Rheinisch-Westfälische Technische Hochschule Aachen

Lehrstuhl für Informatik V
Prof. Dr. Matthias Jarke

Mobile Web Service Discovery in JXTA/JXME

Master Thesis

Adem Toprak
Matriculation number: 248123

December 4th 2006

First Supervisor:	**Prof. Dr. Matthias Jarke** **Lehrstuhl für Informatik V, RWTH Aachen**
Second Supervisor:	**Prof. Dr. Wolfgang Prinz** **Lehrstuhl für Informatik V, RWTH Aachen**
Advisor:	**M. Sc. Satish Narayana Srirama** **Lehrstuhl für Informatik V, RWTH Aachen**

Statement

Hiermit versichere ich, dass ich die vorliegende Arbeit selbständig im Rahmen der an der RWTH Aachen üblichen Betreuung angefertigt und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

I guarantee that this thesis is done independently, with support of the Informatik V department at RWTH Aachen University and no other unmentioned helping resources are used.

Aachen, November, 2006

(Adem Toprak)

Abstract

The next generation devices like smart phones, PDAs and other communication gadgets are quickly filling up the market today, creating endless possibilities for wireless communication. Demand for related software applications is skyrocketing as well. Furthermore, recent developments in mobile communication technologies like GPRS/EDGE/UMTS has significantly increased the wireless data transmission speed. It has made web services usage a practical reality. In addition, it enables cellular domain to act as a service provider as well.

Recently, the main focus of research in cellular domain has shifted to “Mobile Terminals as Web Service clients”. Basically, Web Services are applications on the web that can be accessed and utilized by other applications or Web Services to perform various tasks and processes. Although Web Services are gaining rapid acceptance, the policy of web service discovery has become a wearisome hindrance in proper implementation of its usage on the internet. The traditional centralized UDDI registries will not adapt efficiently to the large number of services that will be provided by the Mobile Hosts.

Another technology that has gained popularity along side wireless communication, in recent years as a low cost individual computing, is peer-to-peer communication. This technology takes advantage of resources like storage, cycles, content, human presence. P2P technology transports today communication skill to a higher dimension by using the mentioned advantages.

Therefore, the purpose of this thesis work is to combine these two virtuous technologies and introduce the best peer-to-peer solution for the Web Services discovery on mobile and normal hosts. It will provide a solution to merge Web services and P2P technology on mobile and other resource constrained devices. Furthermore, a proposal to establish a mobile network among web services providers and consumers via P2P technology will also be presented. The aim of this network will be to develop a distributed service discovery mechanism. JXTA's P2P provides perfect solution for service (Web Service) discovery and communication among mobile users as “peers”.

Table of Contents

Table of Contents **7**
1. Introduction **9**
1.1. Web Services and P2P 9
1.2. Document Outline 10
2. State of the art **13**
2.1. Web Services 13
2.1.1 Web Services Standards 13
2.1.1.1. XML and kXML 13
2.1.1.2. SOAP and kSOAP 14
2.1.1.3. WSDL 14
2.1.1.4. UDDI 15
2.2. Peer-to-peer technologies 16
2.2.1. Introduction to P2P 16
2.2.2. Development of P2P 17
2.2.2.1. First Generation; Centralized Systems 18
2.2.2.2. Second Generation; Decentralized Systems 19
2.2.2.3. Third Generation; Centralized Decentralized Systems 20
2.2.3. Current P2P architectures in mobile environment 21
2.3. Java Platform, J2ME and MIDLets 22
2.3.2. Configurations at J2ME 23
2.4. Introduction to Project JXTA 25
2.4.1. JXTA Architecture 26
2.4.2. JXTA Virtual Network 27
2.4.2.1 JXTA identification ID's 27
2.4.2.2. JXTA Advertisements 28
2.4.3. Rendezvous Super-Peers 29
2.4.4. Relay Super-Peers 29
2.4.5. JXTA Protocols 30
2.5. Introduction to Project Lucene 32
2.5.1. What Lucene can do? 32
2.5.1.1. Lucene Background 33
2.5.2. Indexing and searching 33
2.5.2.1. What is indexing, and why is it important? 33
2.5.2.2. What is searching? 34
2.5.2.3. Creating an index 34
2.5.2.4. Indexing an object 34
3. Mobile WS-Discovery Design **37**
3.1. Mobile Web Service Provider 37
3.2. Problem Domain 38
3.2.1. Problem Description 38
3.2.2. Proposed Solution 38
3.2. Comparing Web Services and JXTA 39
3.3. Combining Web Services and JXTA 41
3.3.1. JXTA-SOAP Model [45] 43

3.3.2. Proxy Model 44
3.3.3. Port Forwarding Model 45
3.4. Searching Web Services and client application in JXTA 46
3.5. Putting All Together 48
4. Mobile WS-Discovery Implementation 51
4.1. Development Tools/Platforms 51
4.1.1. NetBeans 51
4.1.1.1. NetBeans Mobility Pack [41] 51
4.1.1.2. NetBeans Profiler [42] 52
4.1.1.3. Sun Java Wireless Toolkit 52
4.1.2. Eclipse 53
4.2. Getting started with JXTA Shell 54
4.3. WS-Discovery Application Development 55
4.3.1. Service Provider Application 55
4.3.2. JXTA Proxy/Relay 59
4.3.2.1 Searching at JXTA proxy/relay 59
4.3.2.2 Deep Search Mechanism at JXTA proxy/relay with Lucene 61
4.3.2.3 Deploying Web Service 63
4.3.2.4 Sequence diagram of the Web Service Search 63
4.3.3. JXME Mobile 64
4.3.3.1. JXME Client Mobile Application 66
4.3.3.2. Invocation through Pipes (port forwarding model) 67
5. Conclusion 69
6. Future Work 71
List of Figures 72
List of Examples 74
Appendix – Shell and Mobile Application images 75
Literature 77

1. Introduction

In recent years, mobile devices such as hand-held PCs, personal digital assistants (PDAs), and smart cellular phones have evolved rapidly. Smart cellular phones encapsulate computer technology so fast with respect to processor power, memory and communication channel, such that it can do almost everything that a normal personal computer does. Before we were considering cellular phones as a client side technology as a consumer. Now we achieved one step more and we started considering cellular phones as a mobile service provider or a server.

Web Services are emerging as a dominant paradigm for distributed computing in industry. Web Services are enterprise applications that exchange data, share tasks, and automate processes over the Internet. They are designed to enable applications to communicate directly and exchange data, regardless of language, platform and location. A typical Web Service architecture consists of three entities: service providers that create and publish Web Services, service brokers that maintain a registry of published services and support their discovery, and service requesters that search the service broker's registries. A detailed discussion of Web Services is provided in Section 2.2.

Mobile Host [1], a light Web Service provider, built on top of a normal Web server, was developed for mobile phones. Further information of this is available at section 3.1. "Mobile Web Services Provisioning". Combining Web Services with Mobile technology brings us a new trend. Web Services has a broad range of service distributions, on the other hand cellular phones has great amount of users and it is increasing day by day. Consuming as well as providing Web Services in cellular phones gives us high range of application areas. For instance Mobile photo album service, allows end users to share photos rapidly and easily. Further application areas could be found at [1].

1.1. Web Services and P2P

Using traditional Web Services standards for mobile web service discovery brings out some drawbacks. Lets examine briefly how UDDI [4] works; service providers define their interfaces with WSDL [3] and publish the WSDL to service registry UDDI, so that user can find them easily. At this situation service registry plays a big role on service discovery. UDDI a master directory for all public web services, which keeps WSDL files in a centralized server. By introducing mobile web service provider and consumers into the Web Services market, the amount of Web Services will increase. The increasing number of Web Services will lead to difficulties on; discovering of exact services, up to date services, and quick response. Moreover Centralized registries are performance bottlenecks and may result in single points of failure.

Mainly Web Services built for static networks. When we consider service discovery and registry for mobile web service provider, we face problems with centralized UDDI registry. Mobile networks are dynamic due to node movement. Nodes can join or leave network at any time, they can switch from one operator to another operator over the network. Keeping up to date information of the published services in central registries is really difficult.

By involving Web Services on mobile network devices we had stated drawbacks. We had to find a solution to stated problems. Peer-to-peer is a distributed computing model where peers act as clients and server, there is no central server managing the network. Nodes in peer-to-peer architecture are decentralized and distributed. Peer-to-peer technology can provide alternatives for service discovery and communication. We have made a detailed survey among peer-to-peer technology and found out which fits best to our thesis work. As a result we decided to use JXTA platform [5] as a peer-to-peer solution to our thesis project. The currently available P2P systems tend to use protocols that are proprietary and independent of other networks, incapable of leveraging their services. This problem was solved by the project JXTA which provided a common P2P platform that is platform and language independent.

Web Services and P2P technologies have emerged from different problem domains. Both technologies solve the challenge of connecting consumers and providers of services across the internet. However, the problem that must be solved lies in an open communication protocol [6].

The objective of this project is to provide a model for web service discovery and invocation in the JXTA P2P framework that will solve the problem of service invocation in addition to discovery. The idea is that a JXTA Group service could be transparently implementable as a web service or alternatively, given a reachable web service, it should be able to invoke from inside a JXTA Group. This should be transparent to the user without the knowledge that this particular service is a web service. This means that this service should be discovered, located, invoked similar to any other JXTA service.

This needs two things to be enabled for implementation.

- A way to expose the definition of the web service (for example a WSDL), which fits into the module advertisement framework so that no special platform upgrade is needed.

- A way to invoke this service, either as the dynamic client proxy generation capabilities of some of the SOAP toolkits such as Systinet WASP [7] or a precompiled static client proxy that would have its class name in the Module Implementation Advertisement.

The only other requirement would be the existence of the SOAP client library supporting classes in the class path. Basically, it is about taking a client proxy and incorporating it into the JXTA module advertisement framework and noting that the real is provided by a Web service.

1.2. Document Outline

Section 2 describes the State of Art of different technologies used in this study. Initially a brief description of Java Platform and J2ME along with configurations and profiles explained for restricted devices is provided. Later on Web Services and its components like (SOAP, WSDL, UDDI, etc.) are explained. The section also describes detailed survey made among peer-to-peer technologies, to find best p2p solution to our described problems. Moreover detailed information given about project JXTA. Finally Lucene search technology is introduced and explained with examples.

Section 3 gives a brief information about project “mobile Web Services provisioning” and the problem description and proposed solution stated for discovering services deployed on these Mobile Hosts. A detailed comparison of Web Services and JXTA is made and three alternative models are analyzed to combine Web Service and JXTA. In addition abstract information given about searching Web Services and client applications.

Section 4 gives a detailed explanation of the implementation part pf this thesis. Initially development tools are introduced like NetBeans, Eclipse. Then JXTA shell is described. Finally WS-Discovery Application development is explained with detail. Moreover searching and deeps searching mechanisms analyzed and explained with detail. JXME client mobile and invocation through pipes are examined.

Section 5 gives a conclusion to the thesis project, and it shows how it succeeds to its goals.

Section 6 provides information to the extension of this thesis project and provides some ideas for future work.

2. State of the art

2.1. Web Services

Before going to describing of web services we should keep in mind this thesis work is only an enhancement of a part of the thesis work "Concept, implementation and performance testing of a mobile Web Service provider for Smart Phones". For detailed information you can refer [27].

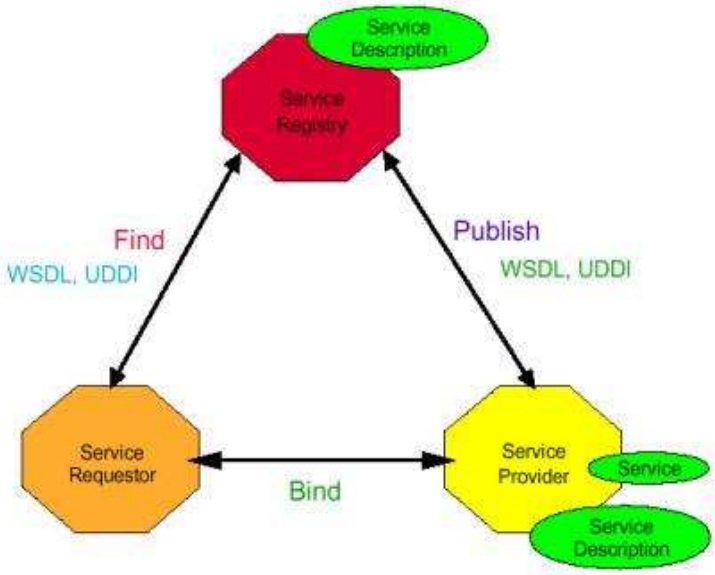


Figure 1 : Web Services architecture [28].

Web Services represent the convergence between the service-oriented architecture (SOA) and the Web, using the XML, SOAP, WSDL and UDDI open standards over an Internet protocol backbone. XML is used to tag the data, SOAP is used to transfer the data, WSDL is used for describing the services available and UDDI is used for listing what services are available. This thesis work focuses on UDDI, developing P2P distributed UDDI mechanism. Web Services stands with interaction of three modules as shown at Figure 1, these modules described as below:

- **Service Provider:** Service provider supply services for requestors (clients). Service provider describes service definitions, and registers it to the service registry, and defines how to access to this service.
- **Service Requester:** Service requesters ask and use services from service provider. Service requester searches and requests services and service's parameters from Service registry.
- **Service Registry:** Stores description files of services and makes it available for search. Service provider or requestor can search registry for service information. Service registry keeps information about services call parameters.

2.1.1 Web Services Standards

2.1.1.1. XML and kXML

Extensible Markup Language (XML) is a text based markup language designed for more flexible and adaptable information identification. It is a metalanguage which lets definition of portable structured data, definition of data descriptive languages such as interchange formats and messaging protocols. Therefore XML is considered as the common language to be used between cross-platforms of companies to get free from system dependent data representation. It uses Document Type Definition (DTD) or XML Schema to describe the data and uses namespace to distinguish between duplicate element types and attribute names. Although it turns to be a standard for electronic commerce, XML has its own drawbacks. Using XML instead of binary data makes the size of transmitted data 4 times bigger, and parsing the XML information also consumes CPU time. Hence it uses the system resources like bandwidth, processing, storage etc in a greedy way. Since research is concerned with resource restricted devices like smart phones, we are going to use an alternative for the standard XML which is called kXML. It is a small XML pull parser, specially designed for constrained environments such as Applets, Personal Java or MIDP devices.

2.1.1.2. SOAP and kSOAP

SOAP is a lightweight protocol designed for exchanging structured information in a decentralized, distributed environment. SOAP is a communication protocol that is designed especially for communication through applications. It is based on XML technologies to enable an extensible messaging framework for message construct that can be exchanged over various protocols. By using HTTP and SMTP SOAP sends messages via internet which is able to go through proxies and firewalls in a language and platform independent way. SOAP is first designed by leading IT companies including Microsoft, SAP, and IBM then offered to W3C for standardization. As far as mobile devices are concerned fro Web Services SOAP, kSOAP is used for the mobile terminals with limited resources, as the parser should have a small memory footprint. kSOAP is a SOAP web service client library for constrained Java environments such as Applets or J2ME applications, based on kXML.

SOAP consists of 3 main parts:

- SOAP envelope defines a framework for expressing what is in a message and who should deal with it.
- SOAP encoding rules to exchange application specific data types.
- SOAP Remote Procedure Call representation.

A SOAP message is an ordinary XML document consists of mandatory SOAP Envelope, optional SOAP Header and mandatory SOAP Body. Envelope is the root element of the XML message. Header is used for application specific information and for adding features for processing of message. Finally body of the SOAP message contains the information intended for the receiver of the message.

2.1.1.3. WSDL

Web Services Definition Language (WSDL) is an XML based “common language” for describing web services. It provides the description of the format of request and response messages and provides an abstract language for definition of what the particular web service

does, with its functions, parameters and data types. It also defines location and binding details of the service.

In other words a web application (service requester) that is about to use a web service provider should know about the functionality, output of that service, so that it can give decision about whether that service is what it needs, on the other hand the web service needs to state the expectations, inputs to fully satisfy and to fulfill the task expected by web application.

WSDL uses the following elements to describe the web services:

- Types: This element is for defining the data type which is used by the web service.
- Message: This element defines the data elements of an operation being used. It can consist of one or more parts. It can be compared to the parameters of a function call.
- PortType: This element of WSDL describes which operation web service performs, and states the messages used. These operations can be one-way that receives only a message without a response, two-way that receives a message and also gives a response.
- Binding: It specifies the protocol details and message format of a PortType. One attribute of this element points to the unique URL, so called Port for binding.

2.1.1.4. UDDI

UDDI stands for Universal Description, Discovery and Integration, which is a specification that enables implementation of a service broker. Its mission is realizing a market where the businesses will register their services and search for the services they need. It is a platform independent, XML based directory of web services, enabling a business to describe, to discover offered services and integrating with those other businesses` web services by communicating through SOAP. With UDDI Businesses can also advertise, state what they are doing in which industry. UDDI presents 3 types of information about the web services; “white pages” that includes the contact details of the company, “yellow pages” that provides categorization according to the business and service type and finally “green pages” that holds technical data about the service.

UDDI is one of the core Web Services standard components. It is designed to be interrogated by SOAP messages and to provide access to WSDL documents describing the protocol bindings and message formats required to interact with the web services listed in its directory. UDDI holds a detailed description of WSDL Web Service, it reference to WSDL file or keep it at a local repository. UDDI keeps WSDL information by different kind of categorization; this can be numerical category codes which are available at internet.

The information that makes up a registration consists of five data structure types. This separation by information types provides simple partitions to assist in the rapid location and understanding of the different elements of a registration.

Business Entity: The Business Entity structure represents the basic information of the business. The information includes contact information, categorization, identifiers, descriptions, and relationships to other businesses of the service provider. The UDDI also allows companies to

establish relationships with one another. Even in such a case, both the companies should have their respective Business Entity structures.

Publisher Assertion: The Publisher Assertion structure is used to establish public relationships between two Business Entity structures. A relationship between two Business Entity structures is visible to the public only when both companies have created the same assertion with two separate Publisher Assertion documents independently. Thus, a company can claim a business relationship only if its partner asserts the same relationship.

Business Service: A Business Entity contains one or more Business Service structures. A Business Service represents a single, logical service classification. A <businessService> element is used to describe a set of services provided by the business. The description of the Business Service includes information like how to bind the Web Service, type of the Web Service etc.

Binding Templates: A Business Service contains one or more Binding Templates. A Binding Template contains the technical descriptions of the Web Services represented by the Business Service structure. It also contains the access point URL of the Web Service, but does not contain the service specification details. It is similar to the <service> element of the WSDL described earlier.

TModels: A TModel, the <tModel> element, is an abstract description of a particular specification or behavior to which the Web Service adheres. For example a TModel can be defined to represent a portType defined by the WSDL. Then a business service implementing the portType can be specified by associating the TModel with one of the binding templates of the business service.

2.2. Peer-to-peer technologies

2.2.1. Introduction to P2P

"P2P is a class of applications that takes advantage of resources--storage, cycles, content, human presence--available at the edges of the Internet."[9]

A peer-to-peer (or P2P) connection, as understood by its name, it is the communication of two or more internet enabled computers, to share data among each other. Mainly P2P networks used for connecting nodes by means of largely ad hoc connections. This type of networks can be used for plenty purposes. Now a days there are too many application which uses P2P technology, for instance ICQ[10], Skype[11], Morpheus[12].

- *Server*, An entity that serves requests to other entities but does not initiate requests.
- *Client,* An entity that initiates requests but is not able to serve requests.

Peer-to-peer networks acts like client and server, in pure P2P network they combine this two term. There are too many different architectures and designs of P2P systems. In instance some of the networks are centralized, like Napster, they have a central server to let nodes connect directly. On the other hand there are some networks which are decentralized distributed like Gnutella[13].

We are going to examine each of the architectures advantages and disadvantages. Finally we will decide for the architecture which best fits to our thesis project.

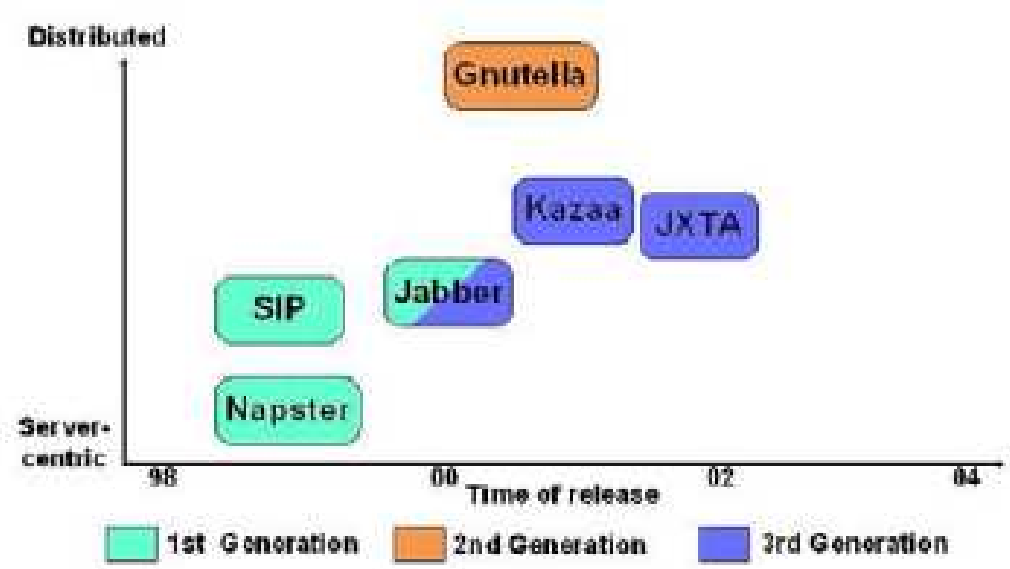


Figure 2 : Generation phase of peer-to-peer technologies [15].

