these technologies are Web Services. A Web service is a platform independent distributed application that is deploy in a Web container or Web server, and can be accessed through the Web. The technology related with Web services have had a good welcome among distributed application developers. The main reason for this is because Web services are focused on the protocols used to exchange messages and not in the specific system infrastructure on which it is build. Web Services allow applications to call methods on objects that can be located in a remote environment to the one that the application is running. Another reason for their success is that Web services are built over existing and proven technologies. The most common application specific protocols to exchange messages in Web services are SOAP [16] and WSDL [17]. Both technologies are based on XML for which widespread support already exist in almost every modern programming language. WSDL definitions are used to describe web services, how to access them and what operations they perform while SOAP Messages are mainly used for data exchange between web services and their clients. NetTraveler prototype is being implemented as Web Services using the Java^{TM} Programming Language. The reasons to choose Java were because its provides very strong network APIs that make the creation of web services easy to develop. Also Java being portable allow us to deploy RS in different computer architectures without having to re-code nothing. Communications with and within Registration Servers are sent over the HTTP protocol as XML messages using the SOAP messaging system.

2.4 Computer Middleware

Middleware systems provide several advantages to achieve interoperability in distributed environments [18, 19, 20, 21]. Like people from different countries who cannot communicate without a translator, applications running different types of software or operating on different computer platforms need help communicating. The middleware system provides this kind of help in distributed environments. Middleware systems provide services such as authentication, directory services, and identification, among others. Database middleware systems were developed to integrate heterogeneous collections of data sources distributed over a network as a homogeneous unit [22, 23, 24]. They extend the basic middleware functionality by adding query services, data integration, and schema translation operations. In order to achieve data integration, the Middleware System imposes a global schema on top of the individual schemas used by each source being integrated. In this way, the client applications are provided with an uniform view access interface to the data sets stored by each data source [25]. A translator or a database gateway is used to “translate” between different schemas.

Existing middleware technologies usually hide the complexity of the communication between client and servers or peers, and the heterogeneity that some of these computational systems present between them. If it was not because of the middleware, theses distributed applications running over a heterogeneous environment would not be able to share information and resources. Therefore, middleware can be seen as a translation layer between a server side application and its multiple clients trying to access it (see figure 2.1). One of the main advantages of this aproach is that Some of those clients connected can be servers themselves requesting information for their own clients, creating a network of heterogeneous units sharing information in a distribute environment. In these environment data sources are application that produce data of interest to a user or another application.

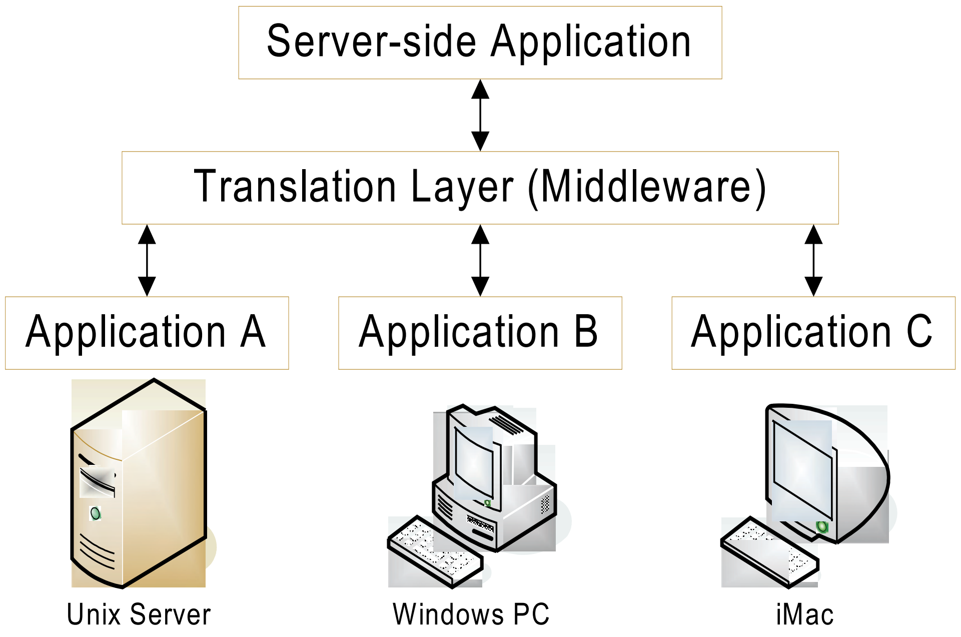

Figure 2.1: A Basic Middleware Structure

2.5 Naming Services

If a distributed application's components cannot locate one another, then they cannot work together. Therefore, distributed applications require, almost by definition, something to help the components to find each other. This service is known as the Naming Service, which is a service that associates names with locations or hosts on the Internet. A Naming Service is used in a distributed environment to resolve names into their counterpart network address. This allow users and applications to look for a specific address using a simple approach, a name that is related to the desired address. Names tend to be easier to remember than addresses. That is why Naming Services are so usefull. The most important characteristic that Naming Services should contain are 1)flexibility; names should characterize somehow what they represent, 2)scalability; to handle as much names as posible, 3)fast; to support frequent name uptades, and finally it must be 4)responsive; to quick notify applications about changes in the name.

Some Examples of Naming Services are:

- COS (Common Object Services) - This is the naming service for CORBA applications; allows applications to store and access references to CORBA objects [26].
- NIS and NIS+ (Network Information System) - Both are network naming services developed by Sun Microsystems. Both allow users to access files and applications on any host with a single ID and password.
- INS (Intentional Naming System) - Is a resource-discovery and communication system [27]. In INS, a distributed collection of resolvers form an overlay network that routes messages to destination names. Thus INS combines name resoution and message routing into a single abstraction.
- DNS (Domain Name Service) - This is the most common and the one most used naming service around the world .This is the Internet's naming service. It maps user-friendly names into computer's Internet protocol(IP) addresses. Interestingly, DNS is a distributed naming service, meaning that the service and its underlying database is spread across many hosts on the Internet. DNS from different networks helps each other to resolve names-address pairs on behalf of the users connected to them. In this sense the DNS is an example of a system that blends P2P networking with a hierarchical model of information ownership.

2.6 Resource Discovery Technologies

Networking protocols, such as DHCP, deal with the problem of assigning an IP address to a computer, as well as proving DNS server and routing parameters. Apple's Rendezvous (Zeroconf) [28] and Microsoft's Universal Plug And Play (UPnP) [29] are two technologies designed to deploy unmanaged LANs. In each case, a client device, such as a PC, PDA, or printer, sends broadcast messages to find information about other devices and services available on the LAN. This information is then used by that particular client device to configure itself with the parameters required to use another device (e.g. network access point) or service (e.g. a print server). However, the emphasis of these systems is in obtaining parameters for network connectivity at the LAN level. Applications such e-mail clients, database clients, and integration servers still need to be configured by hand. In NetTraveler, we developed a framework to enable self-configuration of devices and applications on wide-area networks, featuring capabilities to discover devices, server applications and their interfaces, even when they are in different LANs.

CHAPTER 3

NetTraveler Middleware

3.1 System Overview

NetTraveler is designed to handle mobile devices, their resources and data sets, making them appear or behave as if they were workstations or mainframes. NetTraveler is designed to cope with dynamic wide-area environments where data sources go off-line, change location, have limited power capabilities, and form ad-hoc federations of sites that work together to complete a given task and then go about their business independently. The existing database middleware solutions for data integration are inadequate for these new environments because they are designed for rather static systems based on enterprise server machines that are always on-line on a fixed location, with continuous power sources, administered by teams of professionals, and organized into slow-changing federations of sites.

In NetTraveler, handheld and mobile devices are treated as bonafide data sources, capable of delivering content to other sites in the system such as Web Servers and Data Warehouses. In addition, NetTraveler is designed to enable mobile data sources to access local and remote resources such as disk storage, printers and LCD data displays that become available while they are participating on a given federation.

3.2 System Arquitecture

In Figure 3.1 , we depict the different components of the NetTraveler architecture. The data sources are represented as cylinders, and each circle represents a LAN which can be composed of a combination of 10/100/1000 Ethernet, and wireless networks. Each LAN is a NetTraveler domain. The fundamental organizational unit in NetTraveler is the ad-hoc federation of peer sites, which is a collection of computing devices that have agreed (or are configured) to work together as a group to exchange data, and share computational resources. A federation can span one or more LANs; federations are ad-hoc because they can be formed or dissolved over the time based on the decisions taken by its members. NetTraveler exhibits Peer-to-Peer behavior because a given data source site can be serving data to one or more sites, while it retrieves data from others (perhaps concurrently).

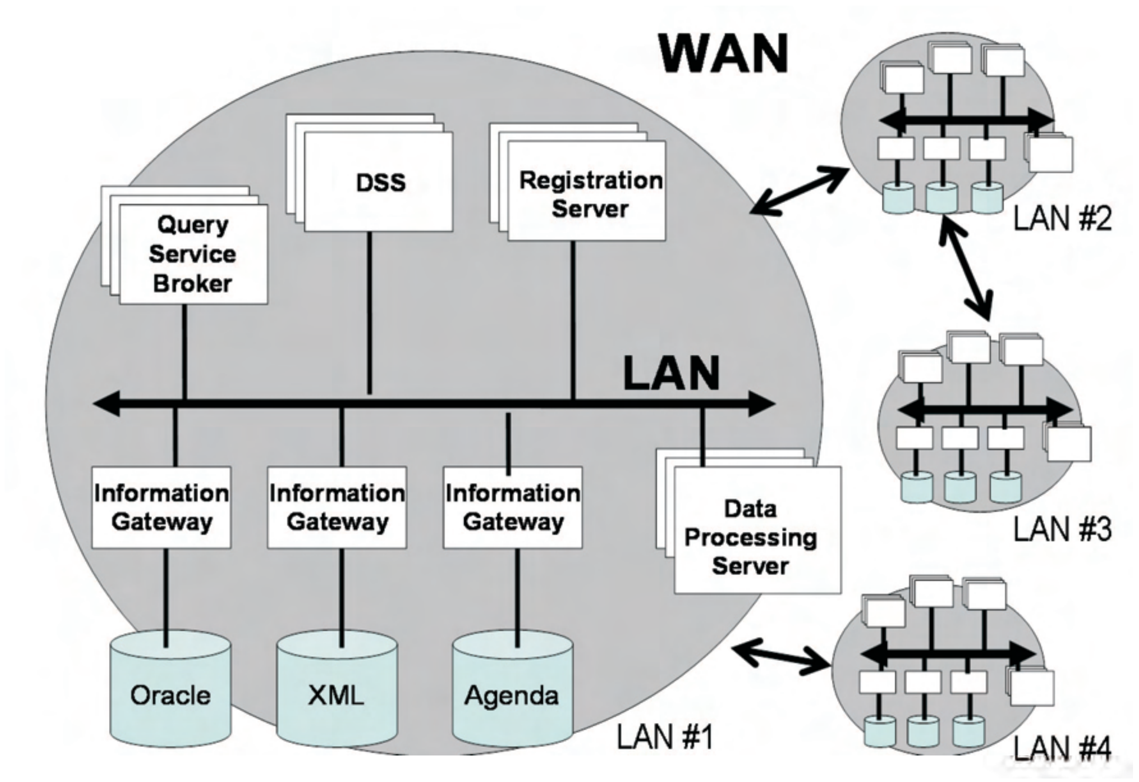

Figure 3.1: NetTraveler Architecture

The role of each of the components in NetTraveler architecture is as follows:

- **Information Gateway (IG)** - Web service that provides access to other members of the system to the data sets maintained by a given data source. This element has an equivalent role as a database gateway or wrapper. IG provides an API that maps local data objects into object-relational tuples, and makes these available to other applications. This API implements a subset of the operations found in relational database access APIs such as JDBC and ODBC, including access controls functions.
- **Query Service Broker (QSB)** - Web service that works on behalf of a client site (which could itself be a data source) to find the data and query processing capabilities necessary to solve a given query. This element has responsibilities such as query plan generation, query coordination, and acquisition of final query results. QSB provides an Object-relational model to develop integrating schemas for the data sources. All queries posed to the system are expressed in a derivate of SQL.
- **Data Processing Server (DPS)** - Web service that provides a commodity service for query execution. It provides CPU time and disk space to run query operations such as aggregation, joins and selections. This is an element that will be most likely used in environments that require parallel processing capabilities, or which have many low-powered devices. The Data Processing Server is ideal for deployment on a cluster of workstations.
- **Data Synchronization Service(DSS)** - Each mobile device will have an associated DSS that keeps a copy of all, or some portion of the data in the device that can be shared. This is equivalent to the synchronization service used in PDAs and in laptops (i.e. "the suitcase folder") to replicate shared data. The DSS is also implemented via Web services.
- **Registration Server (RS)** - This Web service is used to coordinate access to a federation and to the resources it provides. In addition, it handles all bookkeeping

necessary to log events such as departure of a federation member, or reappearance of a member at a different LAN. The RS acts as a services router, which directs client applications to the services they need to consume.

3.3 Central Paradigm

The central paradigm upon which NetTraveler is based is the idea that data and supporting environments set up on a given computing device should accompany the user as he/she moves around his/her circle of influence. Moreover these devices should be able to adapt to changes in the environment to provide users with access to newly available resources. Existing computing systems are based on location-dependant configurations which tend to isolate resources and produce "Islands of Information" that force people to plan around their computer infrastructure. At times it can be very frustrating to have applications, databases, and other tools at work or school which are unavailable when we get home or go on a business trip. Likewise, it can be disappointing to have critical personal data, multi-media files or personal memorabilia which cannot be accessed if we are not using the right computer in which the information is stored. With the new advances on wireless and broadband network technologies these dependences and limitations can be overcome. NetTraveler is moving toward that direction to make possible that your data and supporting environment can "follow you", and become available at different locations where you need access to it.