2.2.2. Development of P2P

The first P2P networks was developed by the U.S. Defense department in a project called ARPANET, in which the computers involved were connected directly together. The development of P2P architecture composed of there generations. The first generation is the one that made P2P a visible option to millions with the development of Napster[14]. This first generation P2P used a centralized file list containing all the files on a connected user's hard drive. Due to the centralized file list, Napster was easily shut down. The second generation which includes Gnutella used a de-centralized network. Unlike Napster, Gnutella would connect users directly to a group of other users and so on. The third generation was much like the second, but with improvements made on architecture like semi de-centralized.

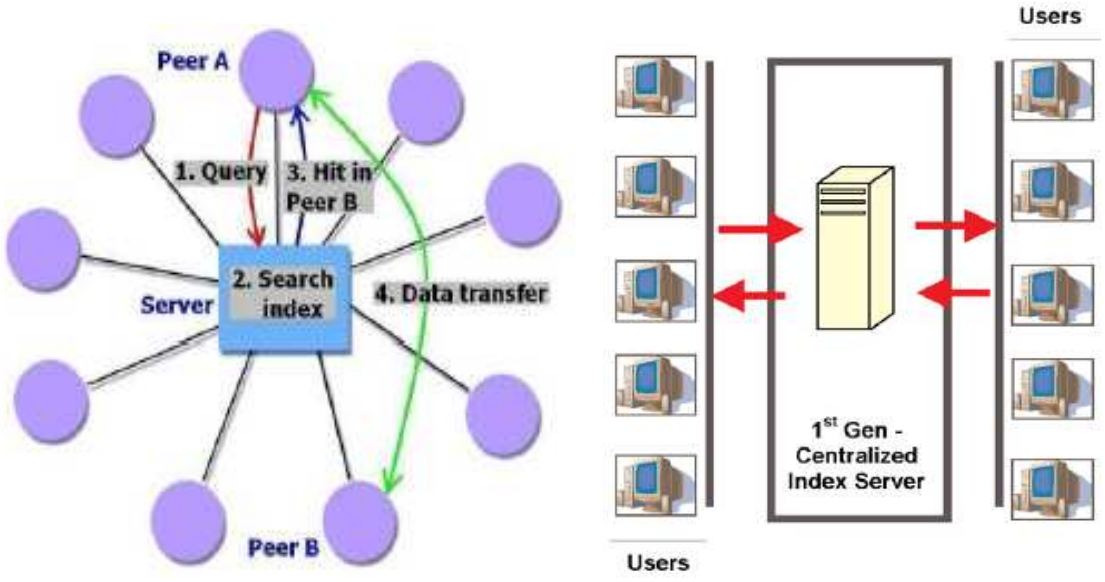


Figure 3 : Demonstrates the operating principle of Napster and centralized systems [15,20].

2.2.2.1. First Generation; Centralized Systems

The initial steps of P2P generation stand to development of Napster by Shawn Fanning in 1999. His aim was to develop an easy music sharing application. The architecture was quite simple. There is centralized server for maintaining an index of the connected peers. When the peers (clients) want to find a song they are connecting server and querying server index for the specified file. When there is a user who own that queried song, IP address of that user is sent back to the peer. Final step is to establish a peer to peer connection with the user who owns the song. With development of internet speed Napster become very popular around the world in sharing music files between millions users. However this popularity made music groups uncomfortable. Napster forced to shut down.

At the early days of computing centralized servers aimed to serve inexpensive terminals, later on this idea shifted to different purposes. As we seen at the architecture of Napster, centralized systems consist of a central server which directs traffic among registered users (peers). The biggest advantages of centralized architecture are easy control and organize. Moreover when we consider this architecture for mobile web service discovery mechanism, it keeps up to date service information on the server, and it makes query processing easily with low overhead. On the other side central servers have single point of failure. Moreover centralized systems produce giant communication traffic and storage on server. These drawbacks yield us to search for other architectural solutions.

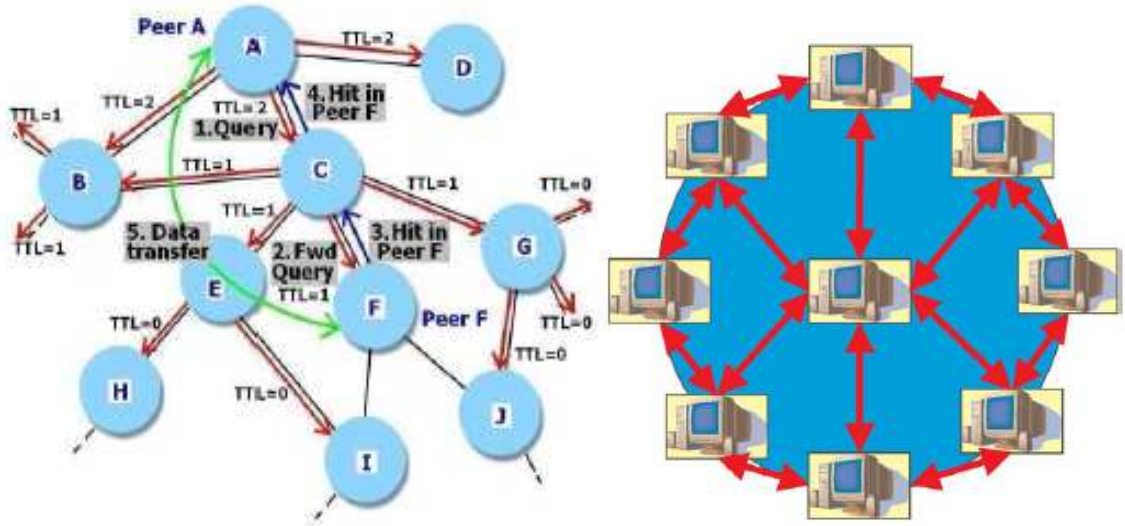


Figure 4 : Demonstrates the operating principle of Gnutella and Decentralized systems [15,20].

2.2.2.2. Second Generation; Decentralized Systems

The second generation of P2P is completely decentralized, this means that there is no more client server architecture; there is a combination of server and client which are called *Servents*. Gnutella aforementioned as a second generation of P2P architecture. Gnutella bring out reforms to P2P architecture, as Napster closed due to its legal issues and its server centric architecture. The Gnutella software initiates communication with other users by using different kind of methods. Frequent used way is using pre-existing list of possibly working node address which is embedded inside the application code. This list can be extend by adding new host IPs. Other methods are using web forms or IRC to share working node IP among users. Once initial connection established, the user gets node list from connected host. Now the user will try to connect to the nodes at the list. This expansion will go on until it reaches a certain user-specific quota.

When Gnutella user wants to do a search, it sends request query to the Gnutella network through direct connected hosts. The host, who gets the request, has the ability to search the query locally and at the direct connected hosts' directory. If the node that sent the search request is not firewalled, the node with the result directly returns the result to the requestors' IP. If the node that sent the search request is firewalled (many are), then the result is (indirectly) routed back along the route the search was received on. After the result is returned, they negotiate the file transfer and the transfer proceeds. When the search is not found on the host the query is forwarded to connected peers (host), this spreading operation continues step by step among the peers. As the query reaches to one host defined value TTL (time to live) is decreased by that peer. When TTL reaches to zero spreading operation stops, so that possibility of infinite loop is eliminated. Finally, when a user disconnects, the client software saves the list of nodes that it was actively connected, for use next time it connects.

The biggest advantage of this architecture is that the peer behaves both as a server and client. Gnutella is so decentralized; it is really difficult to shut down the network with respect to

Napster, where the entire network relied on a central server. On the other hand search function on Gnutella is unreliable, the search my not discover entire network.

2.2.2.3. Third Generation; Centralized Decentralized Systems

The third-generation P2P architectures, like eDonkey and Bit Torrent, attempt to solve the problems of the earlier generation architectures. In reality it is not more then a mixture of the two first generations, it is a new model which combines previous architecture. The third-generation architectures have introduced a new approach to the peers, which is called super-peers. Peers are light end-user peers, whereas super-peers are on a higher level in the hierarchy, working as relays for peers and other super-peers. The role of the super-peers is quite similar to the servents in the 2nd generation architectures, but the functionality is very different. Third generation P2P networks also made enhancements to improve their ability to deal with large numbers of users. The regular peers, so-called *edge-peers* (peers) use super-peers as a gateway to the P2P network. As many functions as possible are left to be handled in super-peers. Super-peers are also used for NAT and firewall traversal. Our example of third generation P2Ps is open-source JXTA development framework [5]. JXTA platform was originally conceived by Sun Microsystems Inc. and designed with the participation of experts from academic institutions and industry. The development of JXTA started in 2001 and is still going on. In a nutshell, JXTA establishes a virtual network on top of the IP or non-IP networks, hiding the underlying protocols. It uses XML messages, and is thus independent of the software and hardware platform. Each JXTA peer has its own logical, network independent ID. Peers organize automatically or manually into peer groups that are either protected private- or public groups of peers that are visible to each other. Peer group is the base unit of JXTA, and basically everything happens within them, which also considerably limits the load to underlying networks.

JXTA dynamically uses either TCP or HTTP protocols to traverse network barriers, like NATs and firewalls. A JXTA network consists of peers (*edge-peers*) and super-peers (*rendezvous peers* and *relay peers*). A peer with enough privileges can become a super-peer depending on its location in the network. When a peer joins a JXTA network, it finds, either manually or automatically, the closest rendezvous peer and creates a special relationship with it. From this moment, the edge peer starts using the rendezvous peer as a gateway to the P2P network. The rendezvous peer maintains a list of its edge peers and their shared resources. Rendezvous peers organize themselves into a loosely coupled network, delivering queries and peer information between each other. Rendezvous peers use DHTs (Distributed Hash Table) for optimizing peer and service discovery. In this respect, JXTA differs a lot from Gnutella's style of broadcasting queries to neighbor peers. Actually, DHTs were already used in some advanced second-generation protocols. JXTA introduces also the *relay peer*s that can route JXTA messages and data between peers that have no direct connection between each other. Relay peers are used also in spooling messages for unreachable or temporarily unavailable peers. With the functions mentioned above, JXTA in practice allows any peer to reach any other JXTA peer independently of its network location.

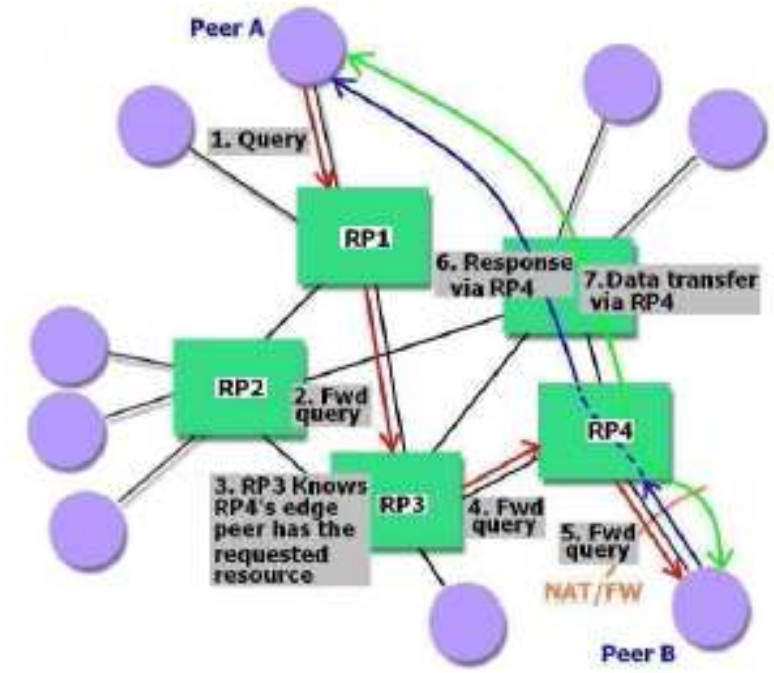


Figure 5 : Demonstrates the operating principle of JXTA [15].

Searching for resources in JXTA is illustrated in Figure 5. A peer (for example A) is requesting a resource, which is in this case in another peer located in the network behind NAT. When querying a resource, A sends a query to its rendezvous peer (for example RP 1) (1.). RP1's index does not contain the requested resource, so it relays the query to its own rendezvous peer RP3 (2.). RP3's index contains the requested resource with the information that the resource is available some of in RP4's edge peers (3.), so RP3 relays the query to RP4 (4.). RP4 knows the resource is in its edge peer B, so it relays the query to B (5.). Because B is in a network using NAT, it sends the response to A via its relay (and rendezvous) peer RP4 (6.). Then, the data is transferred between A and B using RP4 as a relay (7.). The third generation has solved many of the problems of the earlier generations. The hierarchical model has effectively decreased the stress caused to the underlying network protocols when compared to the second generation. Third-generation protocols also provide new services, like peer groups and NAT/firewall traversal.

2.2.3. Current P2P architectures in mobile environment

The current third generation P2P architectures have matured to the point where they work rather well and the overhead inflicted to the network has decreased from the earlier generation architectures. However, this applies only for desktop and laptop environments with wideband Internet connections, and high processing and memory capacity. All the currently available P2P protocols have been designed with a desktop environment in mind, and thus there are no any well known third-generation protocols designed especially for mobile devices. The principle of third-generation architectures as such is suitable for mobile use. Although there are many advantages in using third generation protocols in a mobile environment, there are also drawbacks. Current third-generation protocols, like JXTA, are too heavy for effective mobile use. As a response to this problem, the JXTA community has developed a light version of JXTA for mobile devices, called JXME (JXTA for J2ME). It works in all MIDP devices, e.g. Nokia's Series 60 phones.

JXME has two versions: proxyless and proxied. The proxyless version works similarly to native JXTA, whereas the proxied version needs a native JXTA peer to be set up as its proxy. The proxied version is also lighter than the proxyless one, since it uses binary communication with its proxy, whereas the proxyless version uses XML-based communication. The proxied version cannot work as a super-peer either. Unfortunately, JXME works currently only in Java environment and there are no JXTA versions yet ported to native Symbian C++ language [15].

2.3. Java Platform, J2ME and MIDLets

The Java Platform is the name for a computing environment, or platform, from Sun Microsystems which can run applications developed using the Java programming language and set of development tools. Java has three kinds of versions:

- Java 2 Standard Edition (J2SE)
- Java 2 Enterprise Edition (J2EE)
- Java 2 Micro Edition (J2ME)

These versions developed for different environments and for different devices. J2ME is a collection of Java APIs for the development of software for resource constrained devices such as PDAs, cell phones and other consumer appliances. The intention of J2ME is to provide common functions to the different capability and ability devices. For this purpose modular structure is developed, different kind of profiles and configurations defined.

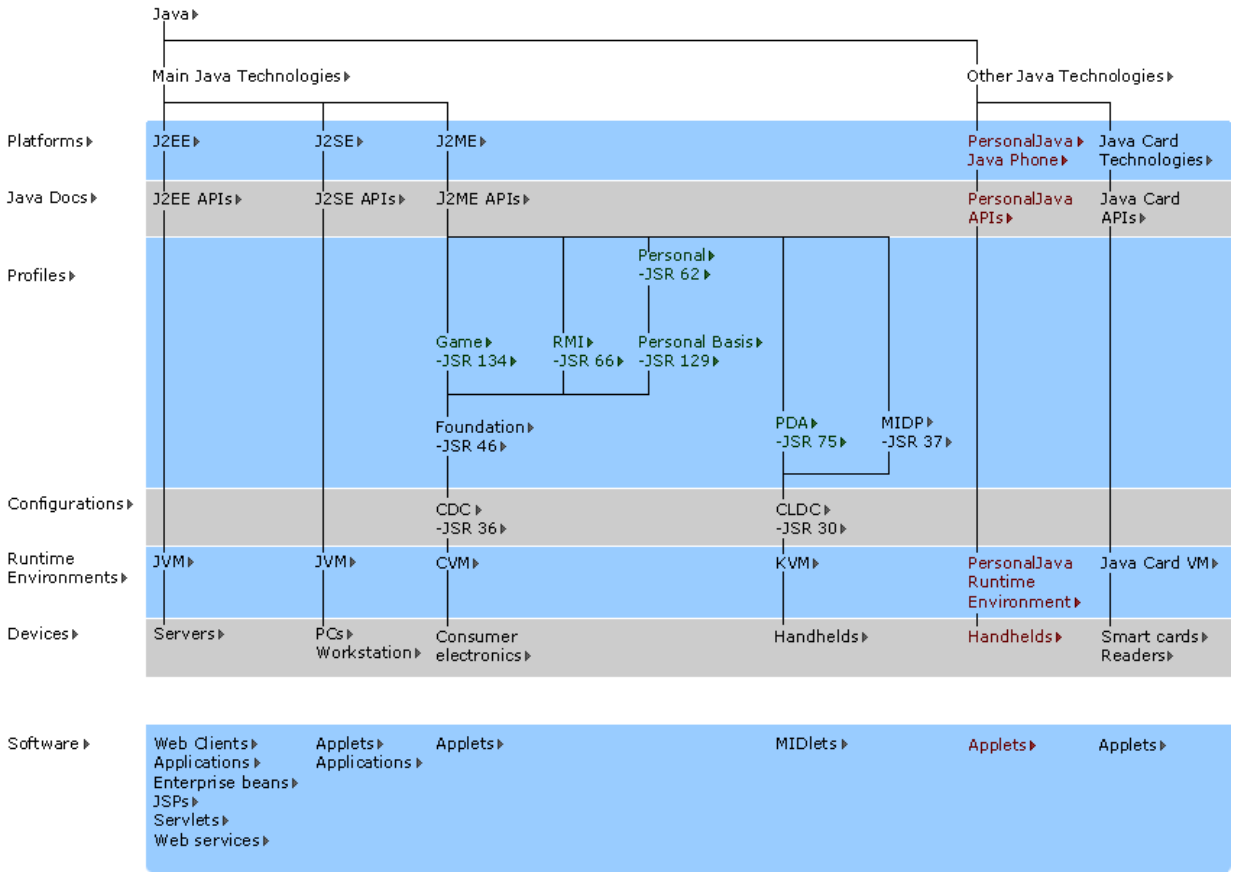


Figure 6 : Configurations and Profiles

J2ME is divided into different kind of configurations and profiles. J2ME **configuration** describes minimal Java platform required by device family. The devices which belong to this family have similar memory and processor capability. A *configuration [8]* provides the most basic set of libraries and virtual-machine features that must be present in each implementation of a J2ME environment. A configuration includes these items:

- Specific Java programming language specifications
- Specific Java Virtual Machine (JVM) specifications
- Specific Java libraries

J2ME **profile** consists of services which provided at application layer. These services can be at any subject. However at some subjects profiles are standard, all the devices which support that profile same service exists at similar form. Profiles run on a configuration. A profile can operate over another profile. A device can support more then one profile however it can have only one configuration. For instance SMS messaging is one profile. This profile is commonly used for mobile phone configuration.

2.3.2. Configurations at J2ME

Configurations may include some classes at Java Standard Edition (J2SE). Limited devices do not include all the classes due to technical restriction of these devices. A class which is defined in a specific configuration in J2SE should be specified in similar way. It can not include different method.

J2ME has two configurations:

- **CLDC (Connected Limited Device Configuration) :** for personal intermittent network connections.
- **CDC (Connected Device Configuration) :** for continuous network connections.

CDC is defined for devices which have more then 2 MB memory, however CLDC is used for less developed devices which have less then 512 KB and slow processor. CDC definition covers whole CLDC configuration. CDC has all the libraries which is defined by CLDC.

The **Mobile Information Device Profile (MIDP**) offers the core application functionality required by mobile applications, including the user interface, network connectivity, local data storage and application management. Combined with CLDC, MIDP provides a complete Java runtime environment that leverages the capabilities of handheld devices and minimizes both memory and power consumption.