3.4 Applications

NetTraveler could be used to establish dynamic federations (similar to workgroups) of machines on manufacturing plants, hospitals, business centers, disaster zones, or schools. On a given federation, local data sources will be able to interoperate with each other, and with data sources that might be located on remote geographic sites connected via a wide-area network. For example, police officers, firefighters and emergency personnel might have PDAs and laptop computers to keep track of their locations, current supplies, and the conditions on their working environments. During an emergency situation, these computing devices might form a content network that integrates the data from each unit to give field commanders a clear picture of the situation at hand, and prepare remote supporting units such as hospitals and shelters to receive victims. Thus, the information gathered from the system can be used to better distribute resources, prioritize tasks, find dangerous and safe areas, and make decisions to move personnel out of harms way. Moreover, central authorities could get a first hand account of the situation, and make critical decisions that are not normally delegated to field commanders.

CHAPTER 4

The Registration Server

4.1 Introduction

The Registration Server(RS) is a web service used to coordinate access to a federation and the resources it provides. A RS should be located at each Local Area Network (LAN) using NetTraveler. The RS must provide a series of protocols to form, maintain, and dissolve ad-hoc federations. The RS also provide methods for membership and resources management. The RS implements a security schema to help prevent access to unauthorized users to the federations, however this schema will not impede the access, and will depend to members, as their discretion, to ensure they are collaborating and sharing resources with trusted peers. Even if the RS can not guarantee the dependability of peers, it provides the mechanisms to help determine the level of trust a member can have, should a peer member request it. The RSs are directory manager located at cooperative LANs that works together to keep track of moving resources and route request as appropiate.

4.2 Registration Server Unique Identifiers

The RS is responsible for maintaining the metadata describing the federations, devices, and resources available at a particular LAN. In NetTraveler each RS, federation, device, and resource has a unique identifier that is specified using a Uniform Resource Name (URN) [30]. Each of the different components follows a specific NetTraveler URN pattern. These patterns uses the prefix *urn:nett* to show the fact that this is a NetTraveler's URN. After the prefix the URN contains the name of the LAN that it belong followed by a descriptive part which is different for each component. Finally an identifier is added to help understanding what this URN is describing. As an example let us assume the LAN in which we are going to deploy NetTraveler is our Electrical and Computer Department network called ece.uprm.edu. A few example URNs in this network are:

- The RS's URN is assigned by the administrators at deployment time. The pattern that follows this URN is *urn:nett:ece.uprm.edu/rs/01*
- The URNs for federations are assigned by the RS when the federation is formed. All these URNs contains the RS identifier part. The URN for a federations can be as follows: *urn:nett:ece.uprm.edu/01/fed/01*
- The URNs for devices are assigned by the RS when the Device is added. All these URNs contains the RS identifier part. The URN for a PDA that belongs to the user jensenat can be as follows: *urn:nett:ece.uprm.edu/01/dev/PDAjensenat01*
- The URNs for resources are assigned by the user adding the resource. This URN follows a specific pattern that contains the RS and device to which it belongs. The URN for a MP3 music repository at jensenat's PDA can be as follows: *urn:nett:ece.uprm.edu/01/PDAjensenat01/src/mp3Repository01*

4.3 Federations

One of the RS main objectives is to manage all logical operations related to the NetTraveler ad-hoc federations. Federations are the fundamental organizational unit in NetTraveler. An Ad-Hoc Federation is a group of computers that form a collaborative team of devices. Federations can be used to share data, computational resources, or scientific instruments (e.g. motion sensors). An Ad-Hoc Federation gets form by the autonomous interactions of two or more of its constituent computing devices. They are said to be Ad Hoc because they are formed and dissolved dynamically by its constituents. As a special case a permanent federation is one that is never dissolved and corresponds to the type of federations assumed by existing middleware solutions. Federations can be either contained into one LAN, or might spawn several ones over a WAN. As shown in the figure 4.1.

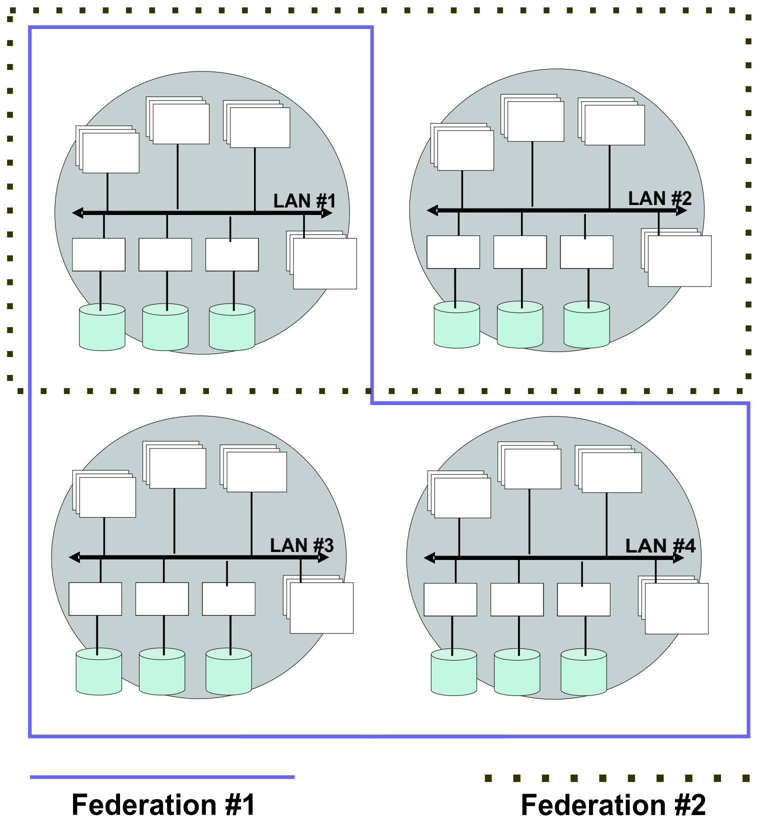

Figure 4.1: Two federations spawned into four LANs. Federation 1 (the full box) includes LAN 1, LAN 3 and LAN 4 while Federation 2 (the doted box) includes LAN 1 and LAN 2. Note that LAN 1 is spawned in both federations.

4.3.1 Federation Types

NetTraveler will support four types of Federations:

- **Generic Federation** - This federation can not be dissolved by any member and all NetTraveler members can connect to it in his/her home RS. The idea of this federation is to support management options that are not related to a federation but to members and devices (see sections 4.4.1 and 4.5.1). With this federation a member that just wants to change his/her password does not need to connect to a real working federation to do it. This federations does not include and does not allow the inclusion of any resource or service other than those mentioned above. The RS comes with this federation already defined and the URN, although it is not necessarily this way, is urn:nett:LAN Name/fed/00

- **Public Federations** - Any authorized NetTraveler member can have access to this type of federation and its resources. These are similar to file sharing communities built with eDonkey [11] or BitTorrents [12].
- **Private Federations** - Only accept certain clients or resources into the system. Typically, some arrangement and security clearance will be required to join. For example a federation formed by emergency personnal units to manage data related with a disaster falls into this category.
- **Associative Federations** - This have a hybrid type of access control; in this case, after the security clearance, access to a federation for a new member is granted by a voting majority of the members. This type of federation can be used when quality of service (QoS) guarantees are to be maintained. If the arrival of a new member poses too much of a burden on the system, the existing members can vote to deny access and maintain the current level of performance.

4.3.2 Federation Management

The RS must provide the federation management operations. The first two operations described below can only be executed connected to the generic federation, the other three must be executed connected to the federation in which you want to make the changes. The operations are describe as follows:

- **Federation Formation** - This operation is used to establish a new Federation. The federation will be initially based on the home LAN for the initial member that creates it, and the RS of that LAN will recieve the necessary information to add the new federation. This will be the home RS for the federation and this user the administrator. Later other RS can add the federation, its members and resources as its spread through the WAN. In this operation the administrator decides what services or what devices is going to support the federation. Services are predefined in the RS and can go from something as general as supporting file sharing at all to something as specific as supporting only mp3 files. Services are defined by the RS administrator and are provided as an organizational way to categorize the federation resources.

- **Federation Dissolution** - This supports the elimination of a federation. A federation can be dissolved by an administrator, or because all of its members have abandoned it. All the metadata about the federation will be moved to a historical repository for archival purposes, and messages will be propagated to invalidate all the registry records that have been cached throughout the system.
- **Member Association** - This operation is for registering a new member within the federation and specify the types of services the member can enjoy. A new member can register with the federation by means of the RS in its current LAN. This RS will contact the home RS of the federation and depending of the federation access procedures the member gets access to the federation or not. Public federations are suppose to not implement any kind of access policy, private federations may include some policies of what devices or resources can be added. Associative federations also can have some policies but also the majority of the members should vote to decide whether or not the access is granted.
- **Member Disassociation** - This operation helps the system gracefully recover query work that can not be finished because of the departure of a resource
- **Member Re-association** - This is an operation designed to support resources that move between regions. When the data source disassociates it will indicate if the action is permanent or temporary, the re-association will allow old peers to know about the

return of the data source, and pending operations will be re-started.

4.4 Members

In NetTraveler the members are the users of the system. They are identified by an username, password and a home RS. The inclusion of a home RS is because members can connect to a NetTraveler Federation from any LAN in the cooperative environment that include the federation they are connecting. The home LAN is the LAN that contains the RS in which the user was added to NetTraveler. A user trying to connect to a LAN different than his/her home LAN is said to be a foreign user. In order to connect as a foreign user a communication between the home RS and the current RS that the user is trying to connect is necessary (See section 4.9).

A member can have one or more devices bounded to him/her. Also Members can belong to more than one federation and can be connected to them simultaneously even with the same device. A member can connect to a RS without specifying the device he/she is using although the RS will limit the member to a guest status and no other privileges will be granted until he/she defines the device used to connect.

4.4.1 Members Management

The member management operations are operations directly related to the users. Most of those operations are completely independant to any federation therefore they can be execute connected to any of them as long as you are in your home RS. Also it is recomended to connect to the generic federation to manage this options. This recomendation is only based in the fact that any user connected to the generic federation will not create messages to the rest of the system avoiding the unnecessary creation of network messages for a member that just will do some management operations and then leave. The operations defined are the following:

- **Add User** - This operation is for adding a user to the system. A unique username, within the RS, a password and a permission level for the system are needed (for more information on NettTraveler permission levels see section 4.7). Additional information such as the name, and the e-mail of the user can be added optionally.

- **Edit User** - This operation is to help existent users to edit their account changing password and e-mail address. To change the permission level or the name, a NetTraveler administratior is needed.

- **Delete User** - This option will remove not only the user, but devices and resources related to this user. This operation can only be executed in the home RS of the member. All other RSs will eventually remove the device and resources after a predefined time of inactivity. Users information is never passed from the home RS to any other RS hence there is nothing to remove from servers others than the home RS.

4.5 Devices

NetTraveler is designed to help mobile devices cope with their mobility advantage however NetTraveler is not limited to only those kind of devices. Any device with processor power or storage capacity and with a wireless or wired network capability can use the system to produce or consume services and resources within a NetTraveler federation.

4.5.1 Devices Management

- **Register Device** - The registration operation consist in adding the device to the system and associating it to a exiting member. A device must have a name, a base RS (must be the RS to which the member is connected), and a network address, also a profile with optional information can be submited, however this profile will not be used by the RS in any case but it is expected to be used by others of NetTraveler main components like the Query Service Broker(QSB). The RS will provide the device with an URN once the registration is complete. Devices belongs to members, any device registered in NetTraveler must be associate directly to at least one member.

- **Delete Device** - This operation removes a device from the RS. Deleting a device will delete all resources affiliated to it. This operation can only be executed in the home RS of the member. All other RSs will eventually remove the device after a predefined time of inactivity.
- **Edit Device** - This operation allows the members owning a device to change the profile also RSs will use this operation internally to reflect the different network address the device is taking while moving between cooperative LANs
- **Device Request** - Valid members that know already what device contains the resources they are looking for can make a device request to recieve the current network address of this device. This operation required the device URN, also you can force your current RS to look for the device most up to date information in the device's home RS. This can be done adding the RS URN or URL to the search and request the RS to perform a BaseRegistrationServer search.

4.6 Resources

Resources are the heart of NetTraveler, they are added through the RS by the members. Resources can be anything that could be of the interest of the members and can be share through a network. Most resources reside on devices, some of them are the device itself (e.g. a printer). Resources can have a local or global scope. A local resource is only available to devices connected to the same RS that the device containing the resource is. Global resourses are available to all members of the federation no matter in which RS they are connected. Resources must be directly associate with at least one user and only one device. They can not be associate with a generic federations and usually private and associative federations are limited to some specific service types only. The RS does not interfere with the data or resources shared in a NetTraveler federation; its only job is to keep tracking of the resources and their locations. Also the RS will provide this information to any federation participant and to other RSs running on cooperative LANs.