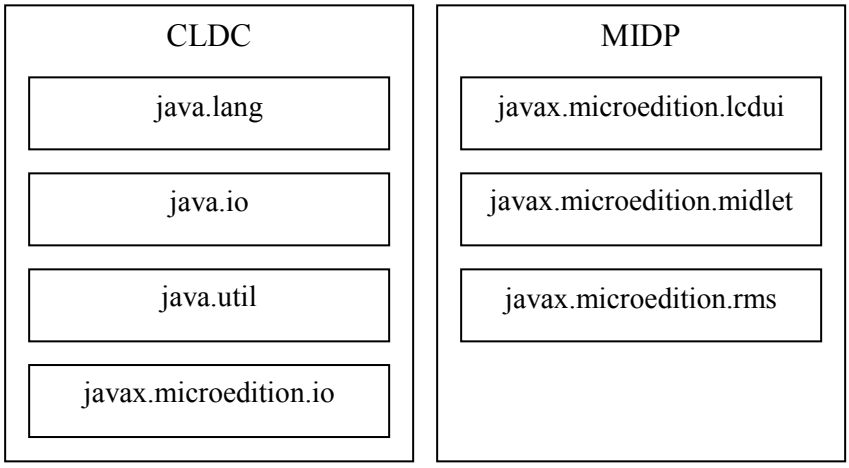


Figure 7 : MIDP packages 78[35].

The characteristics of MIDP are [35]:

- 128 KB of non-volatile memory for the MIDP implementation
- 32 KB of volatile memory for persistent data
- a screen of at least 96x54 pixels
- some capacity for input, either by keypad, keyboard, or touch screen
- two-way network connection, possibly intermittent

MIDP offers portability, which is achieved through Java. An application that uses the MIDP APIs will be portable to any MIDP device. MIDP allows the execution of multiple MIDlets. The model

defines how the MIDlet is packaged, what runtime environment is available, and how it should behave when resources are constrained. The model also defines how MIDlets can be packaged together in suites and how to share common resources. Each MIDlet suite has also a JAD file, which is a descriptor file that allows application management software (AMS) on the device to identify what it is about to install prior to installation. The model also defines a lifecycle for a MIDlet which allows starting, stopping and cleanup of a MIDlet [36].

The MIDlet life cycle:

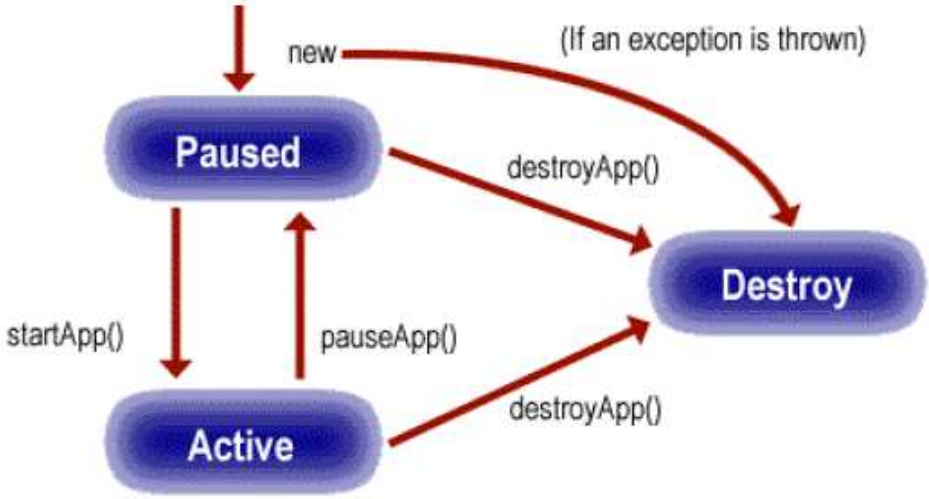


Figure 8 : The MIDlet life cycle [37]

A MIDlet is managed by the Java Application Manager, which executes the MIDlet and controls its life cycle. The MIDlet can be in one of the following states: paused, active, or destroyed. When you first create and initialize a MIDlet, it is in the paused state. If an exception occurs in the MIDlet's constructor, the MIDlet enters the destroyed state and is discarded. The MIDlet enters the active state from the paused state when its startApp() method call is completed, and the MIDlet can function normally. The MIDlet can enter the destroyed state upon completion of the destroyApp (Boolean condition) method. This method releases all held resources and performs any necessary cleanup. If the condition argument is true, the MIDlet always enters the destroyed state [37]. Figure 7 illustrates the various states of a typical MIDlet life cycle.

2.4. Introduction to Project JXTA

JXTA [5, 19] is an open source project. This industry leading peer-to-peer (p2p) platform was proposed by Sun Microsystems Inc and designed with the support of experts from academic institutions and industry. The number of experts were relatively small in the beginning but later grew.

JXTA is a set of open, generalized peer-to-peer (P2P) protocols. JXTA allow communication and collaboration as peers among any connected devices on the network from mobile to PDA, from

PC to server. The JXTA protocols are independent of any programming language and multiple implementations exist for different environments also called Bindings in JXTA terminology.

2.4.1. JXTA Architecture

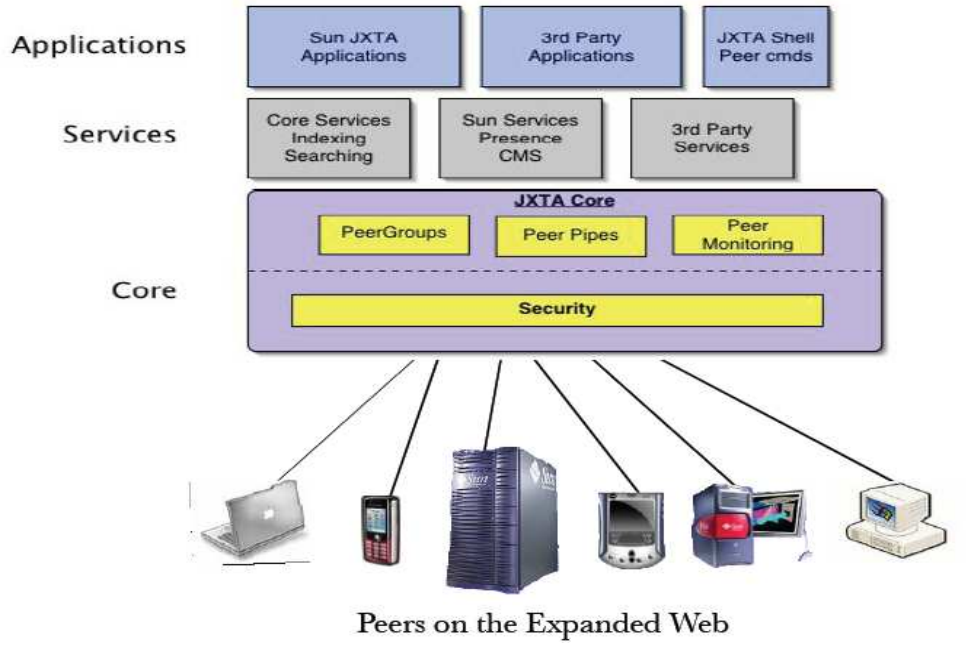


Figure 9 : JXTA Architecture [26].

- Platform Layer (JXTA Core)

The platform layer (also called JXTA Core) summarizes smallest and important fundamentals that are common to P2P networking. It incorporates building blocks that enable key mechanisms for P2P applications like discovery, transfers data with firewall handling providing http transport, the creation of peers and peer groups, and associated security primitives.

- Service Layer

This layer contains network services that are frequent in the P2P environment but may not be totally required for the operation of P2P network. For instance searching and indexing, directory, storage systems, file sharing, distributed file systems, protocol translation, bringing resource together and renting, authentication and authorization, and PKI (Public Key Infrastructure) services.

- Applications Layer

The application layer includes implementation of integrated applications, like P2P instant messaging, entertainment content management and delivery, document and resource sharing, distributed auction systems, P2P Email systems etc.

2.4.2. JXTA Virtual Network

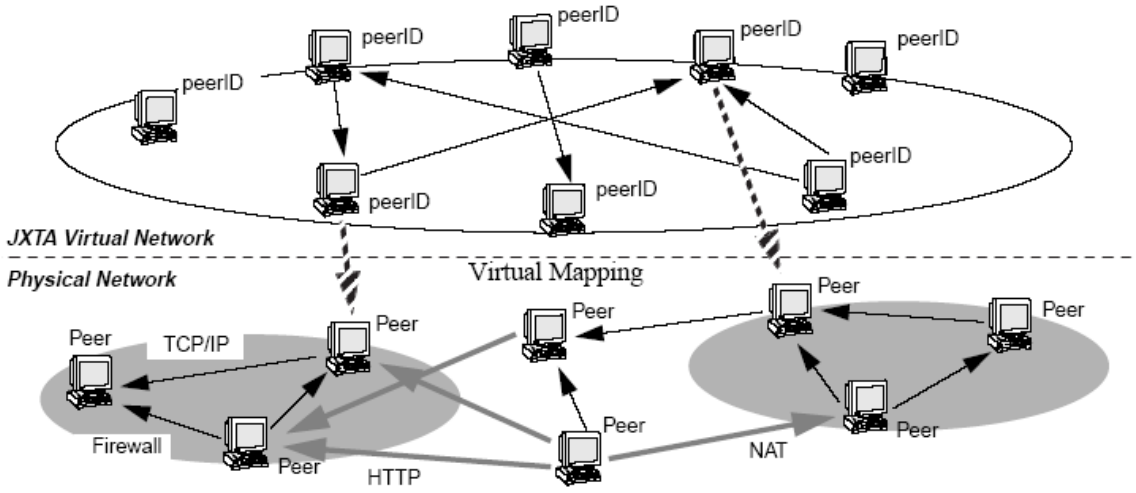


Figure 10 : JXTA Virtual Network [25].

A virtual network overlay is created by the JXTA protocols that lie on top of the existing physical network infrastructure. This virtual network allows a peer to exchange messages with any other peer independent of it's network location (firewalls, NAT's or non-IP networks). These messages are routed transparently, potentially traversing firewalls or NAT's, and using different transport protocols to reach the receiving peers (see Figure 10). The peers can communicate through Project JXTA without the require to understand or administrate complex and changing physical network topologies, thus allowing mobile peers to move transparently from one location to another. The virtual network regulates the manner in which peers discover other peer's resources, find out each other, make communication with each other and organize themselves into peer groups.

2.4.2.1 JXTA identification ID's

The identification form of Project JXTA is based on a consistent, unique and location free logical identification or addressing form. A unique JXTA ID is given to each network resource like peer, pipe, data and peer group etc. every JXTA peer is allowed to self generate its individual IDs by using 128 bit unique random UUIDs. In JXTA network each network resource like peer, pipe, data and peer group etc. is uniquely identified by its peer identification ID, this allows the peer to be existed or located independent of its physical address (see Figure 10). For example, a computer booting via DHCP and having many different IP addresses overtime, will always have the same peer ID. Like wise, a peer supporting multiple network interfaces e.g. Ethernet LAN, WiLAN, Bluetooth etc will be addressed as a single peer identification independent of the interface used. The concept of peer ID provides a peer to involve not just the physical transports, but also logical transport protocols such as HTTP, FTP or TLS. The logical identification form of JXTA brings out important advantages like separating the identification of

a resource and the location of a resource allowing a variety of virtual mappings to be used to determine the physical location of a resource.

2.4.2.2. JXTA Advertisements

JXTA Advertisements represents the announcements of entire network resources in the Project JXTA network like peers, peer group, pipes, module classes and services. JXTA Advertisements are organized and defined as XML documents, where XML documents are language independent metadata files. Project JXTA organize and uniform JXTA Advertisements of the following resources: peer, peer group, pipe, service, metering, route, content, rendezvous, peer endpoint, transport. These resources advertisements can be redefined; in addition new type of advertisements can be defined by developers. In instance, in this thesis project, a module spec advertisement will contain the associated WSDL document on its parameter. Advertisements can be used to virtually describe anything: source code, script, binary, classes, compiled JIT code, Java objects, EJB, J2EE containers.

Example 1 shows an example of a Module Specification Advertisement (jxta: MSA) that describes a unique module specification ID (MSID), a peer service name (Name), a description of the peer group (Desc), a creator name (Crtr), a version filed (Version) and a parameter field. Where the parameter can contain many sub sequent elements like pipe ID, WSDL file etc. These fields can be extended by user needs. Peers can discover and find available network resources by caching, publishing and exchanging advertisements. Resources are discovered by searching for their associated advertisements by peers. All advertisements have lifetime specified while being published that specifies the duration of advertisement in the network. An advertisement can be republished at any time to extend its lifetime before it expires. Expiration date is maintained during the exchange of advertisements among peers. Lifetimes permit to purge expired advertisements without centralized management.

```
<?xml version="1.0" encoding=”UTF-8”?>
<!DOCTYPE jxta:MSA>
<jxta:MSA xmlns:jxta="http://jxta.org">
        <MSID>urn:jxta:uuid-DEADBEEFDEAFBABAFEEDBABE000000010206</MSID>
        <Name>Service Name</Name>
        <Desc>Service description</Desc>
        <Crtr>Creator</Crtr>
        <Version>Service description</Version>
        <Parm>
                <WSDL>
                ……..
                </WSDL>
        </Parm>
</jxta:MSA>
```

Example 1 : Example of a Module Specification Advertisement.

2.4.3. Rendezvous Super-Peers

Rendezvous super peers are called to the peers that have accepted to cache advertisements index. The Project JXTA network provides a default resolver rule based on Rendezvous super peers. Locating and indexing peer advertisements (mentioned above) are done at well known places where known as rendezvous point among the JXTA peers.

For example, a peer can find sufficient peers to initialize a more advanced search strategy using rendezvous. The default rendezvous peer provides basic search mechanism to find out advertisements. However it provides hooks to allow high-level discovery services to participate in the advertisement search process. Advanced level discovery services are expected to bring out more efficient search mechanisms, because they may have a better knowledge of the content topology distribution. Advanced search mechanism will be studied and developed in implementation section.

Generally Rendezvous peers immediately maintain an index of advertisements published by their edge peers. Unlike other proxy/relay services JXTA does not cache advertisements permanently, this makes the rendezvous architecture more flexible and scalable, and reduce the problem of caching expired advertisements, as all the advertisements have TTL (Time to live). Mobile peers use The Shared-Resource Distributed Index (SRDI) service to index their advertisements on rendezvous peers.

2.4.4. Relay Super-Peers

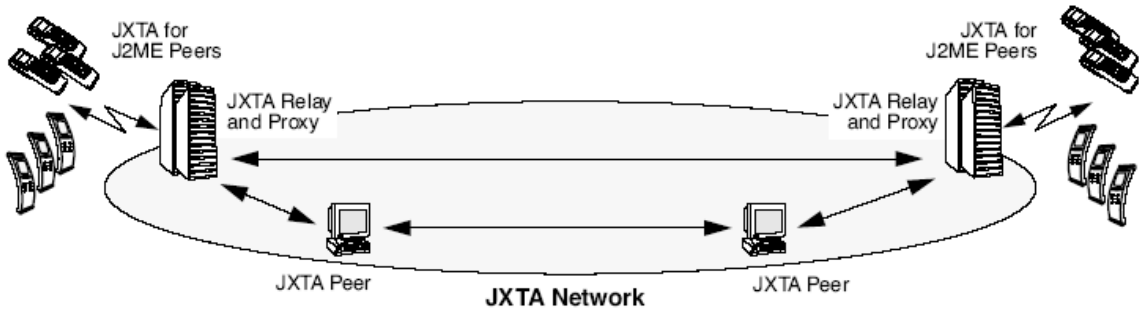


Figure 11 : JXTA Relay [24].

The JXTA relay super peers acts as a gateways for the peers which do not have direct physical connection i.e. peers that lay behind firewalls or NAT. Any peer in JXTA network can act as a relay peer, however it is better to have some capability like bandwidth, process and ram power and direct connection. Relay peers has the ability to forward or keep messages for the temporarily unavailable edge peers or unreachable peers.

JXTA virtual network decide for the routes plainly and dynamically via relay peers. Application address peers via their Peer ID's so they don't need to be aware of the JXTA network relay peer infrastructure. Edge peers or mobile peers rents a connection from one preferred JXTA relay, and

get back messages from their allocated message queues. The messages can be passed through any accessible JXTA relays, not just only from connected one.

As the time passes, edge peers travel around from relay to relay for improving their visibility or enhancing the connectivity quality among peers in the network. One important thing to note, that relay/edge peer connection is temporary in nature. Edge peer may drop connection at any time and reconnect to a different relay at JXTA network. Relays provide service for the edge peer for a settled rent time period. Relay peers traverse among each other to keep a list of available relays similar to the rendezvous peers. The detection of relay peers are bootstrapped by seeding peers where they only needed when there are no relays around. Edge peers keeps available relay list advertisements in the case of reconnect or reboot. Edge peers obviously reconnect to another relay from the list in the case of relay absence.

2.4.5. JXTA Protocols

The JXTA consist of six protocols that have been mainly designed for ad hoc, pervasive, and multi-hop P2P network computing. By using JXTA protocols, peers can work together to form self prepared and self configured peer groups without the need of a centralized running infrastructure, moreover independent of their positions in the network (edges, firewalls, network address translators, public vs. private address spaces).

The JXTA protocols are intended to have very little overhead, have some clear idea about the underlying network transport and inflict a small amount of necessities on the peer environment, moreover protocols are able to be used to arrange a large variety of peer-to-peer applications and services in a greatly unreliable and changing network environment.

JXTA protocols are used by peers to promote their resources and to find out network resources like peers, groups, pipes etc available from other peers. Particular relationships are created when peers form and join peer groups. Peers work together to route messages allowing for full peer connectivity. The JXTA protocols let peers to communicate with each other without the need to recognize or supervise the potentially complex and active network topologies which are increasingly common in peer to peer network.

Peers are allowed by JXTA protocols to dynamically route messages across multiple network hops to any destination in the network potentially traversing firewalls. Every message contains a list of bridge peers information from side to side which the message may be traversed. The traveling around may be helped out by the intermediate peers in the route. This is done by using routes they know of to shorten or optimize the route a message is set to follow.

The JXTA protocols are composed of six protocols that work together to allow the discovery, organization, monitoring and communication between peers. Short description of each is given below:

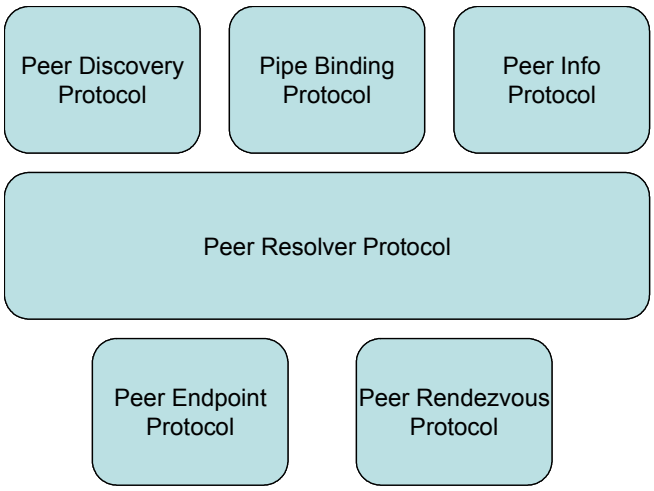


Figure 12 : The JXTA protocols architecture.

- Peer Resolver Protocol (PRP) is method where a peer can send an inquiry to one or more peers, and receive a response (or multiple responses) to the inquiry. The PRP implements a query/response protocol. The response message is matched to the query via a unique id included in the message body. Queries can be directed to the whole group or to specific peers within the group.
- Peer Discovery Protocol (PDP) is the mechanism by which a peer can advertise its own resources, and discover the resources from other peers (peer groups, services, pipes and additional peers). Every peer resource is described and published using an advertisement. Advertisements are programming language- neutral metadata structures that describe network resources. Advertisements are represented as XML documents.
- Peer Information Protocol (PIP) is the mechanism by a which a peer may obtain status information about other peers. This can include state, uptime, traffic load, capabilities, and other information.
- Pipe Binding Protocol (PBP) is the instrument by which a peer can establish a virtual communication bridge using pipe between one or more peers. The PBP is used by peers to bind two or more input/output gates (like socket connections). Pipes provide the base communication mechanism between JXTA peers.
- Endpoint Routing Protocol (ERP) is the mechanism by which a peer can discover a route (sequence of hops) used to send a message to another peer. If a peer “A” wants to send a message to peer “C”, and there is no known direct route between “A” and “C”, then peer “A” needs to find intermediary peer(s) who will route the message to “C”. ERP is used to determine the route information. If the network topology changes and makes a previously used route unavailable, peers can use ERP to find an alternate route.
- Rendezvous Protocol (RVP) is the mechanism by which peers can subscribe or be a subscriber to a propagation service. Within a peer group, peers can be either rendezvous peers or peers that are listening to rendezvous peers. The Rendezvous Protocol allows a peer to send messages to all the listening instances of the service. The RVP is used by the Peer Resolver Protocol and by the Pipe Binding Protocol in order to propagate messages.

2.5. Introduction to Project Lucene

Lucene [30,31,32], is a Java based high-performance, full-featured text search engine library. Nearly any application specially cross-platform applications that requires full-text search can adopt this technology. It is an open source project hosted by Apache. The Java Lucene product is used by many well known websites like Wikipedia [33], online encyclopedia, as well as in many Java applications. It is rapid, reliable tool that has proved its value in countless number of demanding production environments.

Among many developers, Lucene is well-known for its full-text indexing, but they are less aware that it can also provide strong complementary searching, filtering, and sorting functionalities. In fact, many search mechanism provide combining full-text searches with filters on different fields or criteria. For example, a search on a database of employees or workers with the criteria to limit the results to certain types of employees like managers. Usually, this type of condition-based searching is in the monarchy of the relational database. However, Lucene provides many powerful features that allow efficient combination of full-text searches with condition-based searches and sorts.

2.5.1. What Lucene can do?

Lucene allows to add indexing and searching capabilities to user applications (these functions are described in section 2.5.2.). Any type of data that represents textual information can index and make searchable by Lucene. As can be seen from Figure 13, the source, the structure or even language of database is not important for Lucene, as long as the data can be changed to textual information. This means the users can use Lucene to search and index information where kept in JXTA advertisement, files, web pages on remote web servers, documents stored in local file systems, simple text files, Microsoft Word documents, HTML or PDF files, or any other format from which textual information can be extracted.

Similarly, with Lucene's help data stored in databases can be indexed, giving users full-text search capabilities that many databases don't provide. Once Lucene is integrated, applications users can make searches such as +George +Rice -eat -pudding, Apple □pie +Tiger, animal:monkey AND food:banana, and so on.

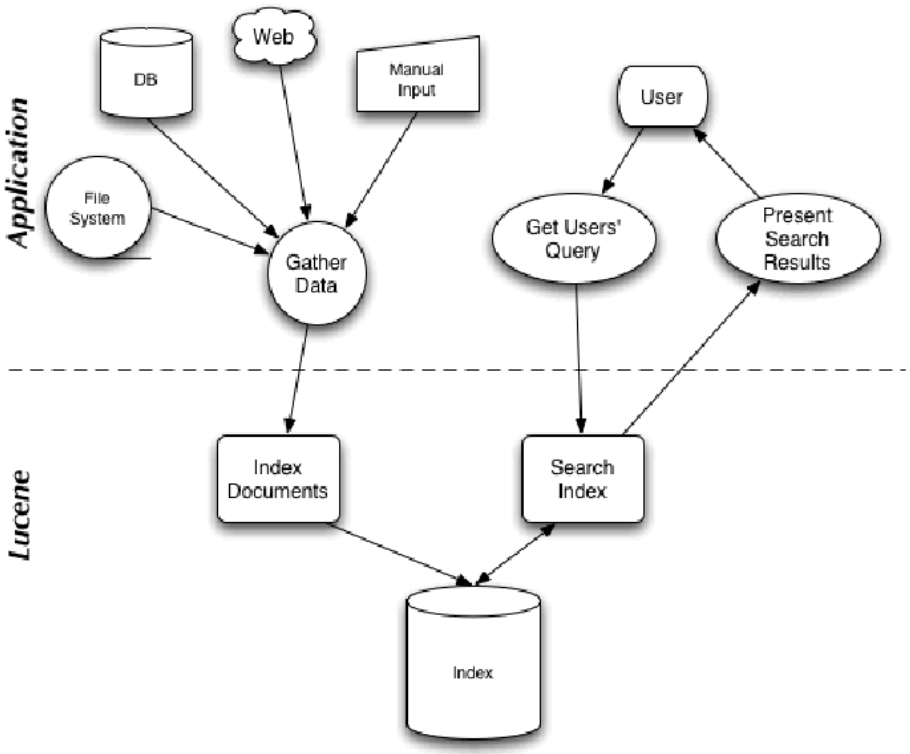


Figure 13 : Lucene architecture [31]

2.5.1.1. Lucene Background

Lucene was initially written by Doug Cutting and is available SourceForge project web site container. Then Lucene project involved in the Apache Software Foundation's Jakarta family of open source server-side Java products in September of 2001. From that time with every new release, the project has gained more visibility, attracting more users and developers. As of May 2006, Lucene version 2.0.0 has been released [32,33].

2.5.2. Indexing and searching

In the spirit of all search engines indexing is the concept where indexing is the process of the original data into an extremely efficient cross-reference lookup in order to make possible fast searching. Searching is done using the generated cross-reference data indexes.

2.5.2.1. What is indexing, and why is it important?

Suppose, there is a need to search a large number of files in order to be able to find files that contained a certain word or a phrase. How a program could be used to do this? A naive approach would be to sequentially scan each file for the given word or phrase. This approach has a number of drawbacks, the most obvious of which is that it doesn't scale to larger file sets or cases where files are very large. This is where indexing comes in: To find out large amounts of text quickly, initially that text must be indexed and then convert it into a format that will let search it rapidly,

eliminating the slow sequential scanning process. This conversion process is called indexing, and its output is called an index.

An index can be thought of, as a data structure that allows fast random access to words stored inside it. The idea at the back of it is equivalent to an index at the end of a book, which allows to rapidly locate pages that talk about certain topics. In the case of Lucene, an index is a specially designed data structure, typically stored on the file system as a set of index files.

2.5.2.2. What is searching?

Searching is the procedure of looking up words in an index to discover documents where they appear. The excellences of a search are usually described using accuracy and remind metrics. Remind measures how healthy the search system finds relevant informations, on the other hand accuracy measures how well the system filters out the unrelated documents. However, when we are thinking about searching a number of other factors must be measured. The importance of speed and the ability to quickly search huge numbers of text are already mentioned earlier. Support for single and multiterm queries, phrase queries, wildcards, result ranking, and sorting are also important. Lucene's powerful software library offers a number of search features, qualities and behaviors.

2.5.2.3. Creating an index

To do a full text searching at Lucene it is required to build an index. To construct an index just specify a directory or a RAM memory and an analyser class. The analyser breaks text fields up into indexable tokens: this is a core part of Lucene, and Several types of analysers are provided by Lucene moreover you can build up your own specific analyser. Here are some of the standard ones [31,34]:

StandardAnalyzer : A sophisticated general-purpose analyser.
WhitespaceAnalyzer : A very simple analyser which just seperates tokens using white space.
StopAnalyzer : Removes common English words which are not usually useful for indexing .
SnowballAnalyzer : An interesting experimental analyser which works on word roots (a search on "rain" should also return entries with "raining", "rained", etc...) .

There are even some language-specific analysers :
GermanAnalyzer (just don't mention the war, as Basil Fawlty would say...)
RussianAnalyzer

It isn't hard to implement one’s own Analyser, though the standard ones often do the job well enough. In this thesis project StandardAnalyser is used.

2.5.2.4. Indexing an object

Lucene Document class is used to index an object, where the user add the fields whatever user wants to be indexed. Lucene supports four field types [31,34]:

UnIndexed : This type of field is stored in the index but is not used in searches. Unindexed fields are useful for storing object IDs, for example.
Keyword : A keyword is stored and indexed as-is. searches by keyword can be done, Keywords may be Strings or Dates, and can be used to build complex keyword-based search functions.
Text : The field value is analysed and indexed for full-text searches. The original value is also stored in the index.
UnStored : The field value is analysed and indexed for full-text searches, but the original value is not stored in the index. This is useful for indexing large blocks of text.

In the case of this thesis project, only simple full-text searching is used, only two fields are added. Module spec Id, so the object can be retrieved later on from the query result list. And Content information where it contains name, description and WSDL file. The details are explained at implementation section.

3. Mobile WS-Discovery Design

3.1. Mobile Web Service Provider

The Mobile Host, Mobile Web Service Provider, was designed and tested on a SonyEricsson P800 Smart Phone and was developed in Personal JAVA. The footprint of the fully functional prototype is only 130 KB. The Mobile Host has been developed as a Web Service handler built on top of a normal web server. The Web Service requests sent by HTTP tunneling are diverted and handled by the Web Service handler. The evaluation of Mobile Host showed that service delivery as well as service administration can be done with reasonable ergonomic quality by normal mobile phone users. As the most important result, it turns out that the total WS processing time at the Mobile Host is only a small fraction of the total request-response time (<10%) and rest all being transmission delay. The following Figure 14 shows the basic architectural setup of the Mobile Host. A detailed discussion of implementation and evaluation details of this Mobile Host [1, 27] is beyond the scope of this proposal.

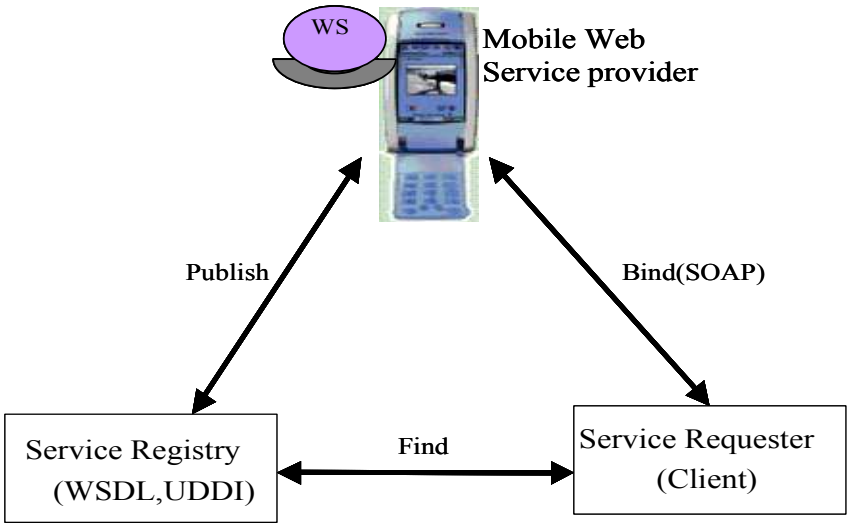


Figure 14 : Basic architectural setup of Mobile Host [1].

The Mobile Host, once commercially viable, can serve some useful services. As a Mobile Host, the mobile terminal becomes a multi-user device where the owner/carrier of the device can work in parallel with users of the Web Service without explicit effort on his/her side. From a commercial viewpoint, there is a reversal of payment structures. While traditionally the information-providing Web Service client has to pay to upload his or her work results to a stationary server (where then other clients have to pay again to access the information), in the Mobile Host scheme responsibility for payment shifts to the actual clients -- the users of the information/services provided by the Mobile Host. Another commercial aspect is the possibility for small mobile operators to set up their own mobile Web Service business without resorting to stationary office structures, thus going one step further in the move from central to P2P architectures [1].

3.2. Problem Domain

Using Web Services standards for web service discovery brings out some drawbacks. By introducing mobile web service provider and consumers into the Web Services market, the quantity of Web Services increase tremendously. There are millions of mobile users in the market. Hence increasing number of Web Services will lead to difficulties on; discovering of exact services, up to date services, and quick response. Moreover Centralized registries are performance bottlenecks and may result in single points of failure.

3.2.1. Problem Description

Initially we can raise some questions like, why we use p2p, what is advantages of it, how we use it. We will have answers to those questions at next section however let us check out main problems which we face by introducing Web Services to mobile networks.

We can compose the problems as below;

- One of the problem is that, when we consider mobile node as a central registry, the node which carrying that registry may not available all the time. User who needs a service which is kept on that registry can not be accessible for some time. We have to think alternative solutions like distributed registries. JXTA/ME can handle this problem easily, we can advertise WSDL files at JXTA peer group network as a JXTA standard service. These services will be available at network even the publisher disappears. These services can be available at network till last node leaves the group.

- Movement of mobile service provider (mobile host) from one operator network to another could lead to communication loss. Moreover the identity of a Mobile Host, which is IP, could change by reconnecting to a new operator network. Clients who use mobile host's service could not be accessible due to new IP. One solution could be republishing WSDL file with new IP address, however this needs maintenance. Moreover making available this WSDL file with new IP takes time. This means the Web Service will not be accessible for a while. As a solution we propose to use JXTA/ME, every peer at JXTA/ME network is identified by an ID, called peer ID. This ID is unique and static, when a node associate with an ID, this ID will stay forever with that device no matter where or which communication protocol is used. By using peer ID mobile host does not have to worry about changing operator network. Mobile Host address is always known to client, as the client can find service provider from JXTA/ME network.

3.2.2. Proposed Solution

The previous section also gives a brief explanation to the question why P2P was used in the project. P2p provides access to information and collaboration without the need for a third party web server like in our problem domain UDDI.

Generally we have explained advantages of involving JXTA to Web Services world. Now we are going to show how to use JXTA with Web Services. First of all we are going to compare these

two technologies then try to combine them. We are proposing three alternative solutions to combine these technologies. However in the implementation section we will show how we made a hybrid solution to the problems which we described.

3.2. Comparing Web Services and JXTA

Web Services and JXTA have a lot of common points. A close examination of this two technology shows; their aims, architectures, problem domains and their method for solving them. We are going to analyze these two technologies, and show how these two technologies produce a strong solution to our main problem [16].

Let briefly examine Web Services and JXTA with common points and differentiations:

- Web Services and JXTA lay on SOA(Service Oriented Architecture). Both define their own communication protocols and computing topology. They are designed to enable loosely coupled systems often employing layered stacks and referencing best-of-breed protocols. Both describe their information using XML, to make platform and language independence. Both have a heavy weight on distributed computing. Their differences are more exciting, despite their similarities.

- Web Services, are based on a centralized model and primarily focused on standardizing messaging formats and communication protocols. JXTA, on the other hand, is based on a decentralized model and primarily focused on supplying processing power, content, or applications to peers in a distributed manner, and less focused on the semantics of messaging formats and communication protocols.

Now we are going to present the differences by comparing the architectures. In fact both architectures are quite similar, however they can give us a nice point of view about what are the differences among them.

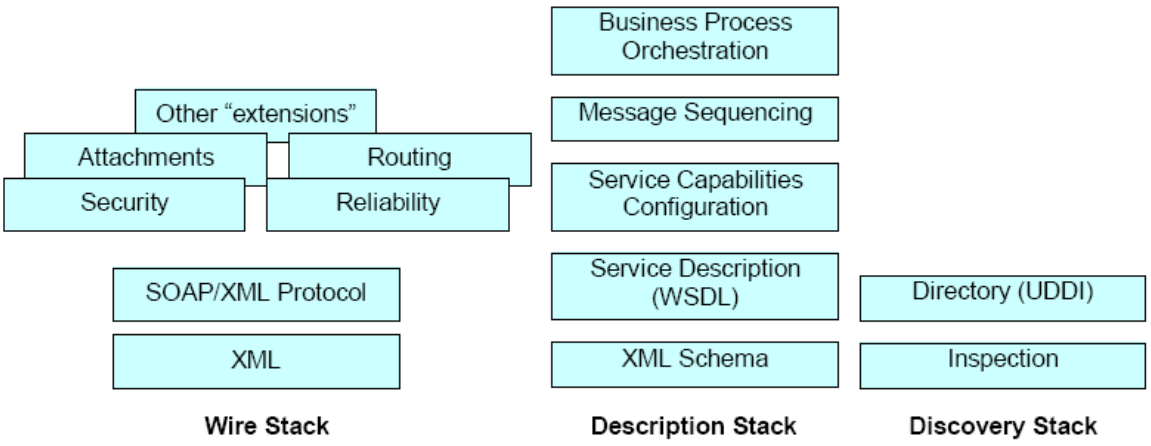


Figure 15 : Web Services Conceptual Architecture (by IBM) [16].

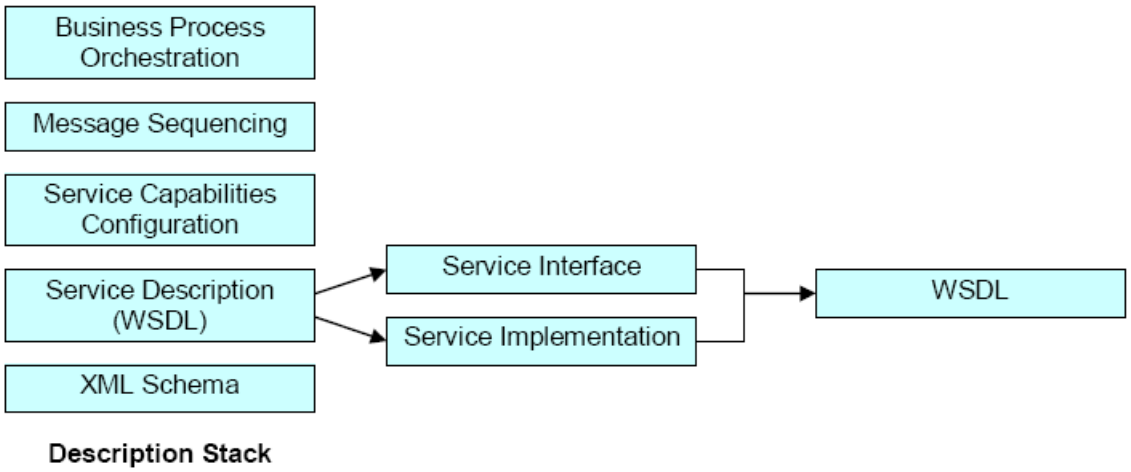


Figure 16 : JXTA Architecture (by Sun) [16].

The main differences of JXTA and Web Services are consists as, communication channels, wire protocols, security mechanisms, routing and envelope structures for moving information around. Moreover, JXTA defines peer domain concepts like, “What is an Advertisement” and “What is a Rendezvous Peer”.

Wire

Mainly servers that run Web Services will be well-known hosts, with static IP addresses and will be on the outside of a firewall. However JXTA peer could be behind a firewall or NAT and proxies. JXTA deals with firewalls and NATs using different methods with respect to Web Services, it introduces Relay peer to handle NATs and proxies. JXTA does not depend on IP, it can communicate through any communication channel by defining peer IDs. Further information available on JXTA section.

Security

Encryption, decryption, authorization, authentication, hashing, etc. are available on both systems, with this perspective they look similar. The enhancements of security library and standards make similar type of features on such systems. The JXTA group is taking on the "web of trust" issues, where essentially one peer can loan out its credentials to another peer. This advancement could be significant and might eventually find its way into Web services.

Discovery

Discovering a service on Wes Services is done by a centralized registry system which known as UDDI. This repository system keeps a listing of all the published services, kind of like the global yellow pages. In addition, services can be categorized and may store to different UDDI. On the other hand in JXTA environment store services on edge peers, and also may be on rendezvous peers, to act as a meeting place for peers with similar interests , this means there is no centralized registry, all information stored as a decentralized manner. Peers make discovery in a variety of ways such as multicasting, querying services etc. As our main goal in this research is discovery mechanism, Web Services can borrow some of the decentralized techniques from JXTA peer

world, while the peers will being to leverage some of the centralized registries in the Web Services world.

Reliability

With reliability perspective both system depend on the environment, there is no standard that specifies reliability issue, both systems general topology shows how it will be handled. Web Services use centralized mechanisms to create available systems. On the other side JXTA system use distributed mechanism, where peers advertise services to be discovered by other peers.

Service Interfaces and Message Protocols

Web services have gone through great trouble to define a standard way of describing a service through the Web Services Definition Language (WSDL). It provides a method of describing an interface via XML descriptors. In addition, Web services utilize SOAP as a standard way for the consumer to send a message to the provider facilitating a remote method invocation in an object oriented fashion. The initial JXTA implementation used formats similar to the Web services approach, but slightly different. The JXTA team is now going back and making adjustments to the core platform to make peers interoperate with Web services using protocols like SOAP and WSDL [16].

3.3. Combining Web Services and JXTA

We have compared Web Services and JXTA in detail with architectural and conceptual view, and with advantages and disadvantages. Now we are going to combine these systems to work together.

Theoretically, we can store JXTA services in centralized UDDI registries. However, since our main purpose in this study is to distribute services in more effective and reliable way, we are going to distribute services in a decentralized manner to provide good fault resistance and network resilience.

Advertisements

In JXTA system decentralization is achieved by using advertisements. Advertisements are language-neutral metadata structures resource descriptors represented as XML documents. A peer which wants to advertise itself or its services, announces an advertisement on the JXTA network. The JXTA protocols use advertisements to describe and publish the existence of a peer resource. Peers discover resources by searching for their corresponding advertisements, and may cache any discovered advertisements locally. Every advertisement exists with a lifetime that specifies the availability of that resource. Lifetimes gives us the opportunity to control out of date resources without need of any centralized control mechanism. To extend the life time of an advertisement, we can republish it.

JXTA Modules

We need to distribute Web Services as JXTA advertisement, so that it can be sensed as a JXTA services among peers. JXTA modules are an abstraction used to represent any piece of "code" used to implement a behavior in the JXTA world. The module abstraction does not specify what this "code" is: it can be a Java class, a Java jar, a dynamic library DLL, a set of XML messages, or a script. Modules provide a generic abstraction to allow peers to describe and instantiate any type of implementation of a behavior. The JXTA platform uses module advertisements to describe where to find the services and implementation, in addition it describes itself. The module abstraction includes a module class, module specification, and module implementation:

- The ***module class*** is primarily used to advertise the existence of a behavior. The class definition represents an expected behavior and an expected binding to support the module. The module class is identified by ModuleClassID.