4.6.1 Resources Management

- **Register Resource** - This operation helps authorized members to add resources and services to the RS. In order to add a resource, a member needs the following
  1. URN - This is provided by the member, who must ensure, following the correct pattern, that this identification is unique.
  2. A Name - The resource name.
  3. Type - This may be selected from a predefined list of service types provided by the RS in a specific federation. Some federations may allow certain service types only.
  4. Port - A port is the endpoint to a logical network connection using TCP/IP. They, in conjuntion with the device's URL, will be the access point to the resource. It is a integer which range is between 1024 and 65535.
  5. Scope - This define if the resource has a local or wide scope.
  6. Description - This can be an XML with extra information about the resource, its methods, types for parameters, return values, and any extra security schema the device implements. Like the device's profile this extra information will not be used by the RS, but can be used by peers connecting among them and others NetTraveler components.
- **Delete Resource** - This operation help the member to delete a resource that he/she is not longer interest in serve from his/her device. The resource is deleted from the home RS only, others RS will remove the resource after a predefined time of inactivity.
- **Edit Resource** - There is not a edit resource operation, the only way to do it is to delete the resource an register it again. RSs will rewrite thier resource definition, once they recieve the changes. For this operation to work, the resource URN can not be changed.

- **Resources Request** - This operation allows authorized members to search their federation for specific resources they might be looking for. A search can be done in many different ways. You can perform a search just by one of the options or combining them together. The different options allowed to perform a search are:

  1. By Keyword - This will look in the resources name or description for a match.
  2. By Service type - The result will be filtered to match only a given service type inside the member's federation.
  3. By Scope - This will filter the resource in a federation by its scope (global or local).
  4. By Device URN - The result will show the resources in a given device only.
  5. By Resource URN - This result will show the details of a specific resource provided by the member.

  Note: Members can not do a cross-federation search, they are limited to search just inside the federation they are connected. RSs internally can do this kind of search for resources propagation purposes only.

4.7 Security Schema

The RS implements two simple although very effective security policies. The first policy is a username - password authentication method and the other is similar to the files and directory permissions on a UNIX operating system environment. The authentication process will begin with a client trying to connect to an Ad-Hoc Federation through the RS. A XML message with the required information will be sent to the RS. The first authentication method will verify if the username and password match in the RS's database, and if so, the second authentication will verify if this member has permission to access the requested federation. If the authentication is successful, the RS will respond to the connection message with a SessionID message. The latter one will contain the current unique identifier of the member in the Federation. If this member is doing a reconnection after an offline period, the RS will also notify the RS from the member previous LAN about the member new location. If the connection is unsuccessful, an error message will be sent to the client with the reason of the failure.

Once a member authentication has completed successfully access to the federation is granted. There are different types of federations which rely on different types of validations methods (see section 4.3.1). Also members for a federation may have different levels of permissions. Administrator, Normal or Guest are the current three levels of permission sets on any given federation, a Guest user only can made request to existing resources. A Normal user also can share its own resources on the federation. Administrator can do the same as a Normal user but also they can change federations settings including changing a member level or even shutting down the federation. A member can belong to more than one federation and it may have different levels of permissions in each federation without any difficulty. Members in addition to have their permissions to each federation will have a level of permission for the RS itself. The same three levels will be applied to the RS with the difference that this permission level will be the same on all NetTraveler RSs over the cooperative network. Guest members practically cannot do anything but connect to any federation in which its have the permission to connect. Normal users can add a federation to the RS and will assume the administration role for it. RS Administrators will be like the root user on a UNIX system and they can change any setting for the RS, its members, or any federation on it.

4.8 Comunications

The communication will be done using a request - response messaging system. Communications within a RS are sent over the HTTP protocol as XML messages using the SOAP messaging system (See Figure 4.2). There are two different types of SOAP messages in the Registration Server, each situation uses a specific type of message but we can define two general structures for all situations. A continuation is an explanation of both messages structures and an example of how those messages works, some SOAP tags have been omitted or modified for simplifying the reading.

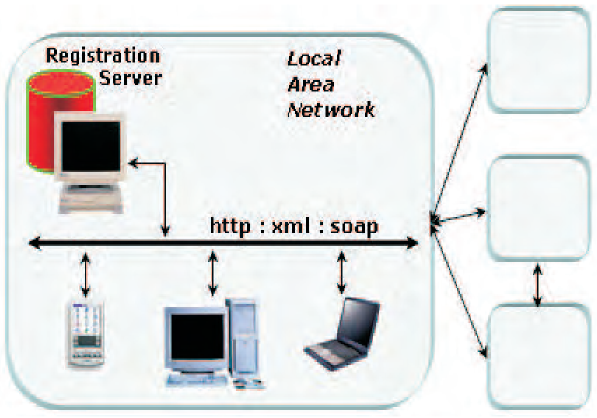

Figure 4.2: A RS on a LAN with some clients and others cooperative LANs conected

- **Request Messages** - The kind of message can be used to ask the RS to perform a specific action. The request can be performed by either a member client or another RS. These messages might need or not a sessionid or a authenticationkey attribute in the header. This is used if the requested action is only a valid option for members logged into the system (sessionid) or another RS from a cooperative LAN (authenticationkey). Figure 4.3 shows the general structure these messages have (A) and a specific example of the Login Request Action (B).
- **Response Messages** - These messages will be the answer to request messages. These kind of messages often will feature as part of the message data structures that came as response to a request. For example a client requesting the list of resources availables on a given federation will recieve the list plus additional information related to each resource (urn, service type, port, scope, etc...). Figure 4.4 shows the general structure these messages have (A) and a specific example of a Search Resources response (B).
- **Error Message** - An error message is a special type of response message. In this case the response header will show Error instead of the expecting response. The body of

```
<SOAP-ENV:Envelope>
        <SOAP-ENV:Header>
                <RS:RequestHeader SessionID=sessionid AuthenticationKey=authenticationkey>
                        Request Message Action
                </RS:RequestHeader>
        </SOAP-ENV:Header>                                                    (A)
        <SOAP-ENV:Body>
                <RS:RequestBody>
                        Additional information
                </RS:RequestBody>
        </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

```
<SOAP-ENV:Envelope>
        <SOAP-ENV:Header>
                <RS:RequestHeader>
                        UserConnection                                        (B)
                </RS:RequestHeader>
        </SOAP-ENV:Header>
        <SOAP-ENV:Body>
                <RS:RequestBody>
                        <Username>jensenat</Username>
                        <Password>123456</Password>
                        <DeviceURN>
                                urn:nett:ece.uprm.edu/01/dev/PDAjensenat01
                        </DeviceURN>
                        <DeviceURL>136.145.116.240</DeviceURL>
                        <FederationURN>
                                urn:nett:ece.uprm.edu/fed/01
                        </FederationURN>
                        <BaseRegistrationServer>
                                urn:nett:ece.uprm.edu/rs/01
                        </BaseRegistrationServer>
                        <PreviousRegistrationServer>
                                urn:nett:math.uprm.edu/rs/01
                        </PreviousRegistrationServer>
                        <LeaseCode>952295</LeaseCode>
                </RS:RequestBody>
        </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 4.3: (A) Shows the general structure of a XML request message. (B) Shows the login request XML message

(A)

```
<SOAP-ENV:Envelope>
      <SOAP-ENV:Header>
            <RS:ResponseHeader>
                  Response Message Type
            </RS:ResponseHeader>
      </SOAP-ENV:Header>
      <SOAP-ENV:Body>
            <RS:ResponseBody>
                  Requested Data
            </RS:ResponseBody>
      </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

(B)

```
<SOAP-ENV:Envelope>
      <SOAP-ENV:Header>
            <RS:ResponseHeader>
                  ResourcesSearchResult
            </RS:ResponseHeader>
      </SOAP-ENV:Header>
      <SOAP-ENV:Body>
            <RS:ResponseBody>
                  <Resource>
                        <Name>Jensenat's Mp3 Repository</Name>
                        <URN>
                              urn:nett:ece.uprm.edu/01/PDAjensenat01/src/mp3Repository
                        </URN>
                        <Service_Type>File_sharing</Service_Type>
                        <Port>3333</Port>
                        <Scope>wide</Scope>
                        <Device_URN>
                              Urn:nett:ece.uprm.edu/01/PDAjensenat01/
                        </Device_URN>
                        <Status>Online</Status>
                        <Description>
                              Additional Description of the resource
                        </Description>
                  </Resource>
                        .
                        .
                  <Resource>
                  </Resource>
            </RS:ResponseBody>
      </SOAP-ENV:Body>
```

Figure 4.4: (A) Shows the general structure of a XML response message. (B) Shows the Search Resources XML response message

these type of messages contains additional information of the error and an error code that can be reference from a table to have even more information on what happened. This error codes can be used by RS clients to determined if an action can be taken without not even notify the user of the error or at least be able to suggest him/her possible ways to avoid this error in the future. For example; SessionIDs contain an expiration date, the user might try to talk with the RS after the expiration date has passed, The system will return a Invalid SessionID error, from which the client can try to reconnect automatically and re-send the request again or at least notify the user about what exactly is the problem with its current sessionID. Figure 4.5 shows the general structure these messages have (A) and a specific example of a SessionID Error (B).

```
<SOAP-ENV:Envelope>
    <SOAP-ENV:Header>
        <RS:ResponseHeader>
            Error
        </RS:ResponseHeader>
    </SOAP-ENV:Header>
    <SOAP-ENV:Body>
        <RS:ResponseBody>
            <ErrorCode>
                ErrorCode
            </ErrorCode>
            <ErrorMessage>
                Error Description
            </ErrorMessage>
        </RS:ResponseBody>
    </SOAP-ENV:Body>
```

(A)

```
<SOAP-ENV:Envelope>
    <SOAP-ENV:Header>
        <RS:ResponseHeader>
            Error
        </RS:ResponseHeader>
    </SOAP-ENV:Header>
    <SOAP-ENV:Body>
        <RS:ResponseBody>
            <ErrorCode>
                15400
            </ErrorCode>
            <ErrorMessage>
                Invalid SessionID
            </ErrorMessage>
        </RS:ResponseBody>
    </SOAP-ENV:Body>
```

(B)

Figure 4.5: (A) Shows the general structure of a XML error message. (B) Shows the SessionID error XML message

4.9 Iteraction between Registration Servers

Communication Between RSs is the key in NetTraveler to achieve the goals of propagate up-to-date connectivity information, advertise the availability of services and resources on a given LAN, and advertise the characteristics of existing federations. The RS to RS iteraction is completely internal for NetTraveler. RSs keep a database table of peers RSs and their URLs. When a server recieves some new information about resources and services, it propagate it among its peers. Once the message arrives at a peer RS, this RS will do the same and will propagate to its peers (excluding the peer that sent the message to it) creating a chain reaction that will help propagate the metadata faster and desentralized. In order to guarantee that the servers will not flood the network with these messages, messages have a time to live span, also any RS will not replicate the same message twice. If any message arrives for a second time to an RS, this will discard the message without taking futher action with it. Messages have a unique id, which include the RS that generate the message, the time in which the message was generated and the concerning device(s). In order to a RS trust another RS that sends a message; RSs will implement an AuthenticationKey System. RSs can have a public-key obtained from a trusted certificate authority, with this, the RS first will try to validate the other RS before going futher with the iteraction.

In addition to propagate metadata, RSs will communicate to validate members that are trying to connect in a foreign LAN. If a member try to connect in a LAN other than his/her home LAN, The foreign RS will communicate with the home RS of the member, and will ask for a user validation on behalf of the member. If the home RS validates positively, then the foreign RS will issue a SessionID and grant access to the member in the foreign LAN. Also the foreign RS can retrieve from the home RS the device information of the member's device if this information is not already on that RS database. If the foreign RS does have this information, then is responsability of the member verify that this information is up-to-date with the home RS, if not, the member can ask the foreing RS to update his/her information with the home RS.

4.10 Iteraction between a Member and a Registration Server

The communication between a member and the RS in which he/she is connected is what made possible everything related to share resources in NetTraveler. Members will issue messages with actions and will recieve responses to that actions. A member will start all communication with a RS requesting a FederationList, the RS will answer that message sending the list of federations available on that LAN. From there the member can connect to any federation he/she has access issuing a UserConnection request to log into the system. Once the validation is complete the RS will assign a SessionID, which is the unique and private identifier that the client is going to use for all communications with the RS after the log-in. Once this is complete, members can search the federation for needed resources, or can register resources he/she is willing to share with peers through NetTraveler. If at any moment a device has to go offline for any reason, it can issue a Disconnect Message in which it can give a reason to the disconnection and the expected offline time (if known). This is not required but can help RSs and peers, since they will have a knowledge of the status of the device. If no message is issue RSs will asume that the client is online, until a peer notify the opposite. At this moment, the RS will try to verify the status an will update the device status if necesary to Unknown Device Status. If no peer notify the missing device, the device will continue to appears online until another RS send new information of the device or its SessionID expires.

4.11 Iteraction between Members

Members iteraction are not part of the RS duties. The RS job is to keep track of the location of the devices and the resources they are providing. The RS does not interfere in any sense with members once direct communication between them had begun. If by any reason a device has to go offline, peers can use the RS to keep track of this device and restart pending operations once the device goes online again (in the same LAN or in another). The limitation in shared resources is imposed by the device itself, the peers, the federation or the network they are connected, but in no way by the RS or any other NetTraveler component. Once a client locates the desired service, the communication between them can take any path, such as a socket, remote-method invocation (RMI), or still can be a http communication. The clients connecting, and the service specification are the ones that define their connection.

CHAPTER 5

Experiments and Results

5.1 Introduction

We conducted some experiments to determinte the RS efficiency, and to test it in a real time environment. Our idea was to setup some computers to produce fake services and some computers to go and consume those services using the RS. Then monitoring both producers and consumers we create some metrics to determine our system performance. In this chapter we are going to explain in details how we tested our system with a real time simulation and the results we obtained from that tests.

5.2 Registration Server Implementation and Deployment

In order to implement the RS, deploy it and test it, we use some computer equipment and some application softwares and tools. We will describe them here before continuing with experiment details.

5.2.1 Computer Equipment

A total of fifty one computers were used to run the experiments. Most of the computers were running a version of Linux Redhat OS, except for three of them that were running versions of Solaris OS. We used computers from different vendors (IBM, Compac, Dell and Sun Microsystem) and with different capabilities and specifications. A continuation we present the computers specifications and the role they play in the RS real time simulation. We will explain each role in detail later in Section 5.3.1.

- 21 Compaq Intel Pentium 3 500MHz with 128MB of system memory. Used for clients, both producers and Consumers.
- 13 Dell Intel Pentium 2 400MHz with 128MB of system memory. Used for clients, both producers and consumers.
- 1 Dell Intel Pentium 3 with two processors 1.4GHz and 1GB of system memory. Used as a producer.
- 1 Dell Intel Xeon with 2 processors 2.8GHz with 2GB of system memory. Used as a producer.
- 4 IBM Intel Xeon with 2 processors 2.8GHz with 2GB of system memory. Used for Registration Servers deployment.
- 2 Solaris Sparc 440MHz with 2GB of system memory. Used for Registration Servers deployment.
- 6 Dell Intel Xeon with 2 Processors 2.40GHz and 512MB of system memory. Used as a producer.
- 1 Compaq Intel Pentium 3 500MHz with 512MB of system memory. Used for Registration Server deployment