- The ***module specification*** is primarily used to access a module. It contains all the information necessary to access or invoke the module. There can be more then one module specifications for a given module class. Each module specification is identified by a unique ID, the ModuleSpecID. In addition module specification implies network compatibility.

- The ***module implementation*** is the implementation of a given specification. There might be more then one implementation for a given specification. Module implementation is defined by a unique ID, ModuleSpecID.

WSDL can be represented with these there modules, and advertisement of these modules represents UDDI behavior.

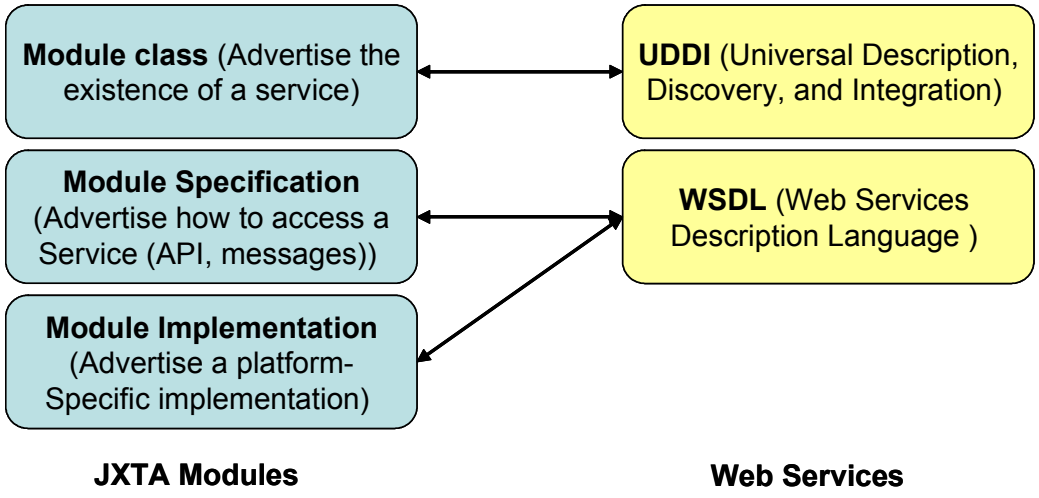


Figure 17 : Comparison of JXTA Modules and Web Services. The JXTA module together represent a combination of UDDI in the sense of publishing and finding service description and WSDL in the sense of defining transport binding to the service.

3.3.1. JXTA-SOAP Model [45]

JXTA SOAP project, which was started by Kevin Burton, is now managed by Michele Amoretti of Distributed Systems Group (University of Parma, Italy). JXTA-SOAP is a package which allows SOAP communication over the JXTA Peer-to-Peer network. The main purpose of this project is to send Web Service messages through JXTA pipes. Instead of making IP invocation to access Web Service, JXTA peer ID is used to push Web Service message into JXTA network. Initially we aimed to adapt this project to mobile peers however due to its complexity we get the ideology and developed third scenario to invoke Web Services through JXTA pipe network.

JXTA-SOAP is a project which allows SOAP communication over the JXTA Peer-to-Peer network. Where JXTA network is acting like a bridge for SOAP messages. A peer is connected to the JXTA network through a pipe service. The pipes are capable of receiving any type of data payload. Web Services use SOAP over HTTP messages, this means the XML payload of SOAP requests and responses travel over HTTP. The JXTA pipe has the ability of receiving and sending any type of data payload. Hence we can use pipes to carry out SOAP XML payload [17].

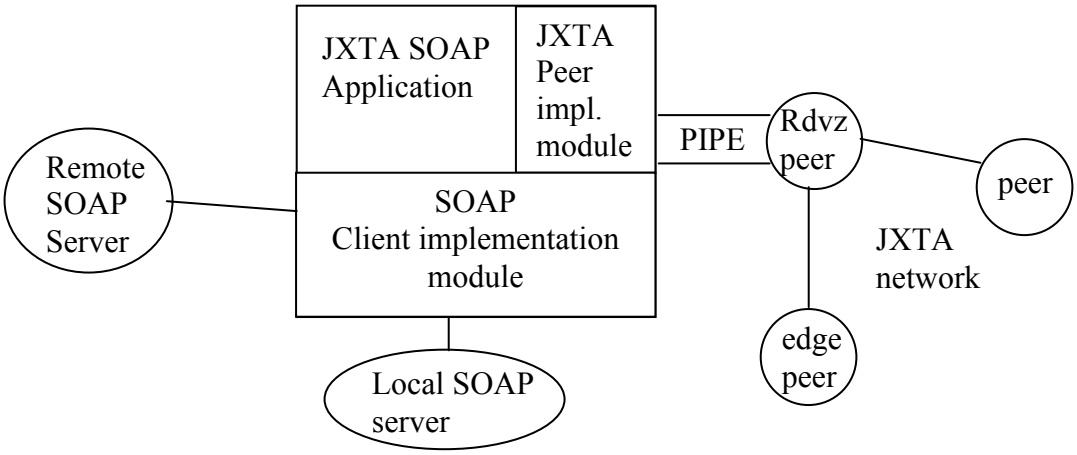


Figure 18 : The JXTA SOAP architecture [21].

This model as illustrated in Figure 18 has two modules; JXTA peer implementation module and SOAP client implementation module. The JXTA peer implementation module provides all JXTA functionality. The peer does not need a static IP address to expose its Web Services outside the network, this job handled by JXTA peer implementation module as JXTA provides unique peer ID.

The service provider that wants to advertise a Web Service over the JXTA network will request the JXTA-SOAP application to advertise it. The application receives information (like name and description of the Web Service) from the service provider. Then the application generates an advertisement with the provided data, and publishes it to all known JXTA rendezvous points. JXTA-SOAP can create an input pipe for listening service request from other peers.

When a peer wants to use a service which is unavailable in the mobile unit, but available at the Web Service provider's server, that server acts like a remoter SOAP server. This server can be any SOAP server over the internet. Remote SOAP server can be invoked by JXTA-SOAP

application through regular client server interaction. JXTA peers have no idea whether the exposed Web Service is located in the remote or local server.

Both the provider and the client peers must be instances of the JXTA-SAOP application in order to make the Web Service invocation over the JXTA network possible. In addition to this, the client must know the name of the Web Service in order to invoke the service. The name of the service can be retrieved from JXTA advertisements [21].

Further information about this model can be found on the name *SOAP-over-P2P Model* from Java P2P Unleashed homepage [18].

3.3.2. Proxy Model

In this model the main purpose is to make JXTA search for the Web Services where a Web Service is defined like a normal JXTA service as we described above in JXTA Modules section. Peers with common interests generate peer groups. A peer can be a member of any peer group, as much as it wants. In the Proxy Model, the user is unaware of if the particular service is a Web Service or a JXTA group service. As a result this means that the Web Service could be advertised, discovered, located, and invoked similar to any other JXTA Service. A JXTA Group Service belong to whole peer group members, it does not belong to a single peer [17,21].

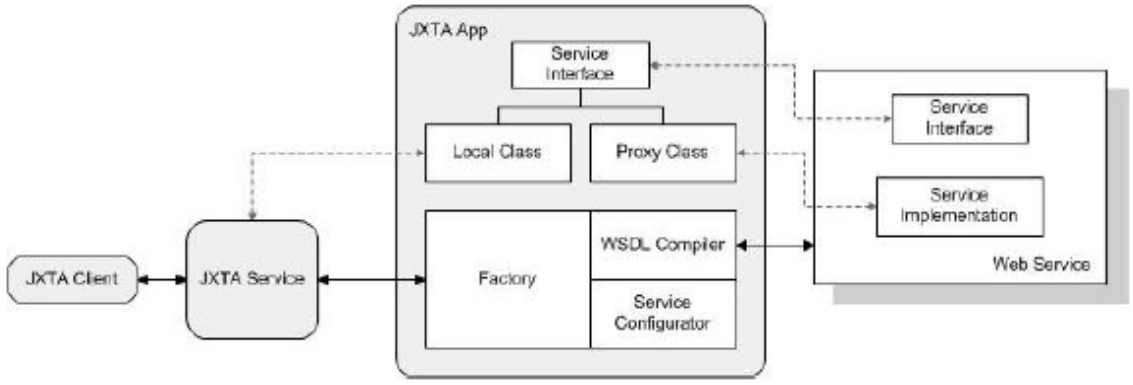


Figure 19 : Proxy Model architecture [21]

Creating the service implementations for the network service involves the creation of the following components that are implemented as part of the service:

- An Interface to the Web service.
- A Local and Remote implementation of the interface.
- A Service Configuration file containing the properties of the service.
- A Factory that handle the client request in accessing the Web service depending upon the service configuration.
- A Runtime WSDL Compiler used to generate the dynamic proxies that delegate the client request to access the Web service.

The Proxy Model, as shown in figure 19, initially service interface dynamically generates classes (local and proxy) from WSDL definition file. So that it will define the Web Service methods. The proxy class provides Web Services to invoke remotely from the JXTA network. The generated interface and the associated files are used by the proxy class implementation to access the Web service. WSDL compiler translates the WSDL definition of Web Service's interface into java source files. The generated classes becomes the JXTA service, which means this services could be advertised as normal JXTA services inside the network. By this way JXTA client would not require prior knowledge about Web Service and its interface. The proxy class uses SOAP to communicate with the Web Service, however the client does not need to understand the SOAP.

The *factory* would make a decision to load the Web Service interface, whether to use a local class (if any exists) or proxy class, while a JXTA peer makes a request for a service. The chosen class should be defined at the service description. This service description specifies how a remote Web Service should be implemented as a JXTA service.

3.3.3. Port Forwarding Model

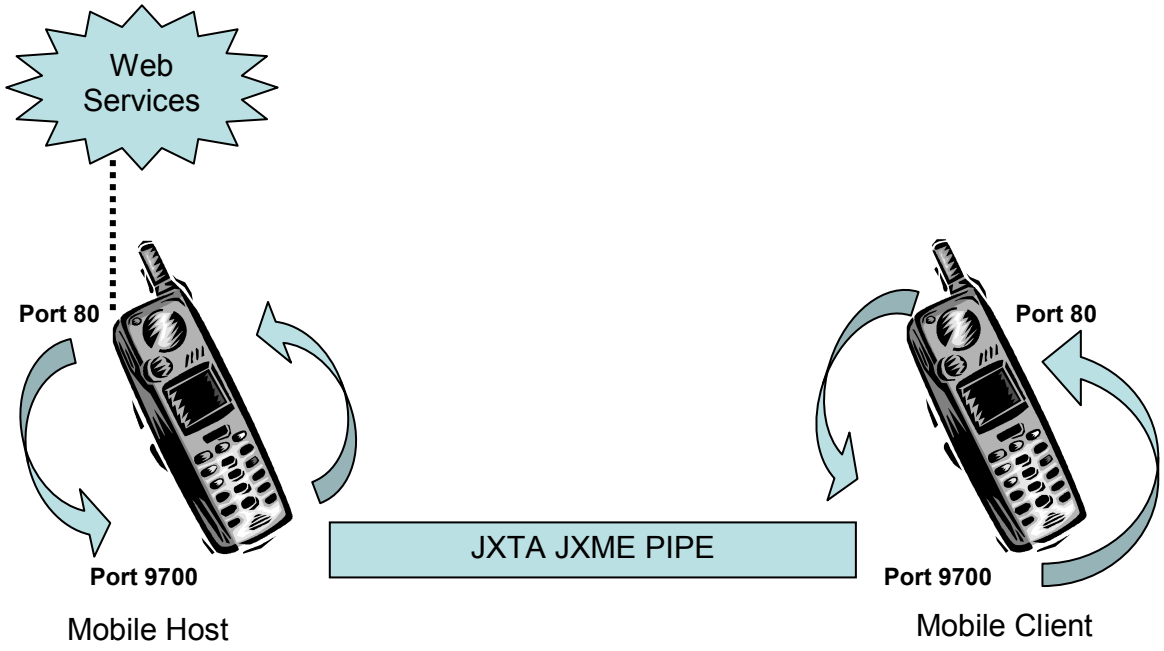


Figure 20 : Port Forwarding Architecture

This architecture aims to find a solution to combine Web Service with JXTA network. Moreover this architecture intends to eliminate IP during invocation of Web Services. This could be done via using JXTA ID instead of IP. Main purpose of this architecture is to carry Web Service messages through JXTA pipes.

When we look at mobile host mechanism we can see that it can only accept SOAP messages from port 80. In fact we can modify JXTA access port so that messages can go directly to port 80 however this will not solve our problem as mobile host can only accept SOAP over HTTP format

messages. This means mobile Web Service provider can only parse SOAP over HTTP message format. As a matter of fact we can modify mobile host so that it can also accept JXTA message format. However this would spoil the main purpose of mobile host, where mobile host not only serves Web Service clients from JXTA network but also Web Service providers and clients from mobile or non mobile networks like Internet.

As a solution to this problem we are trying to define an architecture. Messages among JXTA peers will travel over normal JXTA ports and through JXTA pipes. JXTA pipe messages will convert to SOAP over HTTP format. In addition a message which arrives to JXTA port will be forwarded to http port 80 and the response will be sent vice versa. These operations are performed at mobile phones using applications which can read JXTA messages.

3.4. Searching Web Services and client application in JXTA

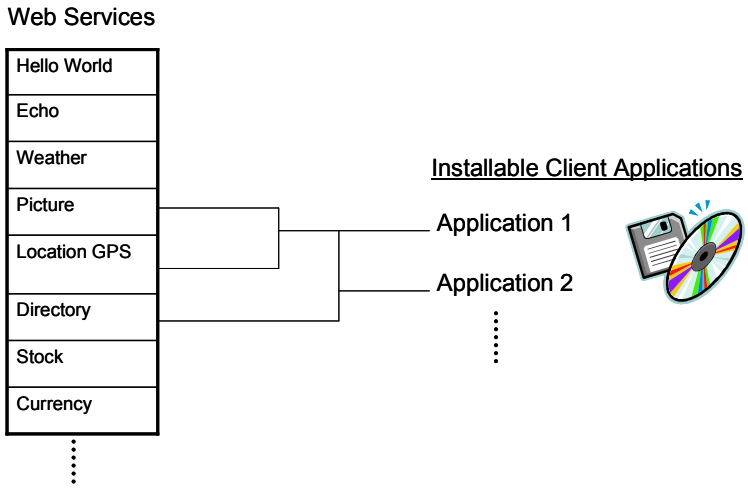


Figure 21 : Web Services and Client Applications structure.

Generally the users in the mobile Operator network attracted in client applications. A client application can be generated from WSDL. This process is explained in implementation section in more detail. A client application might use one or more Web Services at the backend and can be provided as an installable application. The advertisements of these applications are represented as a combination of WSDL and MCS (Module Class Service) which are advertised at specific peer group. Advertising WSDL files in JXTA network can be done by applying Proxy-Model solution as we described above.

Searching Web Services in JXTA is an important issue. Searching is needed at every aspect of this project. Now we are going to explain each step separately.

Client application, which initially establish connection to JXTA network. This tiny application provides user to search Web Services which are represented as Module Class Advertisement (MCA) and Module Specification Advertisement (MSA) at JXTA network, can be searched by name and description parameters. As we are talking about huge numbers of Web Services these parameters might not sufficient to find out specific search. Moreover we would like to extend these search criteria at WSDL level. Which means search parameters would not restricted by MCA and MSA, it will extend by looking up to WSDL tags. This detailed search mechanism

might not be done at edge peer (i.e. smart phone), so we shifted this search mechanism to a standalone middleware, this middleware will get request from resource restricted device and response found search results. Detailed search algorithm works as below.

Each web service has an associated service description (WSDL) that describes its abstract interface and the concrete implementation functionality. The service description will be parsed for all major content elements like the type definitions, elements, operations etc. These elements will be modeled on a tree. The middleware starts parsing from the implementation level (service, port, binding) of a WSDL and goes up to the interface level (portType, message, operation). These results are used for analysis, and find relevant WSDL files. These analysis processes will be done as above algorithm. Now we are going to explain the core algorithm, however on implementation section we are going to examine different solution for this algorithm, for instance Lucene search algorithm.

The **tf–idf** weight (term frequency–inverse document frequency) is a weight often used in information retrieval and text mining. Each word is assigned a weight, it reflects the importance of a word within the document. This value is calculated based on its frequency and its distribution across a collection of documents. The idea behind IDF (Inverse Document Frequency)[29] weighting is that people usually express their opinion by using frequently used words. The similarity of two documents is calculated based on TF-IDF [23] and the cosine similarity between the angles of two vectors which represent the documents. This value is then normalized 0 through 1, and is used to rank the search results.

After the pre-processing step, a description is split up in n-grams, instead of words like in other information retrieval systems. The experiment at UDDI Explorer [22] shows that, the tri(3)-gram, and quad(4)-gram based systems have shown to return better results than the word-based tokens system [22].

3.5. Putting All Together

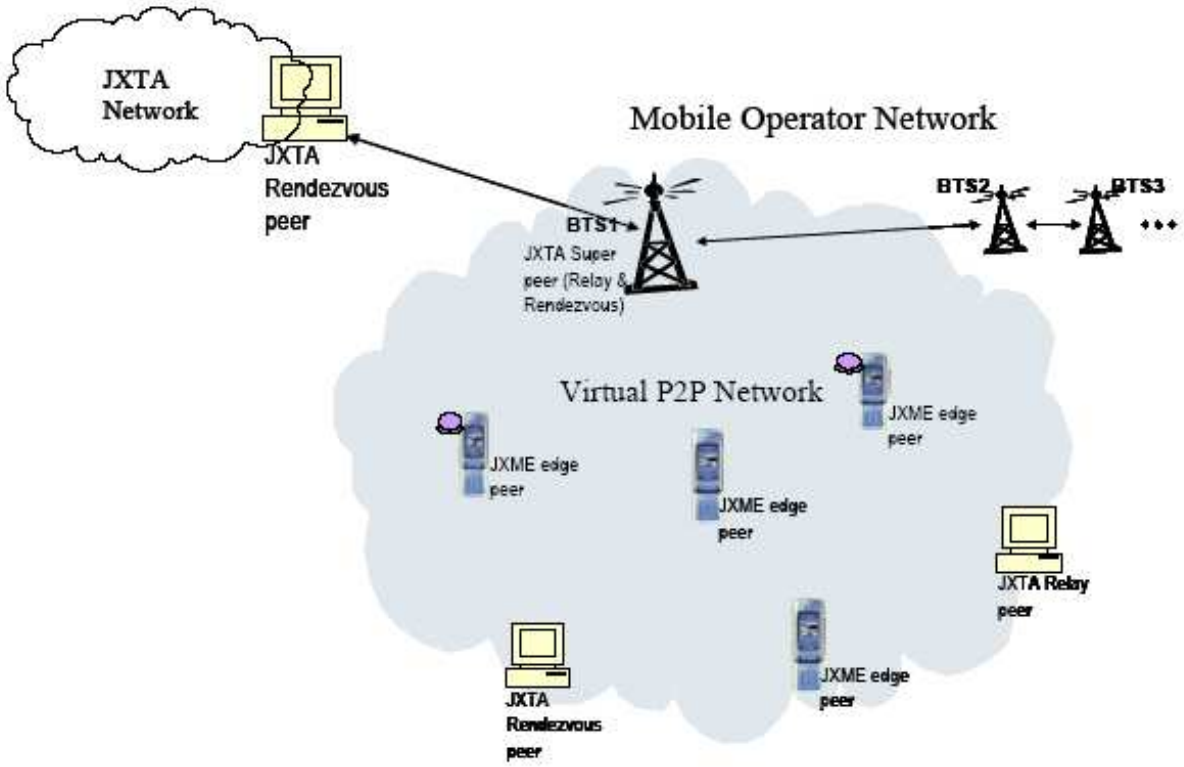


Figure 22 : General Architecture of Mobile Hosts in JXTA network [43].

Figure 22 shows the final architecture by combining mobile Web Services and the JXTA network. We assume that the JXTA network is established in the mobile operator network with a node in mobile network like BTS (Base Transceiver Station) acting as a JXTA super peer (relay and rendezvous peers) however this is only an approach. The BTS is connected to other BTS and JXTA networks, it is like a middleware, provides connectivity to resource restricted devices to connect JXTA network. We are using BTS and this does not mean this project depends purely on that network operator. We are proposing this because we think it is feasible to use BTS as a gateway to serve mobile peers to access outside JXTA network. In fact JXME peers can connect directly to outside relay peers, however our aim to keep them inside the operator network so that it would act like a local area network. This will provide lots of advantage like performance.

JXME edge peers physically connected to JXTA relay peers which we assume it as BTS. Relay peers are special peers used to route messages for other peers within the physical network, more information available at JXTA section. 2.2.4.