- 1 Solaris Sparc with 4 Processors 400MHz and 1GB of system memory. Used as a producer.
- 1 Dell Latitude C840 Notebook with a Mobile Pentium 4 1.6GHz and 512MB of system memory. Used as the Master Control of the experiment.

In total the experiments used fifteen producers, twenty eight consumers, seven Registration Servers, and one Master Control. All computers are part of our university network connected through 10/100 ethernet cards. The twenty one compaq, the thirteen

Dell, and the notebook plus all computers running the RSs were totally assigned to our experiments. Some of the experiments were realized in the Amadeus Laboratory (which have fifteen Dell Intel Pentium 4 2.4GHz with 512MB of system memory and thirty Compaq Intel Pentium 3 500MHz with 128MB of system memory) but due the longitude of some tests and the fact that this laboratory is open to the ECE student community, we faced some problems running the mayority of the experiments there because the students reboot or turn off the computers from time to time, resulting in too many incomplete experiments because the missing results from the offline computers. However, even while we faced this situation in overall the system never crashed because of the missing computers.

5.2.2 Softwares and Tools

In order to complete the RS, we use some softwares and tools that are described as follows:

- Java SDK 1.4.2.01 - The RS is completely programmed in Java^{TM} We used Java because the platform independent flexibility provided by this programming language. Also Java has a lot of tools for helping with the creation of web services [31], allowing us to concentrate our effors in our problem statement instead of the web services itself.
- Java Web Services Development Pack 1.2 (Java WSDP) - This package provides the tools and libraries necessary to simplify the development of web services using the Java^{TM} 2 Platform.
- Apache Tomcat 4.1 Container - This is the web server used to run our Web Service, this is part of the Java WSDP 1.2.
- JAXM 1.1.2 - This is an optional package that its integrated to the Java WSDP 1.2. This package enables applications to send and receive document oriented XML messages using a pure Java API. JAXM implements Simple Object Access Protocol (SOAP) 1.1 [16]. This is the XML protocol used for all RS communications.
- Mysql 4.0.14 - This is the Database server used to store all the information of the

federations, members and resources, also to store the data from the RSs' metadata exchange.

5.3 Experiments

As defined in chapter 3, the RS is a web service that acts as a service router. One of the main feature is that the system integrate mobile devices as reliable data sources in a computer network. In our simulation, our first challenge was to simulate the mobile devices. The main characteristic these devices have, is that they can change its network address (IP) while they move across networks. From now on, we will address both terms, computer and device with a different connotation. A computer will be the physical workstation with a static network address, while a device will be just a concept, a dynamic entity that can change its network address. In our simulation we developed a way to assign different devices to reside on the same computer on different lapses of time. In this way a computer can emulate any device that is assigned to it. With this technique we can emulate mobile devices with a network of computers with static addresses. For a better understing of the mobility simulation, see section 5.3.2, however, we recommend that you read the computer's roles definition first (next section).

5.3.1 Computer Roles

With the mobility characteristic simulated, we assign a role to all computers within the experiment. A continuation we are going to explain the different roles and their meaning.

- **Registration Server** - This will be computers providing the RS web service. There are seven of this kind of computer in our experiment.
- **Producer** - This is a device providing resources or services to others devices. There are fifteen computers but only fourteen devices in our experiment.
- **Consumer** - This is a device that is consuming a resource or service provided by a producer. There are twenty eight computers and twenty eight devices in our simula-

tion.

- **Master Control** - This computer is include in the experiment to simplify the simulation of moving mobiles devices. This computer job is to determine what devices to move and the computer to which will move it. It will send messages to both computers (the one that will release its current device, and the one that will begin emulating the device). This component is only for our simulation, it is not part of the Registration Server or NetTraveler in any sense. There is only one master control in the simulation.

5.3.2 Mobility Simulation

In figure 5.1, we shows how with workstations with static ip addresses we were able to simulate mobile devices. Figure 5.1A shows three workstations with a master control. The master control send an order to the workstation 1 to run the device A profile and to the workstation 2 to run the device B profile, the workstation 3 remains as a free computer without a device to emulate. Then after a predefined amount of time, the master control send an order to workstation 1 to stop emulating the device A, and send an order to workstation 3 to start emulating the device A profile (figure 5.1B), workstation 1 remains as the free computer now. Once these steps are finished we have simulated a device movement moving the device A from the workstation 1 to the workstation 3. Using this idea with 14 Devices as producers and 15 workstations we were able to create the effect that the producers were moving from one place to another.

5.3.3 Purpose of the Experiments

We develop a series of experiments to test our RS prototype. The RS is suppose to be able to exchange metadata with peer RSs. The metadata are encoded XML messages using the SOAP protocol. Since the RS only send messages to its peers, we want to test our prototype with different RS peers configurations. A peer configuration is the way the RSs are connected between them. They resembles network arquitectures, however we prefer to call them peer configurations because we felt that this name was more appropriate to

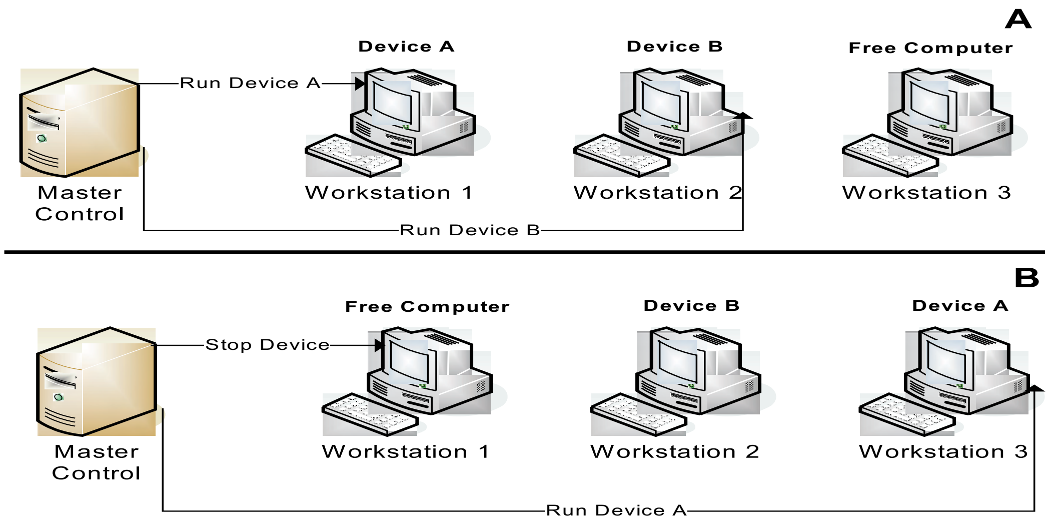

Figure 5.1: How we simulate the mobile devices

what they really are. Figure 5.2 shows the four different peers configurations that we used in our experiments. As you can see from the figure, configuration one to three are decentralized while configuration four is centralized. Each experiment was tested with all four configurations. With the experiments we want to prove that the RS manages the message exchange efficiently and that the system will not flood the network with RS to RS messages. Also we want to verify that the system is able to keep track of mobile devices and route the resources request appropriately. Finally our last experiment verifies that the system has a acceptable response time for client requests.

NetTraveler exhibits a peer-to-peer behavior, thats mean, that any device can be cosuming a resource at the same time that it is producing a resource. However, for simplicity reasons in our simulation the devices will assume only one of the roles (producer or consumer). Also only the producers will be moving between RSs. Since the RS goal is to keep track of devices that contains resources (producers), adding the complexity of moving the consumers does not bring anything relevant to our test results.

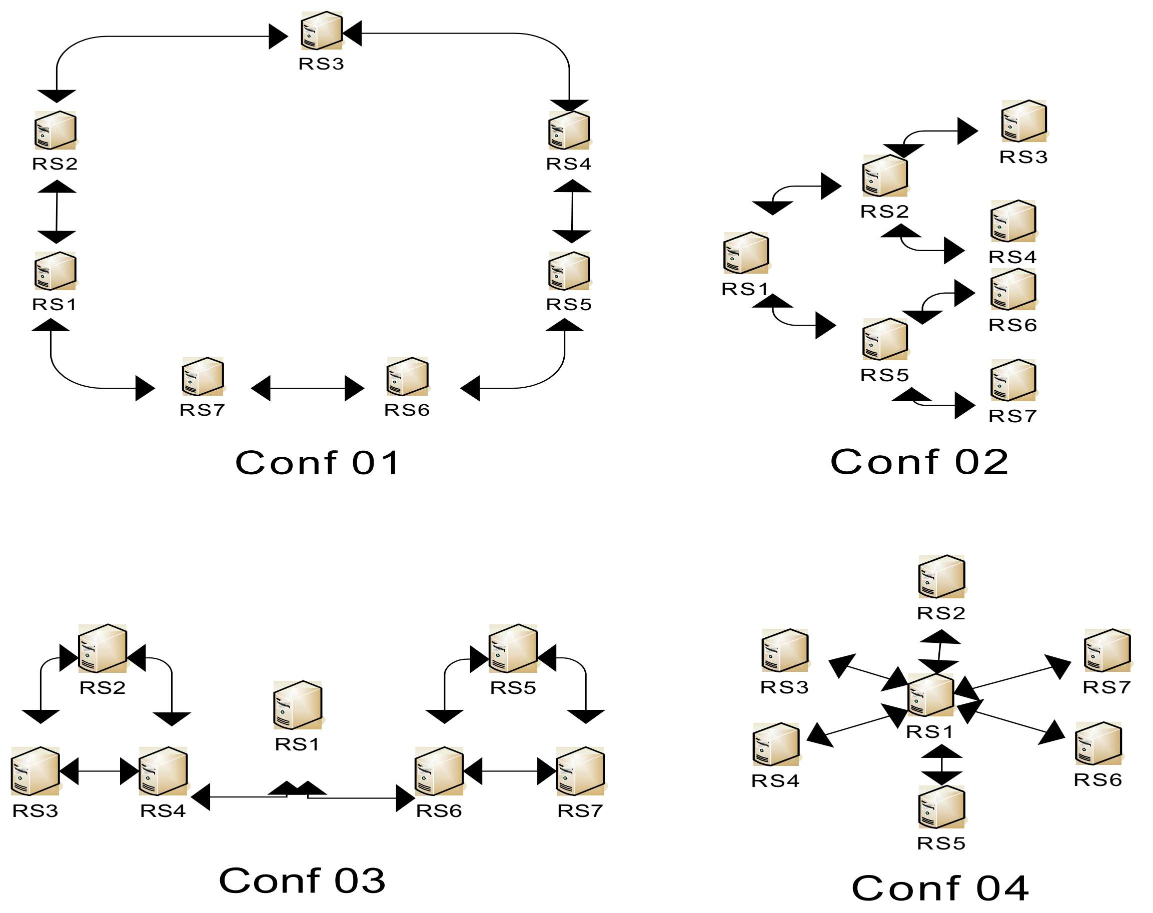

Figure 5.2: The four peers Registration Servers configuration used to run the experiments

5.3.4 Test 1

The first test was to verify the number of messages generated for a specific number of devices movement. A device move is the process in which a device disconnect from the RS, changes network address and connect again to the same or another RS. In this test we count the messages generated by the RSs after a predefine number of devices movement.

5.3.4.1 Methodology

In this simulation, we use the producers, the RSs and the master control, since the main idea of the test was to verified the number of messages generated by the RSs in response to devices movement, and the only devices we are taking into consideration for movement are the ones containing resources (producers) then, consumers were not required in this simulation. In the test, The master control, assigned a device profile to the first fourteen computers. These devices were connected to their home RS, log-in into the system and then registered their resources. Then, the master control start to redesignate the computers and the device profile on it randomly. After a predefine number of reassignments, the master control order all devices to disconnect from their current RS they were connected and proceed to count the numbers of messages generated by all RSs. We count the number of messages after 1, 25, 50, 100, 500, 1000, 2500, 5000 moves. All test were repeated three times, then an average of the results were taken as the final result for the test. These results are shown in the next section.

5.3.4.2 Results

Next we are going to show the test #1 results for all four configurations. A table with the values and the corresponding graph is shown for each configuration. Table 5.1 and figure 5.3 are the test results for configuration #1. Table 5.2 and figure 5.4 are the test results for configuration #2 Table 5.3 and figure 5.5 are the test results for configuration #3. Table 5.4 and figure 5.6 are the test results for configuration #4. As we can see all graphs are close to linear functions, which mean that for the system the peers configuration is not a determinant factor. Also the linear function is expected since it mean that the messages are directly proportional to the number of devices movement on the system, which is the desired behavior the system should follow. Looking at the different graphs we also note that the configuration that create less messages in the five thousand moves test was the configuration 2. If we look at this configuration in figure 5.2 we note that this configuration is, from the desentralized configurations, the one with less connections between peers.

Number of Moves	Number of Messages
1	15.33
25	48.86
50	79.57
100	154.95
500	711.43
1000	1395.38
2500	3415.81
5000	6954.62

Table 5.1: Results for Test 1 - Configuration #1

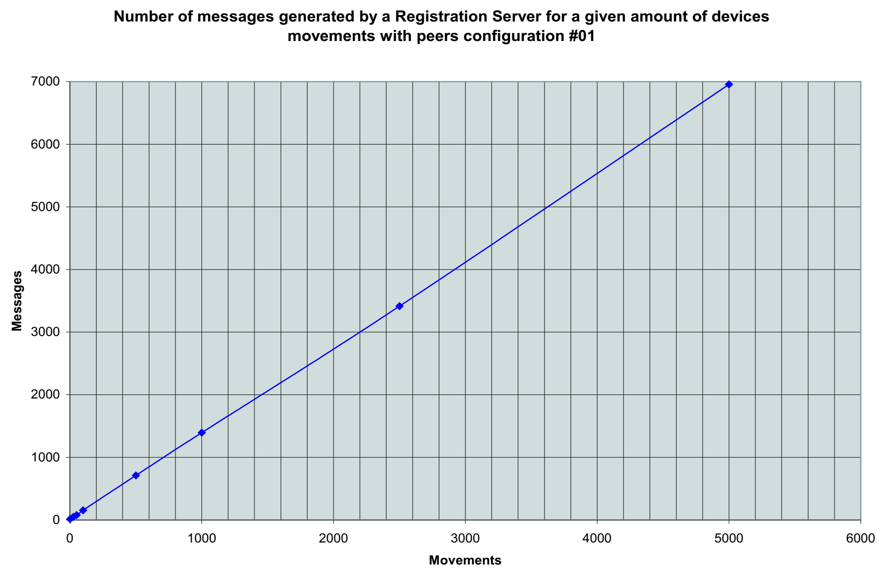

Figure 5.3: Test 1 - Configuration #1, This is the graph for table 5.1

Number of Moves	Number of Messages
1	14.76
25	45.19
50	76.43
100	142.29
500	651.62
1000	1275.14
2500	3149.76
5000	6329.10

Table 5.2: Results for Test 1 - Configuration #2

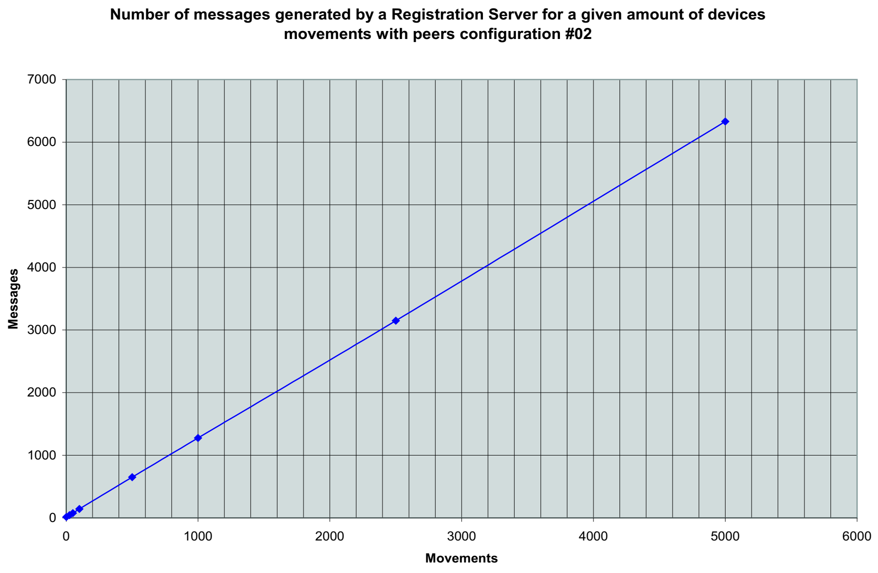

Figure 5.4: Test 1 - Configuration #2, This is the graph for table 5.2

5.3.5 Test 2

The second test was to verify that the RSs are capable to keep track of the mobile devices as they change locations. In order to keep track of the mobile devices, RSs have to exchange metadata between them, but at the same time they need to answer consumers requests. The veracity of the responses to those requests are what determined the efficiency

Number of Moves	Number of Messages
1	15.52
25	47.29
50	79.71
100	143.62
500	670.14
1000	1310.90
2500	3255.90
5000	6572.05

Table 5.3: Results for Test 1 - Configuration #3

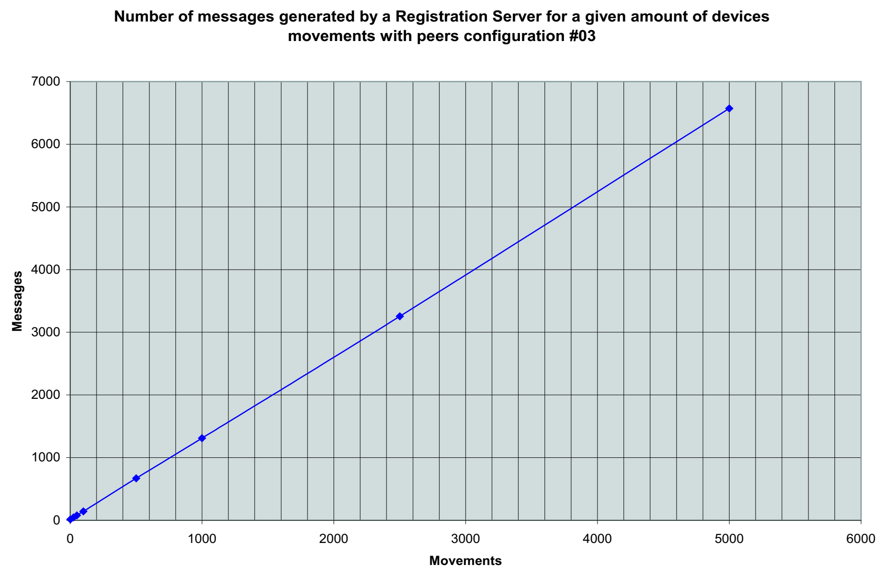

Figure 5.5: Test 1 - Configuration #3, This is the graph for table 5.3

Number of Moves	Number of Messages
1	14.86
25	46.76
50	80.48
100	146.43
500	669.67
1000	1341.43
2500	3285.48
5000	6626.71

Table 5.4: Results for Test 1 - Configuration #4

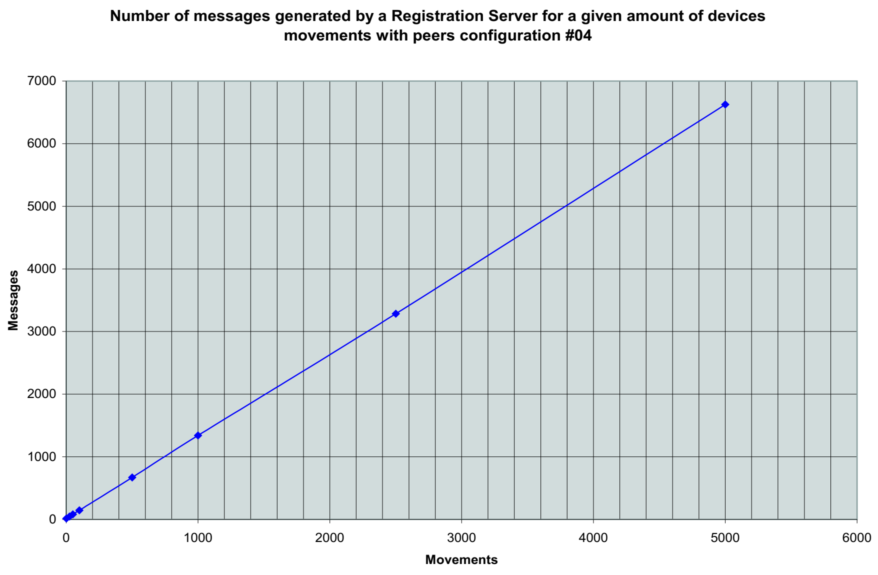

Figure 5.6: Test 1 - Configuration #4, This is the graph for table 5.4

of the RSs exchanging metadata. In this test we verify the responses given by the RSs to the consumers, as they search for resources. We count those responses and we classified them into four categories:

- Hits - These are good responses, the RS was able to answer with the correct information.
- Miss - These are wrong responses. the RS was unable to provide with the correct information about the resource current location.
- Force Hits - After a wrong response, the devices have the option to "force" the RS to search for the desired resource asking the home RS of the resource. If after that forced action the RS response is correct, it will be counted as a force hit because it was necessary to do the extra step in order to get the correct answer.
- Offline - This are requests for resources that are not available at the moment, but the RS was able to inform the consumer properly about the resource unavailability.

5.3.5.1 Methodology

In this simulation, the idea was to count and to categorize the RSs responses as the producers were moving through the network and the consumers were asking for different resources. Both, the producers movements and the consumers request were done randomly. In this simulation all devices (consumers and producers) began in their respectives home RSs, then the master control started to move the producers so they can change their ip address. Consumers in the other hand, remained on their home RSs for the whole simulation. They asked the RS for the resources list and from there, select a random resource and try to connect. The result from those attempt were counted and categorized as described in section 5.3.5. This simulation was done varying two parameters, the producers frequency which was the time between producers moves and the consumer's request rate which was the time between consumers request. We create simulations for thirty seconds, two minutes and four minutes frecuencies. With each of those frecuencies we performed tests for different consumers rates. we use sixty, eighty, one hundred, one hundred twenty, one hundred fifty and one hundred eighty request per minute. All simulations were done for all four peer configurations. Each simmulation ran for about an hour, then we gather from all consumers the data concerning the responses to all their requests. Each test was repeated three times then an average was taken as the final result for the test. These results are shown in the next section.

5.3.5.2 Results

Table 5.5 shows a summary of the twelve simulations done as part of this test. It shows the peer configuration and the producers frequency used for each simulation. Also it shows the reference to a graph that shows each simulation results distribution. These are pie charts divided into the four categories we classify the RS responses (hits,miss,forced hits, offline).

As you can see from all pie charts, none of the configurations follow a specific

Peer Configuration	Producers Frequency	Figure #
1	30 secs	5.7
2	30 secs	5.8
3	30 secs	5.9
4	30 secs	5.10