Let us examine how operations are performed at above architecture. Mobile hosts and client mobiles join the JXTA/ME network. Mobile Host first creates Web Services then advertises WSDL files using JXTA modules. These module services are advertised as MCA and MSA at JXTA/ME network in specific peer groups like the Web Services group to keep the WSDL advertisement in same group. JXTA services are maintained by the BTS as they act as

rendezvous and relay peers. Once the mobile client application is started client peers joins JXME network. The application shows a screen with search field on it, the user types a few words and starts the search. The search operation is done at JXTA relay/proxy this will be analyzed in more details at implementation section. When the user gets the result of the searched item in a list, a Web service is selected from the list, and search is repeated. This time the JXTA relay/proxy sends a response which contains a link URL where the Web service jar file is located in. The user then downloads and establishes the jar file for invoking Web service.

4. Mobile WS-Discovery Implementation

This section goes through explaining the thesis related development tools, platforms and the implementation detail of Web Services discovery with explaining the achieved steps to combine Web Services together with JXTA/ME network.

4.1. Development Tools/Platforms

The development tools which are used in this thesis work uses Java Platform, Micro Edition (J2ME) where a detailed explanation available at section (2.3). This subsection explains the NetBeans platform with Wireless Toolkit, Mobility pack and Profiler, and Eclipse platform where used to extend functionality of JXTA/ME network.

4.1.1. NetBeans

NetBeans [40] refers to both a ***platform*** for the development of Java desktop applications, and an integrated development environment (***IDE***) developed using the NetBeans Platform. The NetBeans Platform consisting from modular software components called module, where it allows developers to develop applications using there modules. A module is a Java library file that contains Java classes created to interact with the NetBeans Open APIs and a manifest file that identifies it as a module. Modular development provides flexibility like allowing applications to be extended by adding new modules. Since modules can be developed separately and independently, applications based on the NetBeans platform can be easily and powerfully enlarged by third party developers.

The NetBeans IDE (integrated development environment) is an open-source compiler for rapid development of all Java application types like J2SE, web and especially mobile applications where it is used in this project to develop mobile applications for JXME peers. NetBeans is written entirely in Java using the NetBeans Platform. Along with other features are an Ant-based project system, version control and refactoring.

4.1.1.1. NetBeans Mobility Pack [41]

The NetBeans Mobility Pack is a development tool for implementing applications that run on mobile phones. Mobility Pack can be used to write, test, and debug applications for the Java Micro Edition platform (Java ME platform) technology-enabled mobile devices. It integrates support for the Mobile Information Device Profile (MIDP) 2.0, the Connected Limited Device Configuration (CLDC) 1.1. One can easily integrate third-party emulators like WTK (Wireless Toolkit) for a robust testing environment.

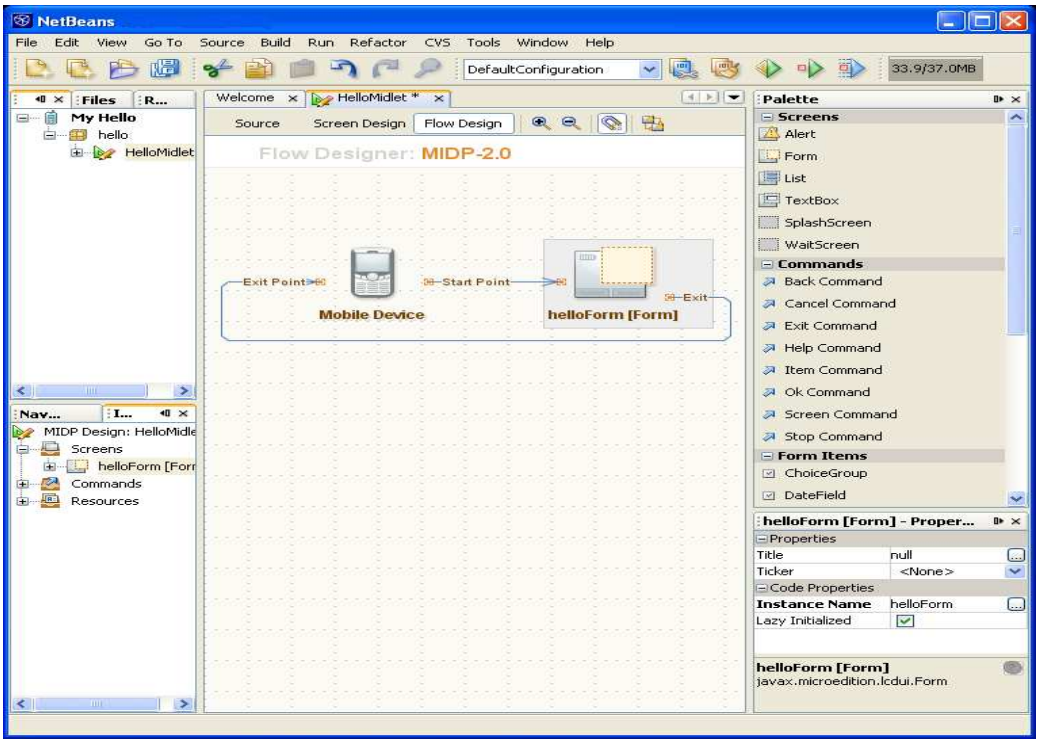


Figure 23 : NetBeans IDE with mobility pack Visual MIDlet screen.

4.1.1.2. NetBeans Profiler [42]

The NetBeans Profiler is a project to provide a full-featured profiling functionality for the NetBeans IDE. Profiler helps to find memory leaks and optimize speed. The profiler is developed from JFluid which it was Sun Laboratories research project. That research exposed specific techniques that can be used to lower the overhead of profiling a Java application.

While the size and complexity of Java applications is raise, keeping applications performance at a required level becomes increasingly difficult. Hence optimizing the applications becomes increasingly important. JFluid technology which is using dynamic byte code instrumentation and additional algorithms, suits the goal of the profiler perfectly. The NetBeans Profiler is able to obtain runtime information on applications that are too large or complex for other profilers.

4.1.1.3. Sun Java Wireless Toolkit

In fact we don't use Wireless Toolkit directly in our project. NetBeans mobility pack includes it by default for testing environment, we can use other toolkits for testing our MIDlets however WTK is one of the free reliable toolkit on the market.

The Sun Java Wireless Toolkit [38] is a group of tools for creating, compiling and deploying Java applications that run on Java Technology for the Wireless Industry specification compliant devices. WTK provides user- friendly development environment for programmers to design and

improve applications on J2ME devices with the help of its build-in tools, utilities and a device emulator for exact offline simulation of the J2ME device to test MIDP applications. The toolkit supports both versions of CLDC (CLDC1.0 and CLDC1.1) and MIDP (MIDP1.0 and MIDP2.0). The other APIs which the toolkit supports are Wireless Messaging API, Mobile Media API, and PDA Optional Packages for the J2ME Platform which consists of file access mechanism, Bluetooth and 3D APIs.

4.1.2. Eclipse

Eclipse [39] is an open source community, whose projects are focused on building an extensible development platform, runtimes and application frameworks for building, deploying and managing software across the entire software lifecycle.

JXTA is large and vide project. Handling, modifying, compiling, deploying and testing such projects needs a capable platform. We made some modification on relay (proxy) peer; detailed modification will be explained in next sections however before that let see what Eclipse is capable of.

The Eclipse SDK contains the Eclipse Java Development Tools. SDK provides an IDE with a built-in Java compiler and a full form of the Java source files. This lets developper for advanced refactoring techniques and code analysis. The IDE also provides out the use of a workspace, in this situation a set of metadata over a flat file space allowing external file modifications as long as the corresponding workspace "resource" is refreshed afterwards.

Plugins

Eclipse works with plug-ins in turn to provide all of its functionality on top of the rich client platform, with respect to some other IDEs where functionality is usually hard coded. This plugin mechanism is a small piece of software construction. Moreover, the plugin framework of Eclipse works with typesetting languages like LaTeX, networking applications such as telnet and database management systems in order to allow Eclipse to be extended using other programming languages such as C and Python.

On the other hand Eclipse can manage mobile applications such as including WTK compiler plugin, however still it is not developed as much as NetBeans. The plugin structural design supports writing any preferred expansion to the environment, such as for configuration management. Java and CVS maintain and support is existed in the Eclipse SDK. It does not have to be used exclusively to support other programming languages.

Widgets

Standard widget toolkit SWT implements for Java environment of Eclipse's widgets, despite of major Java applications, which use the Java standard Abstract Windowing Toolkit (AWT) or Swing. Eclipse's user interface UI also provides a midway GUI layer called JFace, which makes it simpler to construct applications based on SWT.

4.2. Getting started with JXTA Shell

The JXTA Shell is a command line interface to manage, configure, maintain and monitor the JXTA platform. JXTA Shell is built on top of JXTA Platform. JXTA Shell can be configured as a normal peer, Relay peer, Rendezvous peer or JXME proxy peer depending on the requirement. In this thesis work it is configured as relay, rendezvous and JXME proxy peer, so that mobile peers can connect to the JXTA network.

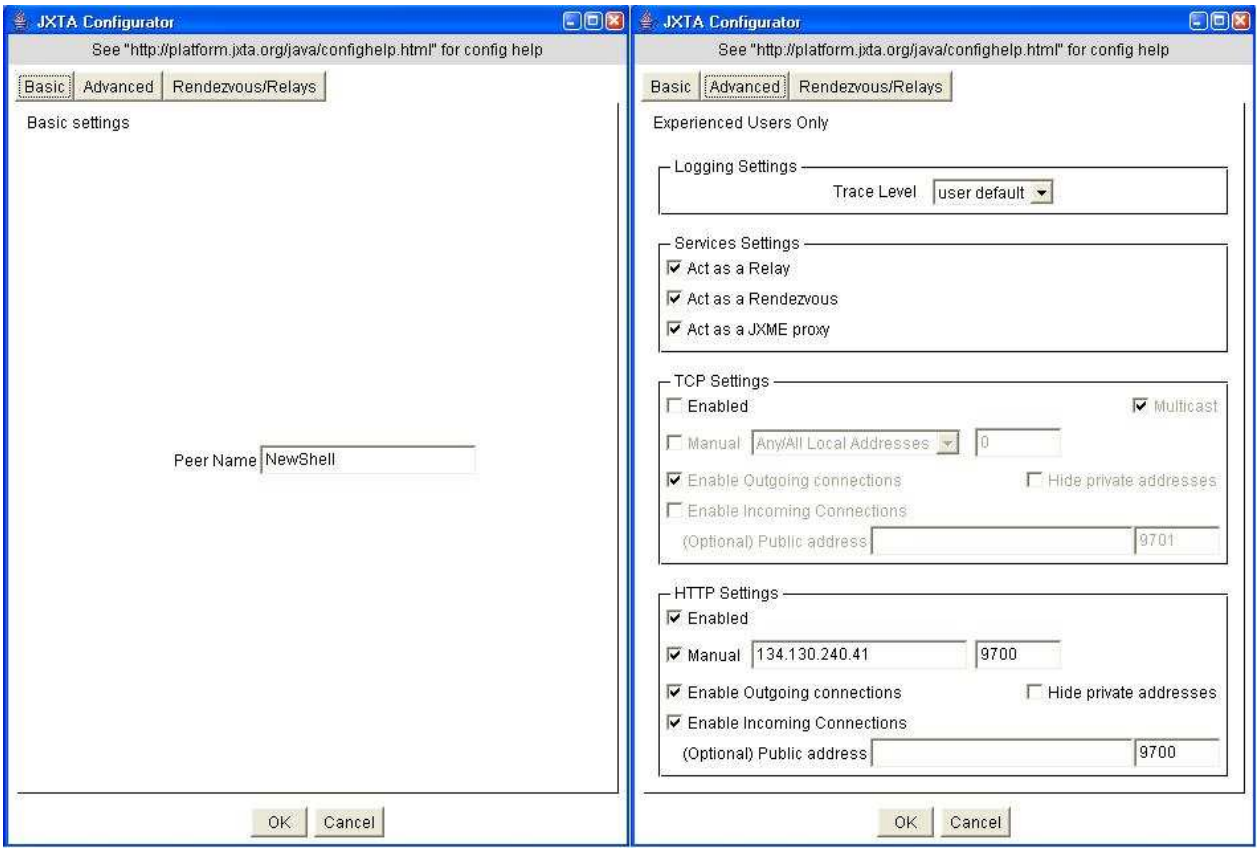


Figure 24 : JXTA Shell configuration screen.

Running and configuring the Shell

You can find and download latest Shell from JXTA web site. For Microsoft Windows users, the JXTA Shell can be run from the Start Menu at: Start->Program Files->JXTA->JXTA Shell. The first time the JXTA Shell is run, a configuration screen will appear. This screen allows you to set up the software to operate with your network. Most users will only have to enter a nickname and a password in the field and leave the rest of the settings at their default values. Additional settings may be adjusted for users behind firewalls or those wishing to behave as a Relay, Rendezvous or JXME proxy peer.

Once you have completed configuration of the JXTA Shell, it will respond with a welcome message and prompt :

JXTA>
Some basic commands for JXTA Shell:

The *peers* command is used to discover other peers.
The *groups* command is used to discover peer groups.
The *join* command is used to join a peer group.
The *search* command is used to search peers, pipes, groups and advertisements.
The *cat* command is used to print content of the advertisements.
The *whoami* command is used to find peer information like ID, IP, port etc.

4.3. WS-Discovery Application Development

This section is going to explain all the implementation of this thesis work. Mainly this section contains of three subsections, those are;
-Service provider is an application where it reads WSDL file and pipe information, and attaches it to a JXTA module class and advertises it at JXTA network.
-JXTA proxy/relay middleware where advanced search and deployment of WSDL files managed.
-JXME Mobile where a mobile peer accessed to JXTA advertisements by advanced search and invoked Web Service through JXTA pipes.

4.3.1. Service Provider Application

This section shows how to integrate or combine a Web Service Definition (WSDL) and JXTA module. As we defined modules in section (3.3. Combining Web Services and JXTA), the JXTA modules consists of three abstract classes, in this section we will examine how WSDL is plugged into Module Specification Advertisement.

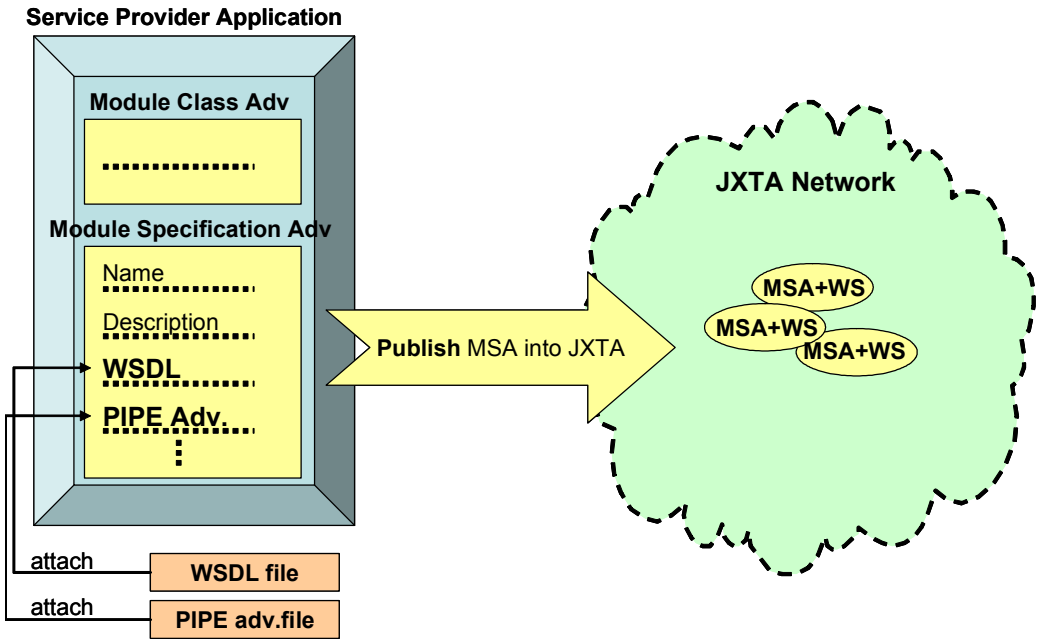


Figure 25 : Service Provider Application

We might thing that why we don't do this implementation on Mobile Web Service Provider side. As we discussed before mobile peers are not capable of handling, creating and publishing JXTA Module Classes. Moreover developers are concerned in advertising WSDL files into JXTA network, this section provides a detailed information on how to do this.

The service provider application advertises the JXTA service with WSDL and Pipe attachment and starts the service as seen in Figure 25. The service associated module spec and class advertisements are published in the NetPeerGroup, this also can be a user defined specific peer group so that the Web service discovery advertisements would be in the same cluster. Clients can discover the module advertisements and create output pipes to connect to the Web Service provider.

In addition to WSDL file as it seen in Figure 25 we are attaching Pipe advertisement, in fact it is not required to attach pipe in order to make WSDL search at JXTA network. Pipe advertisement is required to define the input pipe of the mobile Web Service provider. The purpose of that Web Service invocation can pass messages through JXTA pipes; hence one of the main goals of this thesis work could be done by using JXTA peer or pipe ID instead of IP. This will be explained in section JXME Mobile Application in more detail.

This application defines a single class, it is called ServiceProvider:

main()
This method creates a new ServiceProvider object, calls startJxta() to instantiate the JXTAplatform and create the default net peer group, calls startServer() to create and publish the service.

startJxta()
This method instantiates the JXTA platform and creates the default net peer group :
group = PeerGroupFactory.newNetPeerGroup();

Then it retrieves the discovery and pipe services: The discovery service is used later when we publish our service advertisements.
discoSvc = group.getDiscoveryService();

startServer()
This method creates and publishes the service advertisements. It starts by creating a module class advertisement, which is used to simply advertise the existence of the service. The AdvertisementFactory.newAdvertisement() method is used to create a new advertisement :

```
ModuleClassAdvertisement mcadv = (ModuleClassAdvertisement)
                                 AdvertisementFactory.newAdvertisement(
                                 ModuleClassAdvertisement.getAdvertisementType());
```

It is passed one argument: the type of advertisement we want to construct. After we create our module class advertisement, we initialize it:

```
mcadv.setName("JXTA-WSDL Service Provider");
mcadv.setDescription("Jxta Module WSDL and PIPE service provider");

ModuleClassID mcID = IDFactory.newModuleClassID();
mcadv.setModuleClassID(mcID);
```

The name and description can be any string. A suggested naming convention is to choose a name that starts with "JXTA-WSDL" to indicate this is a JXTA module with WSDL attachment. Each module class has a unique ID, which is generated by calling the IDFactory.newModuleClassID() method.

Now that the module class advertisement is created and initialized, it is published in the local cache and propagated to our rendezvous peer:

```
discoSvc.publish(mcadv);
discoSvc.remotePublish(mcadv);
```

Next, we create the module spec advertisement associated with the service. This advertisement contains all the information necessary for a client to contact the service. For instance, it contains a pipe advertisement to be used to contact the Web service provider and a WSDL file to make an advanced search. At original JXTA the parameter element is defined and implemented, hence we made some modifications at JXTA source code on *ModuleSpecAdvertisement.java* file, so that the param field can be used at MSA. Moreover at proxy *requestor.java* the return types are defined. Similar to creating the module class advertisement, AdvertisementFactory.newAdvertisement() is used to create a new module spec advertisement :

```
ModuleSpecAdvertisement mdadv = (ModuleSpecAdvertisement)
AdvertisementFactory.newAdvertisement(
ModuleSpecAdvertisement.getAdvertisementType());
```

Here a new param type defined as StructuredTextDocument, this types purpose is to carry WSDL file in its parameter.

```
StructuredTextDocument param = (StructuredTextDocument)
        StructuredDocumentFactory.newStructuredDocument(new
        MimeMediaType("text/xml"), "Parm");
```

An Element is created in the param definition with a tag WSDL and string value of the WSDL file where in this example it is *weather.wsdl*, later on the created element is attached to the param type.

```
Element wsdlElem = param.createElement("WSDL",readFile64("Weather.wsdl"));
        param.appendChild(wsdlElem);
```

After the advertisement is created, we initialize the name, description, version, creator, ID, URI and param:

```
mdadv.setName(“weather”);
mdadv.setVersion("Version 1.0");
mdadv.setCreator("weather.com");
```

mdadv.setModuleSpecID(IDFactory.newModuleSpecID(mcID));
mdadv.setSpecURI("http://www. weather.org/Ex1")
mdadv.setParam(param);

We use IDFactory.newModuleSpecID() to create the ID for our module spec advertisement. This method takes one argument, which is the ID of the associated module class advertisement (created above in line)

We now create a new pipe advertisement for our service provider with service provider ID. The mobile client must use the same advertisement to talk to the service provider. When the mobile client discovers the module spec advertisement, it will extract the pipe advertisement to connect to service provider's pipe and create its pipe. We read the pipe advertisement from a default configuration file where mobile service providers pipe ID is located in it.