1	2 min	5.11
2	2 min	5.12
3	2 min	5.13
4	2 min	5.14
1	2 min	5.15
2	2 min	5.16
3	2 min	5.17
4	2 min	5.18

Table 5.5: All simulations related to Test 2, each of these test was run with all consumers' request rate.

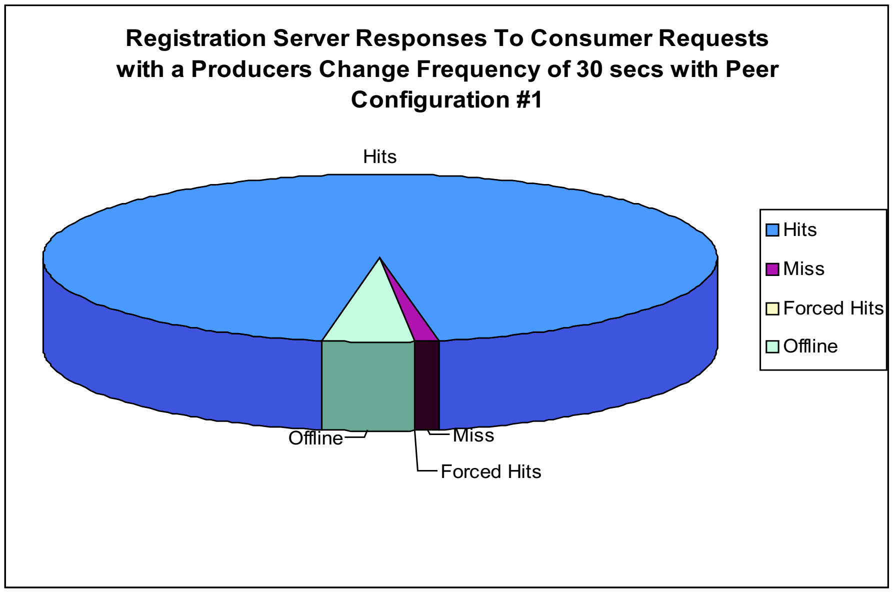

Figure 5.7: Test 2 - Configuration #1 - 30 Seconds Producers Frequency

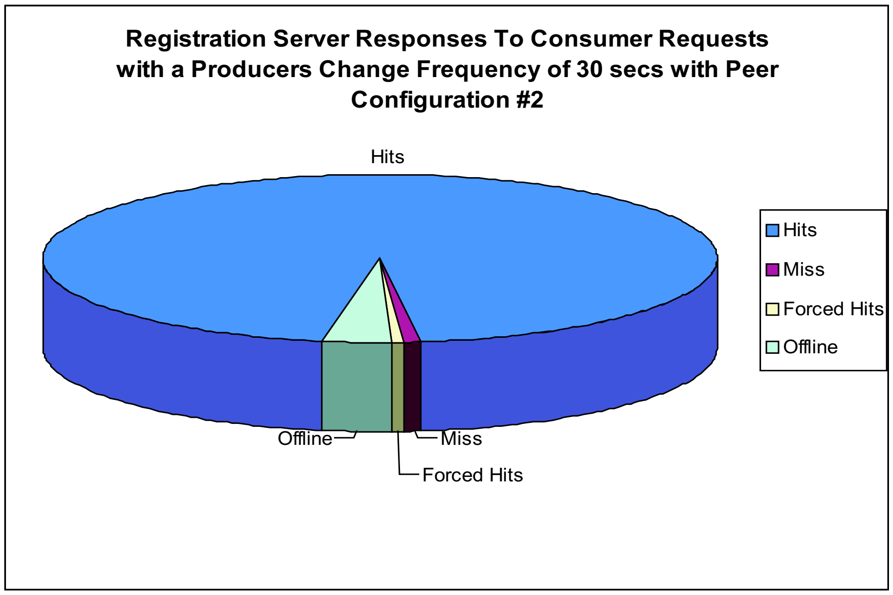

Figure 5.8: Test 2 - Configuration #2 - 30 Seconds Producers Frequency

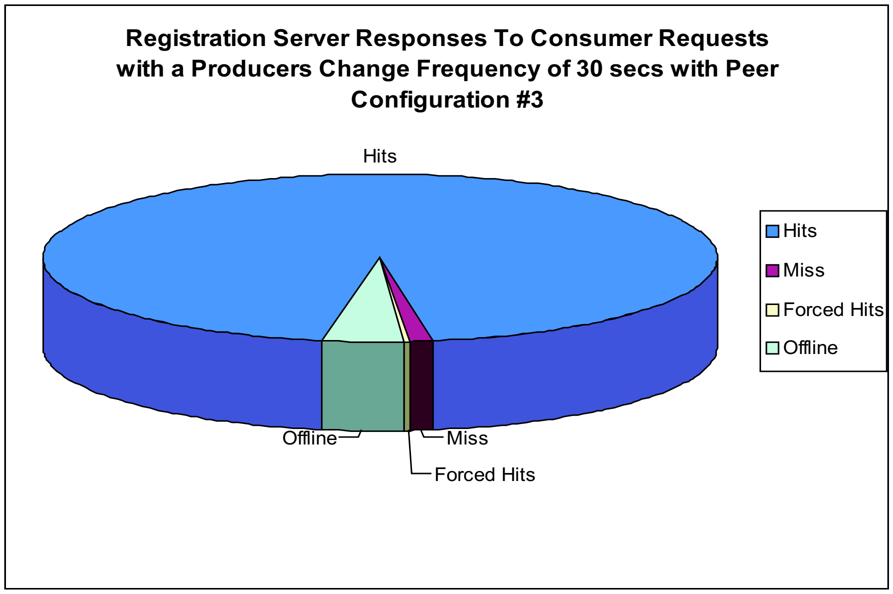

Figure 5.9: Test 2 - Configuration #3 - 30 Seconds Producers Frequency

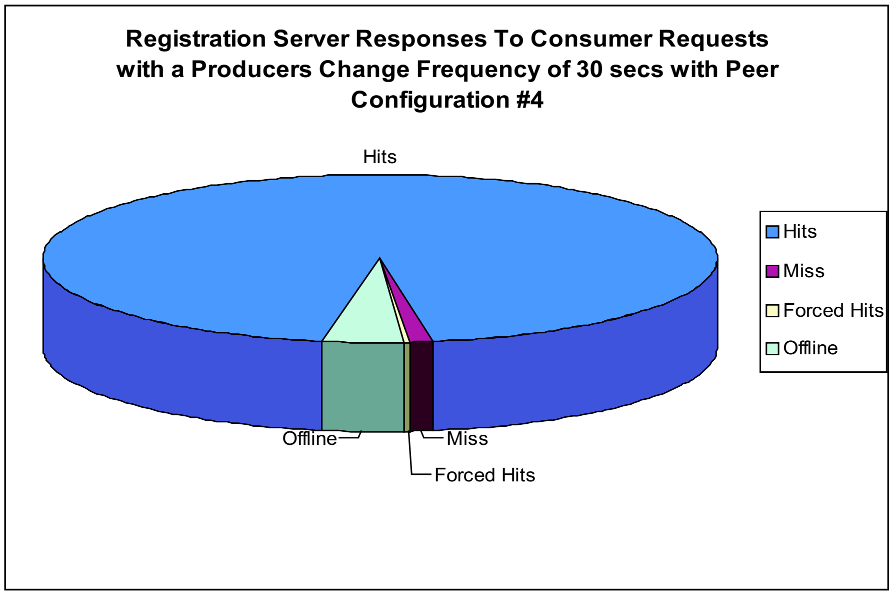

Figure 5.10: Test 2 - Configuration #4 - 30 Seconds Producers Frequency

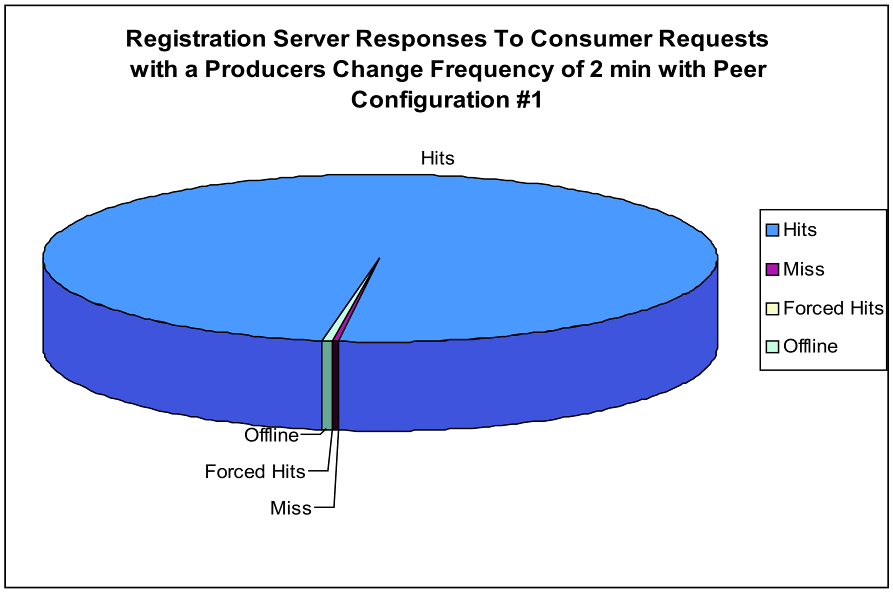

Figure 5.11: Test 2 - Configuration #1 - 2 Minutes Producers Frequency

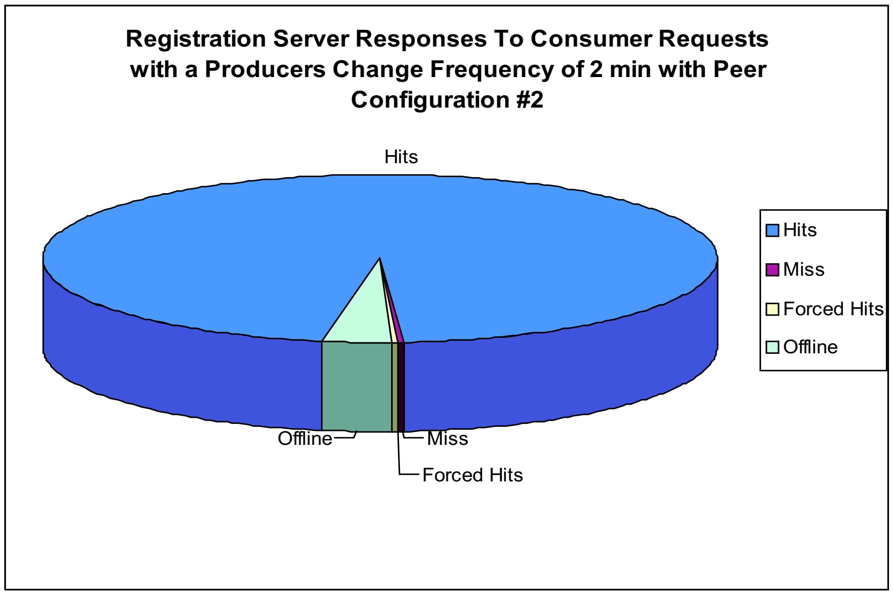

Figure 5.12: Test 2 - Configuration #2 - 2 Minutes Producers Frequency

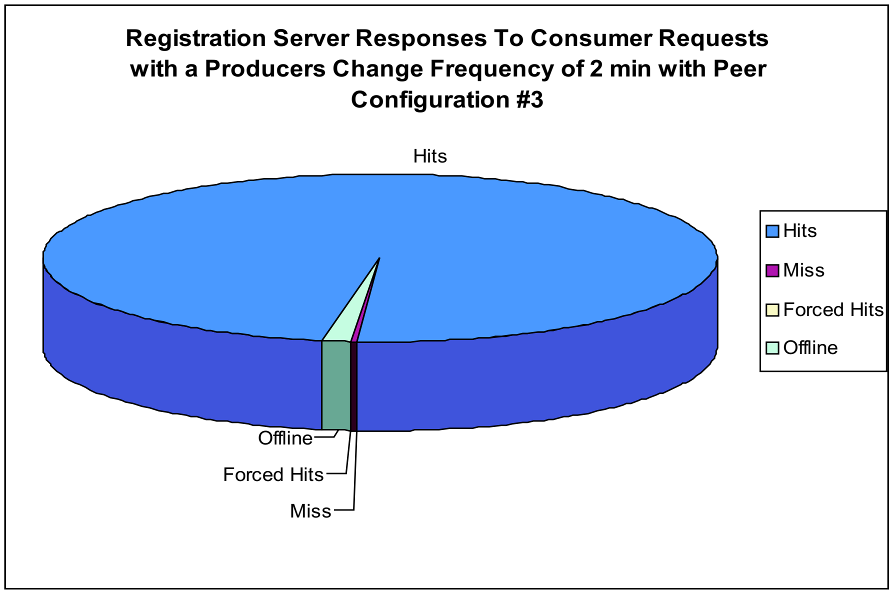

Figure 5.13: Test 2 - Configuration #3 - 2 Minutes Producers Frequency

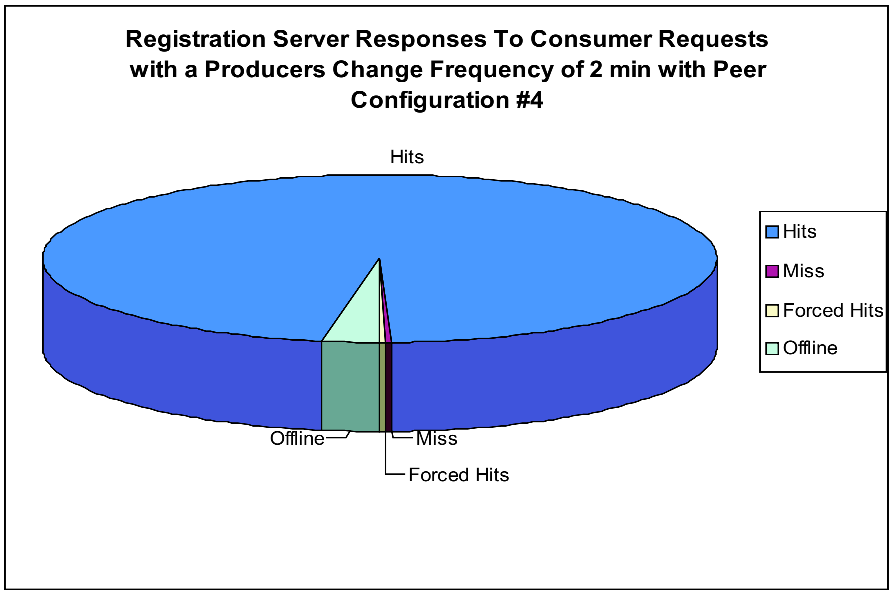

Figure 5.14: Test 2 - Configuration #4 - 2 Minutes Producers Frequency

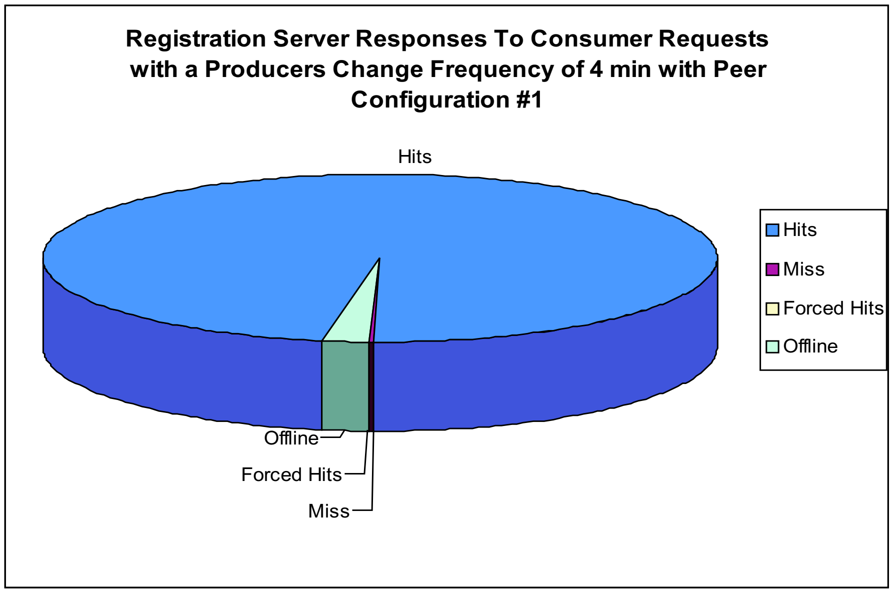

Figure 5.15: Test 2 - Configuration #1 - 4 Minutes Producers Frequency

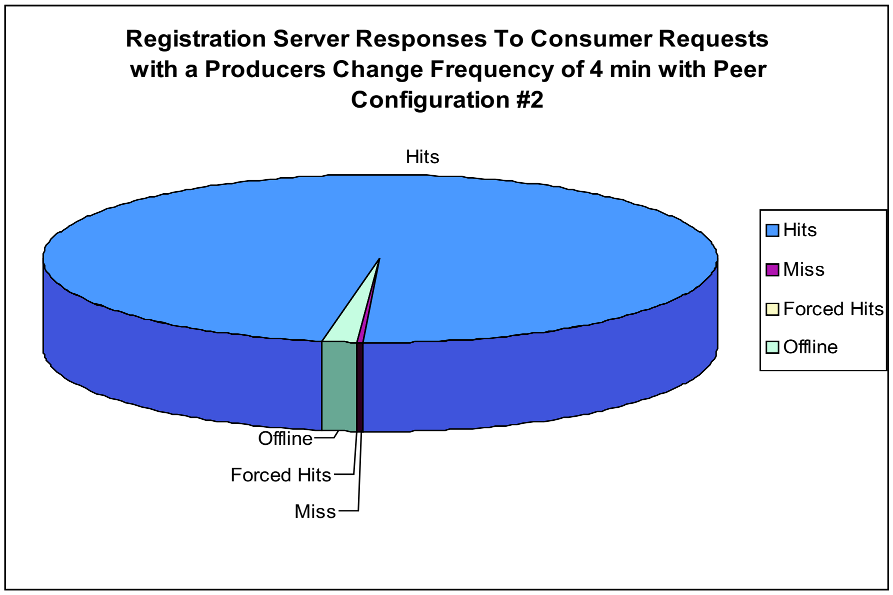

Figure 5.16: Test 2 - Configuration #2 - 4 Minutes Producers Frequency

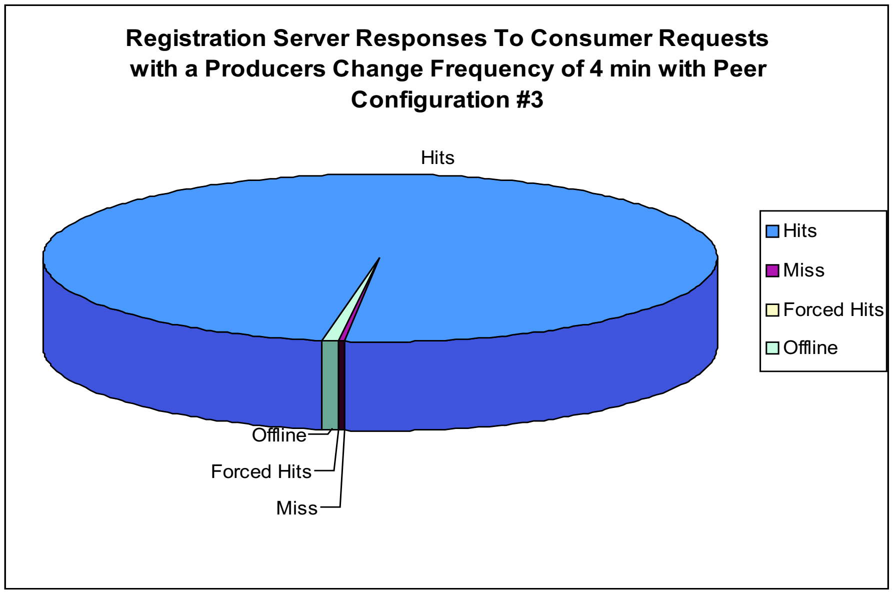

Figure 5.17: Test 2 - Configuration #3 - 4 Minutes Producers Frequency

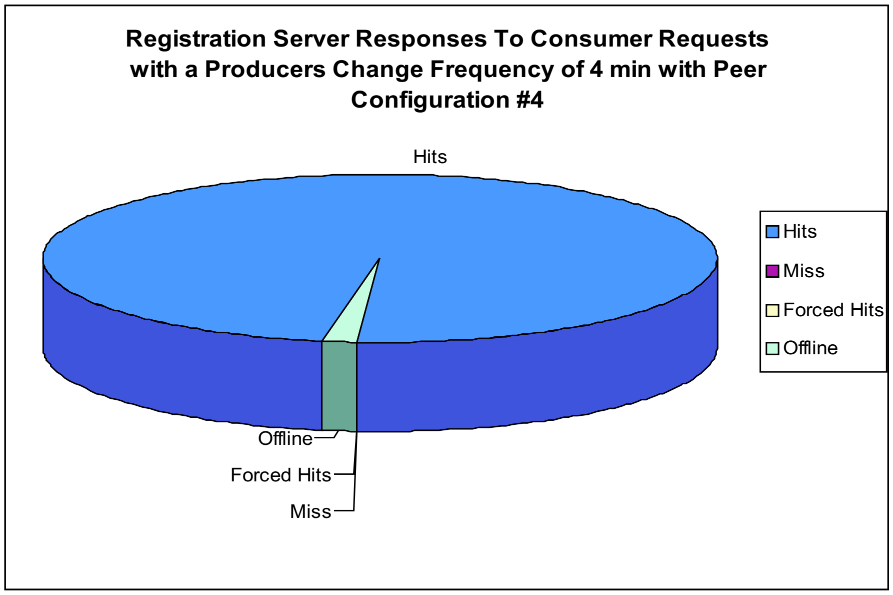

Figure 5.18: Test 2 - Configuration #4 - 4 Minutes Producers Frequency

pattern, or have a defined tendency, this could be possible, mainly, because the test was set from randomly movements and requests. However as you can see all simulations produced a very high numbers of hits in contrast to the numbers of misses. Also note that forced hits and offline resources can be counted as correct answers, since the RS was able to determine the resources status and only the misses category reflects a wrong answer provided by the RS. With this test we confirmed that our RS prototype is completely independant for the RS peers configuration used. Also this test shows that the RS is able to react properly despite different producers frequency and consumers' request rate.

5.3.6 Test 3

The third test was to verify the response time that the RS took in answering any given consumer request. A consumer request start with a client (the consumer) creating a RS request with a specific action, then this action is sent to the RS that the client is connected. The RS take the action and return a request response message. The whole process is what we considered a consumer request. This is like the second test just that this time we focused in the response time and no in the response content. In this test we used only one producer frequency of two minutes. We simplify the test with just one producer frecuency based on the fact that in a real time environment a mobile device ussually will not change network connectivity information with seconds of difference. This however is not the same with consumers request rate. We wanted to prove if the system can support a large number of client request per minute so we used, like in the second test, sixty, eighty, one hundred, one hundred twenty, one hundred fifty and one hundred eighty request per minute per consumer. As always each test was repeated three times and then an average of each test was taken as the experiment's result.

5.3.6.1 Methodology

We measure (in miliseconds) the time that took the consumer request process to complete, from the time a client generate a message to the time a response is recieved from the RS. This test is much like the second test, only that here we did not consider the response content. We just wanted to know the time in miliseconds that took to the RS respond to a request message, in other words we just wanted to know how long it takes to the RS to produce a response after a client has send a request. The result for these tests are shown in the next section.

5.3.6.2 Results

In this section we present the results for the third test, here we present tables and graphs with the respond time for all four configurations. Looking at the graphs we can see that there is a common exponential pattern in the results, this mean that in the system, while more requests per minute, the respond time will be bigger. This means that the system will become slower . Please also note, that this times have been influenced by some external factors that are out of our control; Network delays, which tend to be different at different hours of the day, might have influenced the results, also the computational capabilities of the different RSs might have influenced too. Despite these factors, the response times seems to be very similar by all three decentralized configurations while the centralized configuration seems to be going slow at a higher rate than its counterpart. The reason for this behavior is because in the centralized configuration all messages, and request are sent to the central server, creating a bottleneck in communications and computational capabilities on this server. This proves what we already said about centralized arquitectures in chapter 2. Centralized arquitectures are not scalable, here we have a clear example, while all decentralized configurations was in an average of 33 ms, the centralized peer configuration went as high as 1032 ms in a single response time. Figure 5.23 at the end of this chapter, shows a colums graph for all four peer configurations, from this graph we can see the similitude of the three decentralized configurations in response time, while we can see how the configuration #4 outbounds the rest of the configurations by a highly considerable margin.

Number of Moves	Number of Messages (ms)
60	30.5
80	30.4
100	30.8
120	31.6
150	32.5
180	35.2

Table 5.6: Results for Test 3 - Configuration #1

Number of Moves	Number of Messages (ms)
60	31.2
80	31.4
100	31.3
120	31.8
150	32.1
180	34.6

Table 5.7: Results for Test 3 - Configuration #2

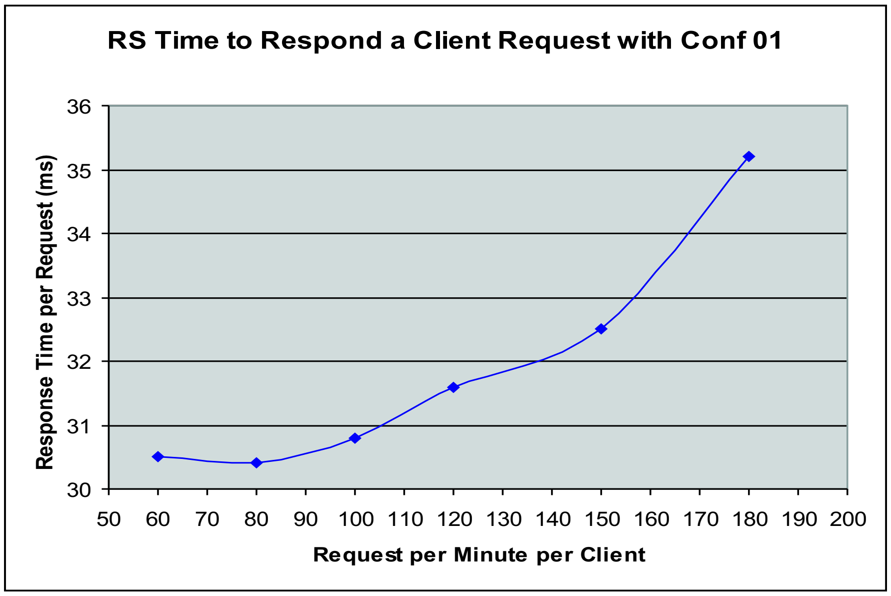

Figure 5.19: Test 3 - Configuration #1, This is the graph for table 5.6

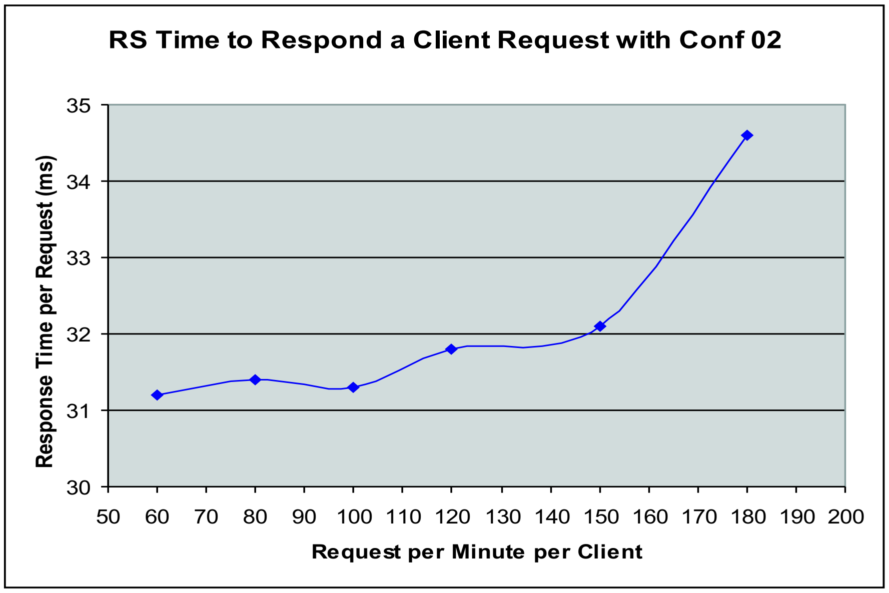

Figure 5.20: Test 3 - Configuration #2, This is the graph for table 5.7

Number of Moves	Number of Messages (ms)
60	30.4
80	30.2
100	30.3
120	31.7
150	32.5
180	35.2

Table 5.8: Results for Test 3 - Configuration #3

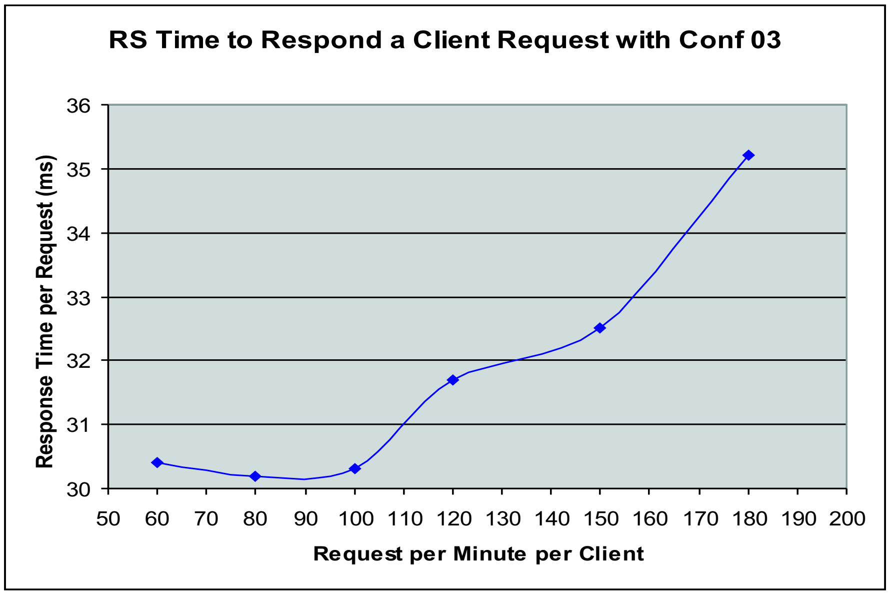

Figure 5.21: Test 3 - Configuration #3, This is the graph for table 5.8

Number of Moves	Number of Messages (ms)
60	490.4
80	546.6
100	609.3
120	692.8
150	814.4
180	1032.5

Table 5.9: Results for Test 3 - Configuration #4

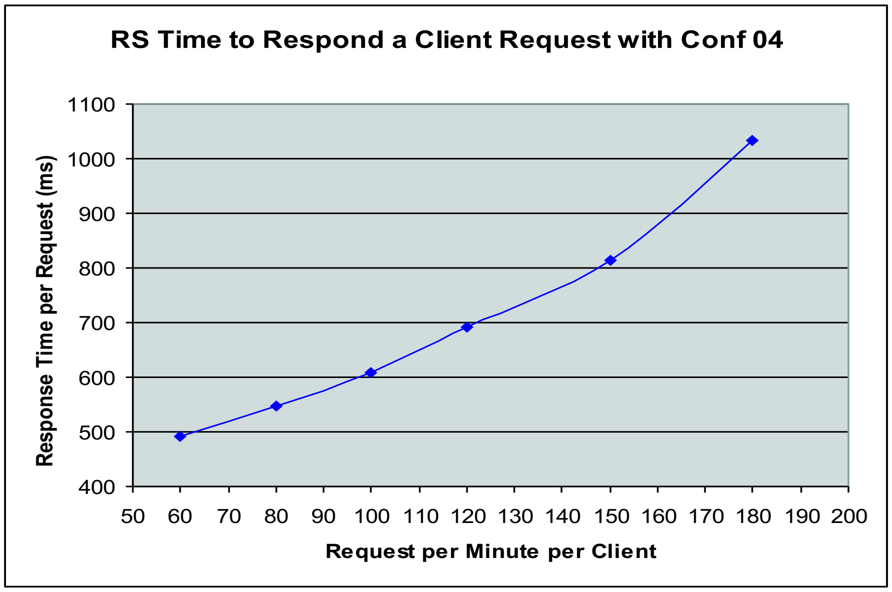

Figure 5.22: Test 3 - Configuration #4, This is the graph for table 5.9

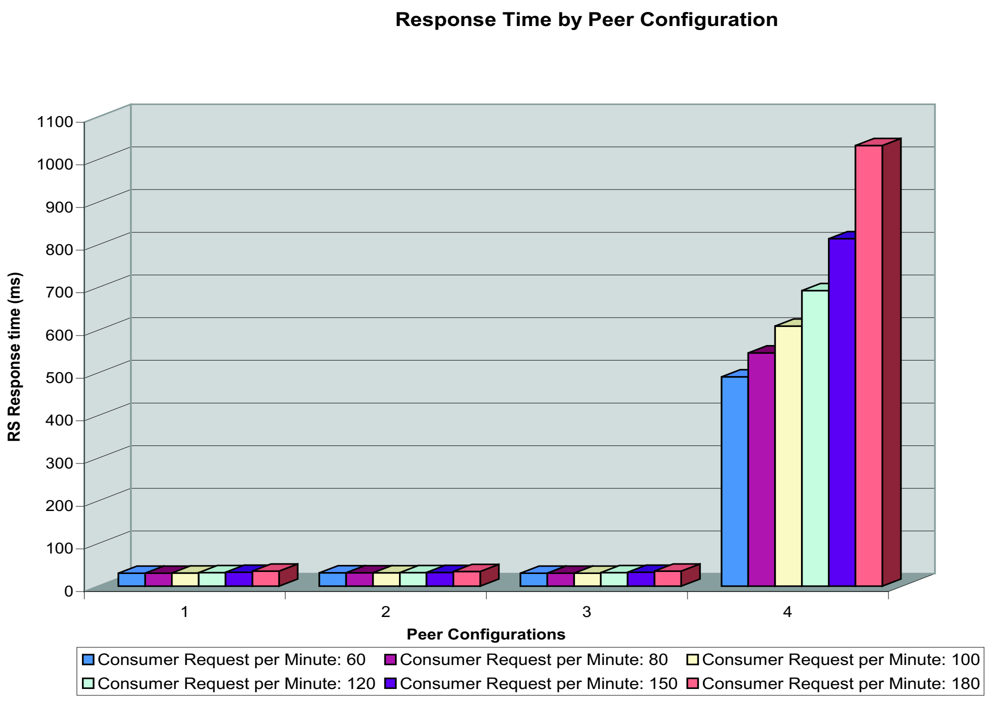

Figure 5.23:

CHAPTER 6

Conclusion and Future Work

In this thesis we presented the disadvantages that existing middlewares have to cope with environments that contains mobile devices. Since these devices have become so popular and so powerfull in recent years, we propose to use them as data sources, and integrate them with more traditional data sources. We present NetTraveler, a database middleware system designed to cope with dynamics wide-area environments and integrate mobile devices as reliable data sources in a computer network. Then we introduced the Registration Server, one of NetTraveler's main components. The RS is a web service located at cooperative local area networks that works together to keep track of moving data sources and route query request as appropiate. We presented our RS prototype and provided the results to the different tests that we used to verify that our proposed solution has the capabilities to cope with the moving data sources in a computer network.

6.1 Research Conclusion

We have presented the Registration Server for NetTraveler, with the Registration Server we were able to create ad-hoc federations of cooperative sites and track mobile devices as they change locations and move across different local area networks. The resources and data that can be shared are not limited by the RS in any sense. The RS will only coordinate the access point between the member requesting the service and the member offering the service, making an infinite numbers of possibilities for this system to be used. Since the system has the ability to track mobile devices and their current location, it allows users to have a more flexible work environment. Thus, users can rely that their data and crucial resources will be available when and where they need them.

6.2 Future Work

With our Registration Server prototype we provided basic functionality to prove the ability of tracking mobile devices and redirecting resources requests. However there is still a lot of work that can be done with it. We can add messages encryption to provide a higher security level in the RS-RS and RS-Member communication protocols. Also we can add an evaluation system to evaluate the RS performance answering querys and routing requests. We can assign a level of trust to each RS based on their performance. In this way, members and peers RSs can take that level into consideration when dealing with this RS. This will add another level of dependability to the RSs. The same idea can be apply to members. In this way RSs and others peer members can use this trust level to have an idea how good or beneficial the iteration with a device can result. Also administrators can use it to take diciplinary actions to members or RSs in case that they does not keep at least a minimun allowed trust level. Finally we can keep record of member's online-offline and locations patterns so we can apply artificial intelligence methods to make a guess of the users status and location in case the RSs can not provide the correct information from their peers iteration.

6.3 Final Word

Mobile devices have been previously considered only from the client side. Due to the many limitations presented by these devices we still can not take them as reliable data sources. However, in the two last years this new technology has made a huge improvement and while there still some limitations, there are some capabilities and advantages that we can use in our favor. NetTraveler try to exploits these new advantages.

BIBLIOGRAPHY

[1] C. Narayanaswami, N. Kamijoh, M. Raghunath, T. Inoue, T. Cipolla, J. Sanford, E. Schlig, S. Venkiteswaran, D. Guniguntala, V. Kulkarni, and K. Yamazaki. IBM's Linux watch, The Challenge of Miniaturization. *IEEE Computer*, 35(1):33 – 41, January 2002.

[2] J. Li, C. Blake, D. De Couto, H. Lee, and R. Morris. Capacity of Ad Hoc Wireless Networks. *In 7th Intl. Conf. on Mobile Computing and Networking*, pages 66 – 69, July 2001.

[3] Oracle Corporation. Oracle Transparent Gateways.
URL: http://www.oracle.com/gateways/html/transparent.html.

[4] Sybase Corporation. Data Access Migration.
URL: http://www.sybase.com/products/entcon/daccess.html.

[5] R. Ahmed, P. De Smedt, W. Du, W. Kent, M. Ketabchi, W. Litwin, A. Rafii, and M. Shan. The Pegasus Heterogeneous Multidatabase System. *IEEE Computer*, 24(12):19 – 27, December 1991.

[6] S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, and J. Widom. The TSIMMIS Project: Integration of Heterogeneous Information Sources. *In IPSJ Conference*, 1994.

[7] A. Tomasic, L. Rashid, and P. Valduriez. Scaling Heterogeneous Databases and the Design of DISCO. *In 16th ICDCS Conference*, 1996.

[8] J. Enseñat and M. Rodríguez. Design of the Registration Server for NetTraveler Middleware System. *In 7th Intl. Conf. on Internet and Multimedia Systems and Applications*, pages 311 – 316, August 2003.

[9] Andy Oram, editor. *Peer-to-Peer: Harnessing the Power of Disruptive Technologies.* O'Reilly, March 2001.

[10] Gnutella. Gnutella.com.
URL: http://www.gnutella.com.

[11] eDonkey. edonkey2000 user's guide.
URL: http://www.edonkey2000.com/documentation/index.html.

[12] BitTorrent. Brian's BitTorrent FAQ and Guide.
URL: http://dessent.net/btfaq.

[13] Todd Sundsted. The practice of peer-to-peer computing: Introduction and history.
URL: http://www-106.ibm.com/developerworks/java/library/j-p2p.

[14] S. Gribble, A Halevy, Z Ives, Maya Rodrig, and D. Suciu. What Can Databases Do for Peer-to-Peer? *In 4th Intl. Workshop on the Web and Databases (WebDB'01)*, June 2001.

[15] World Wide Web Consortium (W3C). HTTP - Hypertext Transfer Protocol.
URL: http://www.w3.org/Protocols.

[16] World Wide Web Consortium (W3C). SOAP Specification.
URL: http://www.w3.org/TR/SOAP.

[17] World Wide Web Consortium (W3C). Web Services Description Language (WSDL).
URL: http://www.w3.org/TR/wsdl.

[18] R. Finkelstein. The new Middleware. *SIGMOD Record*, 24(4):102 – 106, December 1995.

[19] S. Ho, S. Itoh, S. Ihara, and R. Schlichting. Agent Middleware for Heterogeneous Scientific Simulations. *In 10th Intl. Conf. on High Performance Networking and Computing*, pages 1 – 7, 1998.

[20] T Phan, R Guy, and B. Rajive. A Scalable, Distributed Middleware Service Architecture to Support Mobile Internet Applications. *In 1st Workshop on Wireless Mobile Internet*, pages 27 – 33, July 2001.

[21] P. Bernstein. Middleware: A Model for Distributed Services. *Communications of the ACM*, 39(2):86 – 97, February 1996.

[22] J. Deutsch. The Evolution of Customer Middleware Requirements. *In 3rd Intl Conf. on Parallel and Distributed Information Systems*, pages 262 – 263, September 1994.

[23] M. Rodríguez-Martínez and N. Roussopoulos. MOCHA: A Self-Extensible Database Middleware System for Distributed Data Sources. *In 2000 ACM SIGMOD International Conference on Management of Data*, 29(2):213 – 224, May 2000.

[24] M. Stonebraker, P.M. Aoki, R. Devine, W. Litwin, and M. Olson. Mariposa: A New Architecture for Distributed Data. *In 10th Intl Conf. Data Engineering*, pages 54 – 65, 1994.

[25] M. Rodríguez-Martínez and N. Roussopoulos. Automatic Deployment of Application-Specific Metadata and Code in MOCHA. *In 7th Intl. Conf. on Extending Database Technology*, pages 65 – 89, March 2000.

[26] Object Management Group. CORBA Services: Common Object Service Specification. OMG Technical Document formal 98-12-31.

[27] W. Adjie-Winoto, E. Schwartz, H. Balakrishnan, and J. Lilley. The Design and Implementation of an Intentional Naming System. *In 17th ACM Symposium on Operating Systems Principles*, pages 186 – 201, December 1999.

[28] Microsoft Corporation. Universal plug and play (upnp).
URL: http://www.upnp.org.

[29] Apple Computer Corporation. Rendezvous Technology.
URL: http://www.apple.com/macosx/jaguar/rendezvous.html.

[30] K. Sollins and L. Masinter. Functional Requirements for Uniform Resource Names. *Internet Engineering Task Force RFC 1737*, December 1994.

[31] Sun Microsystems. Java Technology and Web Services.
http://java.sun.com/webservices/index.jsp.