FileInputStream is = new FileInputStream(FILENAME);
pipeadv = (PipeAdvertisement)
AdvertisementFactory.newAdvertisement(new MimeMediaType("text/xml"), is);
is.close();

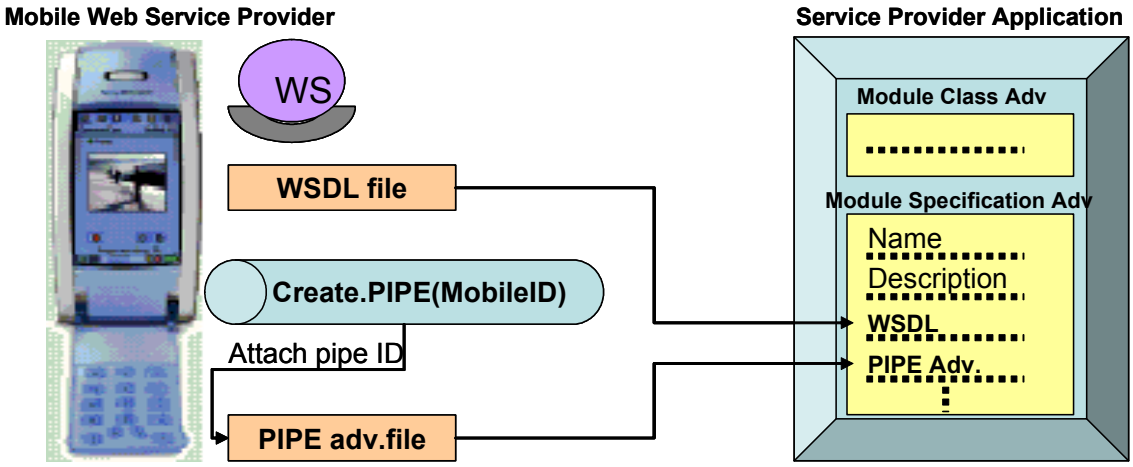


Figure 26 : Attaching Mobile Web Service Providers ID as Pipe ID

On the mobile service provider side we create pipe using peer ID of the mobile peer Figure 266, so that pipe ID of the mobile host will be unique and static. Normally pipe ID is dynamic it generates a new ID on each creation, due to this condition we use peer ID as a pipe ID.

After we successfully create our pipe advertisement, we add it to the module spec advertisement:

mdadv.setPipeAdvertisement(pipeadv);

Now, we have initialized everything we need in our module spec advertisement. We print the complete module spec advertisement as a plain text document , and then we publish it to our local cache and propagate it to our rendezvous peer :

discoSvc.publish(mdadv);
discoSvc.remotePublish(mdadv);

4.3.2. JXTA Proxy/Relay

The HTTP transport has been used in the implementation of JXTA for J2ME (JXME), a compact Java implementation of JXTA intended for devices equipped with the J2ME virtual machine. JXME acts as a HTTP client (using the HTTP client facilities of J2ME) and connects to a standard JXTA peer (voluntarily configured as a JXTA proxy/relay), exchanging messages via the HTTP request/response mechanism. The relay authors and forwards all JXTA messages on behalf of the device, and stores all messages intended for the device. In essence, the relay provides Jxta-enabling services to the device as well as acting as a message proxy.

JXTA relay operations like creating peers, peer groups, pipes, and searching peers, peer groups, pipes and advertisements, all handled and managed by JXTA proxy service and requestor classes. JXME messages are received by JXTA proxy service class and forwarded to requested method/procedure.

Normally the current version of JXTA is supposed to support all above operations, however there are some miss behaviors of some actions like searching module spec advertisements from JXME. After digging the source code of JXTA we realized that we have to modify JXTA proxy service and requestor classes so that search operation could be done for the JXME peers. Moreover to do a deep search in JXTA we have to integrate and use Lucene technology (it is described at section 2.5. Introduction to Project Lucene) at JXTA proxy module.

4.3.2.1 Searching at JXTA proxy/relay

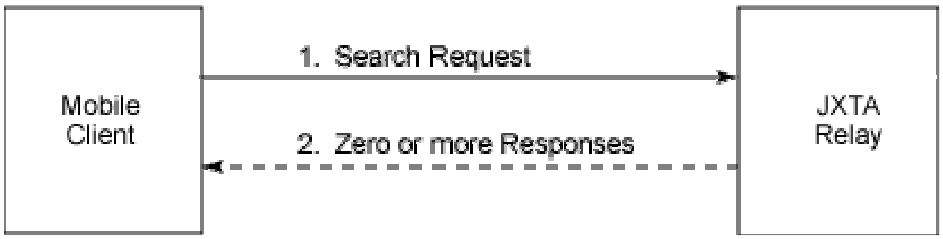


Figure 27 : Request for searching a resource and its response

The search request messages are received by JXTA proxy service to search for pipes, peers, peer groups, and user specific advertisements. Figure 27 represents the search request and its responses from the relay. The number of messages a search request might return depends on the number of resources that successfully match the search criteria. This is represented by a dashed response line in Figure 27.

The search request function is described at mobile peers as below, due to incompleteness of JXTA proxy module this function is not working for advertisement (ADV) type, it is fixed at JXTA proxy module as stated in next paragraphs.
PeerNetwork.search(PeerNetwrok.ADV,"Name",advName,1);

At proxy service class the search request is handled in *handleSearchRequest* procedure. Initially a new proxy type tag is defined as *TYPE_MSA* to recognize module spec advertisement search

request. The main problem at proxy service module was not sending the entire requested response message back to the mobile peer. It was sending only some specific type messages like peer, peer groups and pipes. User specific Advertisements are defined at proxy service and send to the requestor class.

discoveryType = DiscoveryService.ADV;
requestor.send(adv, RESPONSE_RESULT);

Requestor class receives messages from proxy service and checks the ADV weather it is a known instance type to send it to mobile peer. Module Spec Advertisement is defined at requestor class to recognize messages received from proxy service. ADV is checked if it is a type of module spec advertisement:

else if (adv instanceof ModuleSpecAdvertisement) {

The messages attached in a structural way as a message element. TYPE_TAG, NAME_TAG, DESC_TAG and ID_TAG returned to mobile peers as a response, description tag is defined at *ModuleSpecAdv.java* as we mentioned it before.

```
message.addMessageElement(ProxyService.PROXYNS,
                          new StringMessageElement(
                          ProxyService.TYPE_TAG,
                          ProxyService.TYPE_MSA, null ) );
```

Another problem was related with search attributes, by default *PeerNetwork.search* is capable of searching by name, however we need to be able to search by module spec ID (MSID) and description attributes. IdTag and descTag added to create index for search mechanism at *ModuleSpecAdv* class. Moreover "paramTag" parameter tag where it carries WSDL file is also needed to have indexed for deep searching inside WSDL tags, due to structure of param tag and WSDL this aim is not succeed. However deep search mechanism is introduced to handle this problem, this mechanism uses Lucene libraries, it is explained with details at next sections.

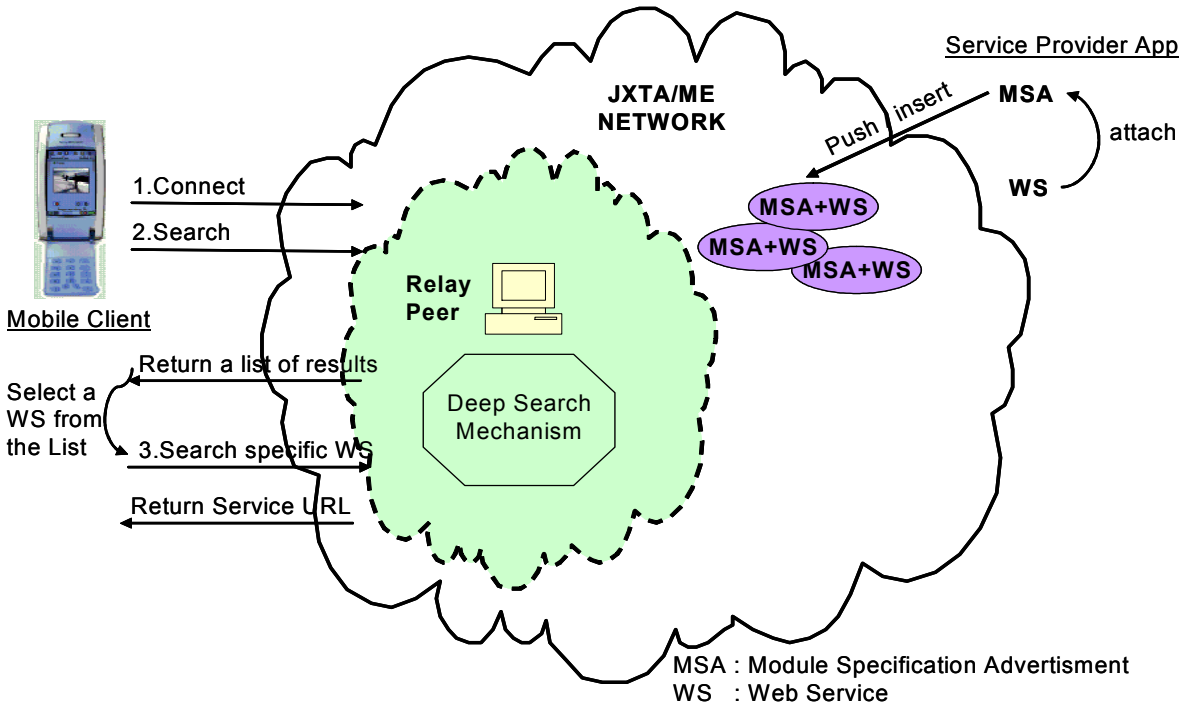


Figure 28 : Searching mechanism at JXTA/ME Network.

Mobile client peer connects to JXTA/ME network Figure 288, JXTA proxy waits for a search request message from mobile peer, as soon as request received by the proxy service, it process the message and prepares the response messages to send back to mobile peer. Mobile peer gets a list of results with name, description and MSID (module spec ID) attributes, the user selects one of the Web service among the list, then the search is repeated with MSID attribute. Preparing Web service for deployment is described in next sections.

4.3.2.2 Deep Search Mechanism at JXTA proxy/relay with Lucene

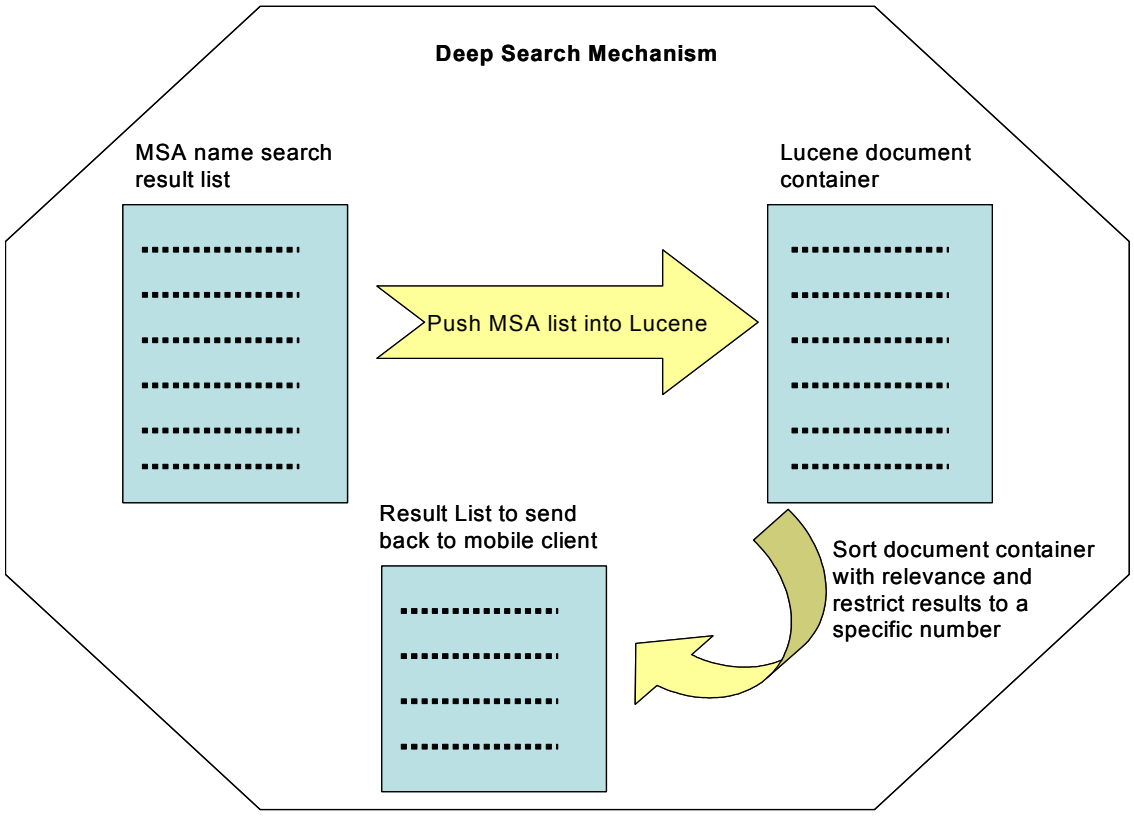


Figure 29 : Deep Search Mechanism Architecture

Once JXTA proxy service receives search request from peer with name or description tag, JXTA does standard searching. Each found result is pushed into al Lucene document container, once standard search finished Lucene makes a deep search using name, description and param attributes where param attributes contain WSDL file. The search is listed with relevance and inserted into container with ID of the results.

Lucene can hold the container in storage disk of device or in random access memory RAM, we preferred to use RAMDirectory.

```
RAMDirectory idx = new RAMDirectory();
```

A developed general-purpose analyzer is StandardAnalyser used in this project, as shown in the following code:

```
IndexWriter writer =
    new IndexWriter(idx, new StandardAnalyzer(), true);
```

Creating document has two field parameters first one is the title as an unindexed field where it is MSID *getModuleSpecID* is used to grab the MSID from module spec advertisement and converted to string. Second field is the content as an indexed field data information is kept on this field. These data informations are extracted by *getContentFromParam* procedure, a combination of name, description and WSDL parameters are returned as a result. All these three methods are available at Appendix A.

```
writer.addDocument(createDocument(modSpecAdv.getModuleSpecID().toString(),
                                  getContentFromParam(modSpecAdv)));
```

Finally searching is done using the IndexSearcher and QueryParser classes. An Analyser object is provided to the QueryParser, this is the same one used during the indexing, where indexing uses the in-memory index. Search method pass values of created searcher and a value to be searched in the container. The search() function returns a Lucene Hits object. This object contains a list of Lucene Documents, in order of relevance. Once user have the Document object, user just retrieve the fields, he wants, using the get() function. The retrieved document results of the search is pushed into a container to be used later by *requestor* class where the result of container is compared with original results and decided to send back to mobile peer.

```
Searcher searcher = new IndexSearcher(idx);
search(searcher, value);
searcher.close();
```

When the proxy service receives search request from mobile peer it forwards the message to *handleSearchRequest* method. When Lucene is introduced inside this method the search operation does not work properly, the standard JXTA search mechanism works in a proper sequence, once this sequence is disturbed the proxy service does not response to requestor class with any result. Hence we made an algorithm without upsetting any working sequence. We do search operation twice in proxy service, first search is normal JXTA search, and the second one is the deep search mechanism which explained above. As a result of deep search mechanism we have a container where the MSID results are kept on it. After search is done requestor class takes the control and prepares the result to be sent back to mobile peer. The below code illustrates the operation done in requestor send method. The results in the container (the results from deep search mechanism) are compared with original results when they match, the results are allowed to be send back to mobile peer.

```
for (int i=0; i<ProxyService.vContainerMSID.size(); i++) {
        if (ProxyService.vContainerMSID.elementAt(i).equals(
                mSpecAdv.getModuleSpecID().toString()))
          sendMsgToMobile = true;
}
```

4.3.2.3 Deploying Web Service

As it described in Figure 3030 mobile peer have a list of Web Services displayed with name and description. When the user selects a WS the mobile peer sends a search request to JXTA proxy service with MSID attribute. This time the proxy service will not use deep search mechanism, normal JXTA search is sufficient to find exact match.

Once the MSA found the WSDL file is extracted from MSA. This operation is done at startMSAtoWSDL method where it takes an MSID parameter to find the service in JXTA network. Then docToXML method is called with document parameter, where the document parameter contains the WSDL param information. WSDL information is extracted to an XML file to be used later on by WS compiler.

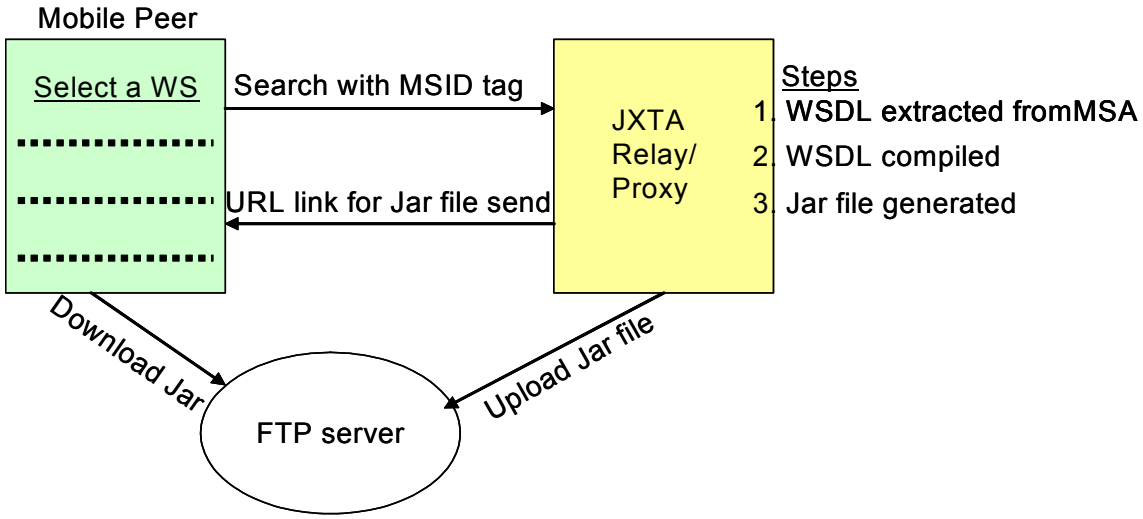


Figure 30 : Generating and compiling WSDL, and deploying it to a server.

After the XML file generated WSDeploy method is called with MSID parameter. WSDeploy compiles the XML file to temporary classes. The temporary classes used to generate Jar file, the jar file name is assigned from MSID to have uniqueness. Then the Jar file is deployed in an IIS or Apache server. Finally a response message is sent back to mobile peer with the URL link. All mentioned methods source code are available at Appendix A.

4.3.2.4 Sequence diagram of the Web Service Search

Service provider application advertises the JXTA service with WSDL and Pipe attachment (as described in 4.3.1. Service Provider Application). Mobile client peer connects to a known JXTA porxy/relay server (Figure 31). Once the peer joins to JXTA network, the client application can search a Web service by providing a name tag. JXTA network sends the result of the query to deep search mechanism (as described in 4.3.2.2 Deep Search Mechanism at JXTA proxy/relay with Lucene). The result of deep search is compared with original results once they match they send back to mobile peer. Mobile peer selects among the result list and redo the search with

specific ID. JXTA proxy/relay server finds the results, compiles and generates the jar file. The file is deployed to a known server to be accessed and download by client peer.

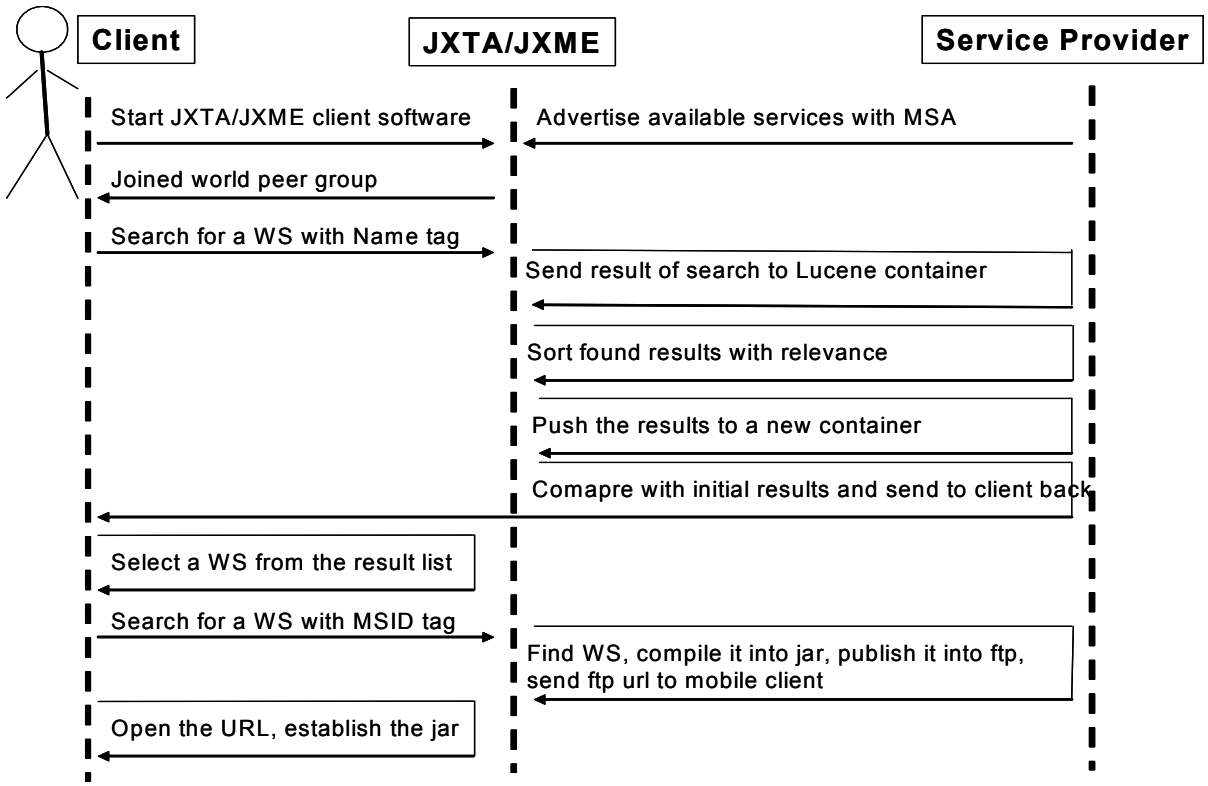


Figure 31 : Sequence diagram of the Web Service Search

4.3.3. JXME Mobile

All the implementations explained above is done at normal peer side, now resource restricted devices implementations are going to be examined. There are two subsections at this section, first one is the JXME client mobile application, and second one is invocation through JXME pipes where port forwarding model used. Before go through subsections lets have a look at JXME API structure to get better understanding of JXME capabilities and restrictions.

JXME API Structure

The JXTA mobile edition JXME uses JXTA relays to connect tiny mobile peers to the rest of the JXTA network. The JXTA relays are also rendezvous JXTA peers that have full capabilities to handle pipes, advertisements, peer group and user defined services. JXME mobile peers communicate with JXTA relays through binary-over-HTTP connections using messages conforming to the JXTA Binary Message format. We need a set of small JXTA APIs for mobile devices to make JXTA networks available to mobile P2P users. The JXME project aims to provide JXTA APIs for the CLDC and MIDP platforms. It can also be used in higher-end J2ME profiles, such as CDC and Personal Profile. This project designed for CLDC and MIDP profiled devices.

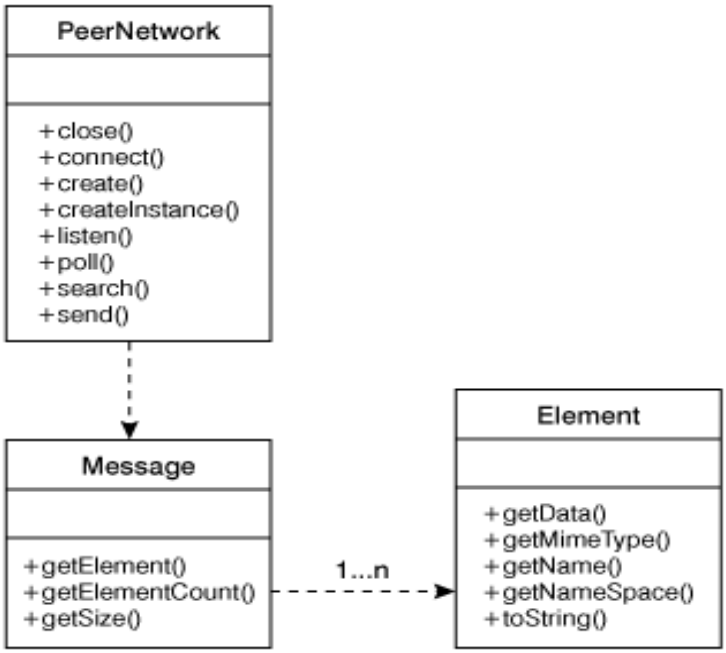


Figure 32 : Classes in package net.jxta.j2me [44].

Let's have a look at into the JXME package itself. The JXME implementation consists of three main classes where all located in the net.jxta.j2me package:

Element symbolize an element inside a JXTA message. An element contains a name, a namespace, a MIME type, and a binary array of data.
Message indicates a JXTA message consisting of several elements. Message provides methods to access those elements.
PeerNetwork is the most useful class. It specifies the JXTA tasks that a mobile peer can perform through the relay. There are several useful methods in the PeerNetwork class:

> ***createInstance()*** is a factory method that returns an instance of PeerNetwork with a specified peer name.
> The ***connect()*** method connects to a relay at a specified HTTP URL. It returns a byte array of persistent state information; this information should be passed through the connect() method in all subsequent connections to the relay.
> The ***create()*** method creates peers, groups, and pipes on the JXTA network through the relay proxy.
> The ***search()*** method searches for peers, groups, and pipes.
> The ***poll()*** method polls the relay for messages addressed to this mobile peer. It can be called iteratively in a server thread.
> The ***listen()*** and close() methods open and close an input pipe, respectively.
> The ***send()*** method sends a message to a specified pipe.

4.3.3.1. JXME Client Mobile Application

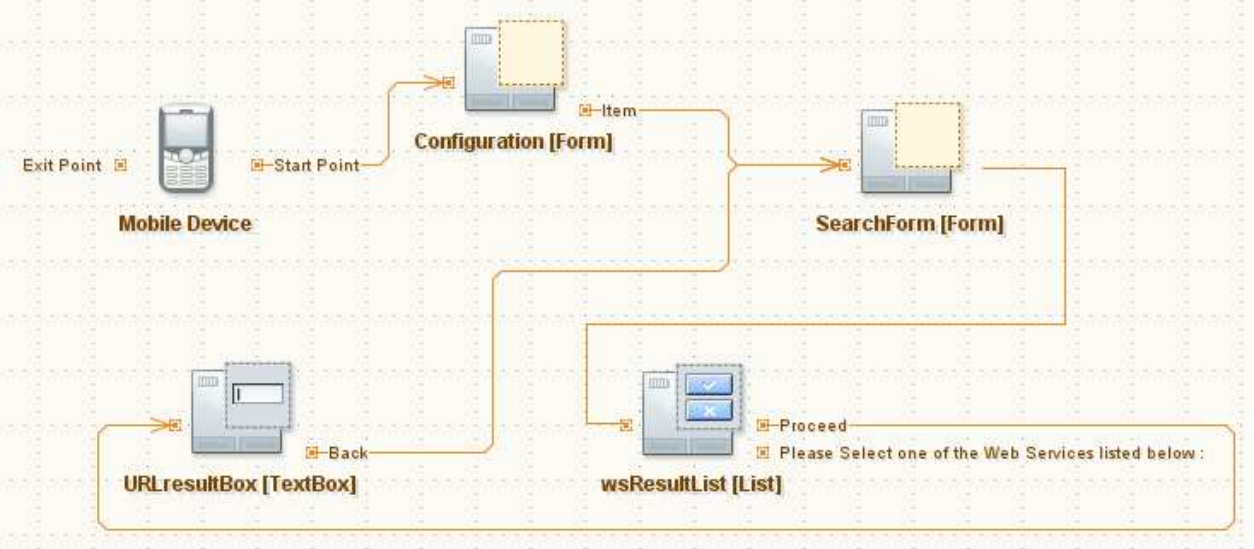


Figure 33 : Flow design of JXME Client Application

When the user starts with client application, a screen appears to enter a known JXTA relay server IP. Once the IP is recognized the mobile application requests a peer identifier from JXTA relay. As soon as it receives identifier response then it requests for connection, relay peer response with connection confirmation message. After a connection is established, a JXME peer can issue other requests to the relay to carry out its operations.

Search form obtains information from user to be searched on JXTA relay peer. *searchPeerAdv* method gets one parameter as a search string where it is *advName*. By using PeerNetwork search method the request is passed to JXTA relay proxy service as described at section 4.3.2.

```
peerNetwork.search(PeerNetwork.ADV, "Name", advName , 100);
```

As mobile devices have not server capabilities to receive messages directly from JXTA relay, a loop started to poll message from JXTA relay. A vector container is created to keep all the result messages received from relay. At the same time the results are append to a list box for the user to have a selection among them.

```
do { msg = peerNetwork.poll(2000);
   v.addElement(new String(msg.getElement(5).getData()));
   wsResultList.append(new String(msg.getElement(3).getData())+
                  " : "+new String(msg.getElement(4).getData()), null);
} while (reciveMsg && loop>0);
```

The user selects one of the Web service from the list and proceeds. This process is the same as above search process, only this time the search parameter will be module spec id. MSID is a unique id where it identifies the web service at JXTA network. Generating WSDL file, compiling and deploying are explained at section 4.3.2. Once the jar file is deployed at a known public server, the URL of this file is sent to mobile peer. When the mobile user obtains the link for jar file it can downloads and installs it.

4.3.3.2. Invocation through Pipes (port forwarding model)

Web services are searched, found and established at mobile devices, now we need to invoke web services. Standard web service invocation is done through IPs, however one of the main purpose of this project was to use peer ID instead of IP. Port forwarding model is described at section 3.3.3 where this model is using JXME pipes to pass messages among peers. The design of this model is explained before now we are going to examine how to implement this.

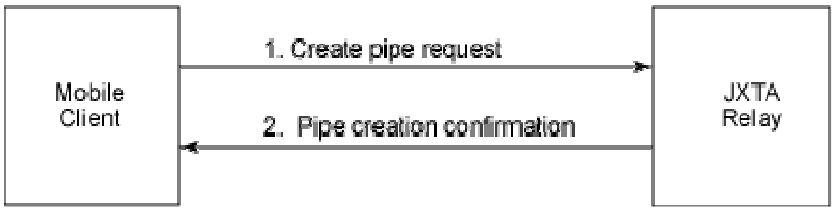


Figure 34 : A request for creating a pipe and its response

When we look from service provider (server) side of mobile application, the server creates a pipe (see Figure 34). To receive incoming messages over the JXTA network it is required to create a pipe. Other peers use the pipe you created to send messages back to you. At this project the pipe is crated using peer ID, to make the pipe ID easy to find and unique. This pipe ID is attached to module spec advertisement while the service provider application pushes the MSA service into JXTA network.

```
peerNetwork.create (PeerNetwork.PIPE, pipeName, peerIDasPipeID
                                , PeerNetwork.UNICAST_PIPE);
```

On the other hand when we look from client side, the peer makes the search and retrieves back a result message containing the service location address URL and pipe ID. The client peer creates a pipe to response back to the message received from server. The Web service message is taken from local http 80 port and combined with client peer pipe ID to send to the server. The client peer ID is required by server to response back to client Web service invocation.

When we look from the big picture a server can be a client also, hence we will consider the implementation in one side. The pipe application consists of three main methods the firs one is already explained above:

1. Creating pipe with peer ID.
2. Listening to pipe.
3. Sending message over pipe.

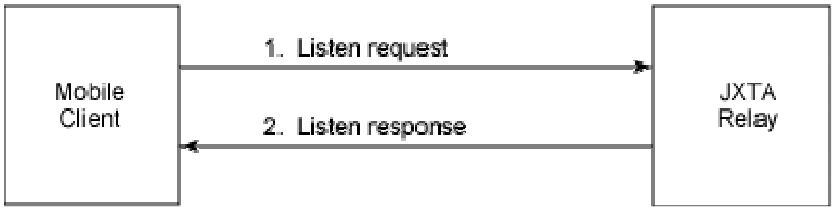


Figure 35 : The request and response sequence for listening pipe action.

For listen a pipe it is required to send a listen request message to the JXTA relay peer. The relay peer starts listening messages for your. By making a loop it is possible to check the relay peer if you have a message or not. This operation is done with poll function which it takes a time variable.

int messageID = peerNetwork.listen (pipeId);
Message msg = peerNetwork.poll(5000);

JXTA relay peer does not forwards received messages to JXME peers immediately, instead it stores the incoming messages. The JXME client contacts the relay (a mechanism called polling) to receive its incoming messages. JXME client always check the relay peer if there is a message for it.

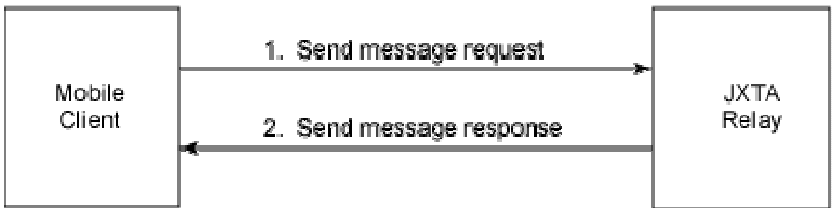


Figure 36 : Request and response for sending a message over a pipe.

Send function is used to send a message through pipes, which it takes a pipe ID and message element. The JXME messages are created with different element fields, each element indicates a different behavior. By default there are many elements at each message. Additionally at thesis work two elements created one is to carry Web service invocation message and the other to carry clients pipe ID.

peerNetwork.send (pipeId, message);

When the mobile host receives the message it forwards it to local port 80 and prepares a WS response message, then the mobile host uses the pipe ID (which was received from client with incoming message) and sends the prepared message to the client mobile. The client mobile takes the message and forwards it locally to port 80. Hence a full invocation process is done.

5. Conclusion

This master thesis work is aimed to provide design and implementation for mobile Web service discovery and invocation through P2P JXTA network. Web service use centralized search mechanism which is called UDDI. This project is aimed to find a solution for discovering Web services in a decentralized manner. Hence at initial stages of this project peer-to-peer technologies are analyzed. After initial digging, JXTA peer-to-peer framework was found most appropriate for the aim of this project, since one of the main goal of this project was to make a WS invocation through JXTA pipes using peer ID without need of IP.

After initial study of JXTA and peer-to-peer technologies, we decided to combine JXTA and Web Services to make mobile Web Service discovery more flexible and dynamic. Combining these two technologies took quite effort and time, since those technologies never considered to be combined for mobile environment. The initial implementation was attachment of Web service to existing JXTA module framework and advertised as JXTA module service. The user is not aware of whether the service JXTA or Web services. To discover WS, JXTA module search is used. Moreover the search capability of JXTA module is extended by using Lucene [30] search mechanism to provide more reliable and relevant results. After solving the WS discovery problem, the invocation of Web service through JXTA pipes is implemented.

At design phase a few existing technique have investigated like SOAP-JXTA [45] and Proxy Model [17]. These techniques gave us a lot of ideas however our domain was mobile devices where those techniques never implemented on mobile devices. In addition there were too many restrictions and limitations at mobile devices. On the other hand some technologies like JXME was immature to handle the thesis aims.

Despite of those restrictions and limitations this thesis work successfully implemented and tested. This thesis work shows how to achieve immature technologies, moreover it shows how to design and implement invocation and discovery of WS at mobile and normal devices using JXTA framework.

6. Future Work

As a future work, deep searching algorithm could be extend by studying more on Lucene [30] technology. Lucene is a very large project and the searching technique where it is used at this thesis work was the basic one. As the number of mobile users increases fast, the Lucene search can be developed to serve mobile peers such that it will response back messages in a very quick way. This could be done by keeping searched indexes in proxy/relay server. Moreover this search mechanism can be extended by context awareness. The client user search request can be kept at a middleware device, when the user makes a new request the kept information could be used to provide better results to the user.

Web services are provided to JXTA network using service provider application. This method may not be too practical, as for each WSDL file this application has to be used. There could be an automation system that will publish these advertisements automatically. Moreover the pipe ID which is taken from mobile host can be advertised without attaching it to module spec advertisement.

Final future direction could be shifting all the operations done at JXTA proxy/relay to mobile phone JXME. There is a proxy-less version of JXTA however at the moment it serves only for CDC devices. As the resources like CPU power and storage capacity etc. of mobile phones develop tremendously, a proxy independent mechanism can be developed to discover and invocate Web services. As a staring point the mobile host [1, 27] can be used as it provides a light HTTP server. In addition Konark [46] project may provide some idea to enhance this design.

List of Figures

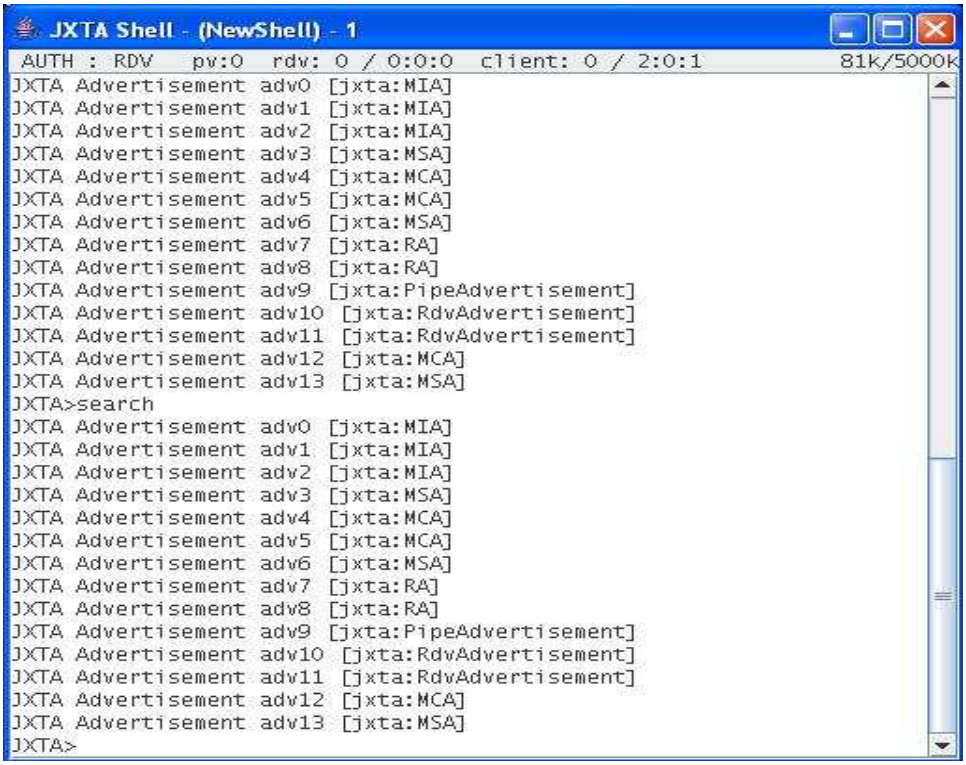
Figure 1 : Web Services architecture [28].....13
Figure 2 : Generation phase of peer-to-peer technologies [15]. 17
Figure 3 : Demonstrates the operating principle of Napster and centralized systems [15,20]. 18
Figure 4 : Demonstrates the operating principle of Gnutella and Decentralized systems [15,20]. 19
Figure 5 : Demonstrates the operating principle of JXTA [15]. 21
Figure 6 : Configurations and Profiles.....23
Figure 7 : MIDP packages 76[35]. 24
Figure 8 : The MIDlet life cycle [37].....25
Figure 9 : JXTA Architecture [26]. 26
Figure 10 : JXTA Virtual Network [25].....27
Figure 11 : JXTA Relay [24]. 29
Figure 12 : The JXTA protocols architecture. 31
Figure 13 : Lucene architecture [31] 33
Figure 14 : Basic architectural setup of Mobile Host [1].....37
Figure 15 : Web Services Conceptual Architecture (by IBM) [16]. 39
Figure 16 : JXTA Architecture (by Sun) [16].....40
Figure 17 : Comparison of JXTA Modules and Web Services. The JXTA module together represent a combination of UDDI in the sense of publishing and finding service description and WSDL in the sense of defining transport binding to the service. 42
Figure 18 : The JXTA SOAP architecture [21].....43
Figure 19 : Proxy Model architecture [21].....44
Figure 20 : Port Forwarding Architecture 45
Figure 21 : Web Services and Client Applications structure. 46
Figure 22 : General Architecture of Mobile Hosts in JXTA network [43]. 48
Figure 23 : NetBeans IDE with mobility pack Visual MIDlet screen.....52
Figure 24 : JXTA Shell configuration screen. 54

Figure 25 : Service Provider Application....56
Figure 26 : Attaching Mobile Web Service Providers ID as Pipe ID 58
Figure 27 : Request for searching a resource and its response....59
Figure 28 : Searching mechanism at JXTA/ME Network....61
Figure 29 : Deep Search Mechanism Architecture 61
Figure 30 : Generating and compiling WSDL, and deploying it to a server. 63
Figure 31 : Sequence diagram of the Web Service Search....64
Figure 32 : Classes in package net.jxta.j2me [44]. 65
Figure 33 : Flow design of JXME Client Application....66
Figure 34 : A request for creating a pipe and its response....67
Figure 35 : The request and response sequence for listening pipe action. 68
Figure 36 : Request and response for sending a message over a pipe. 68

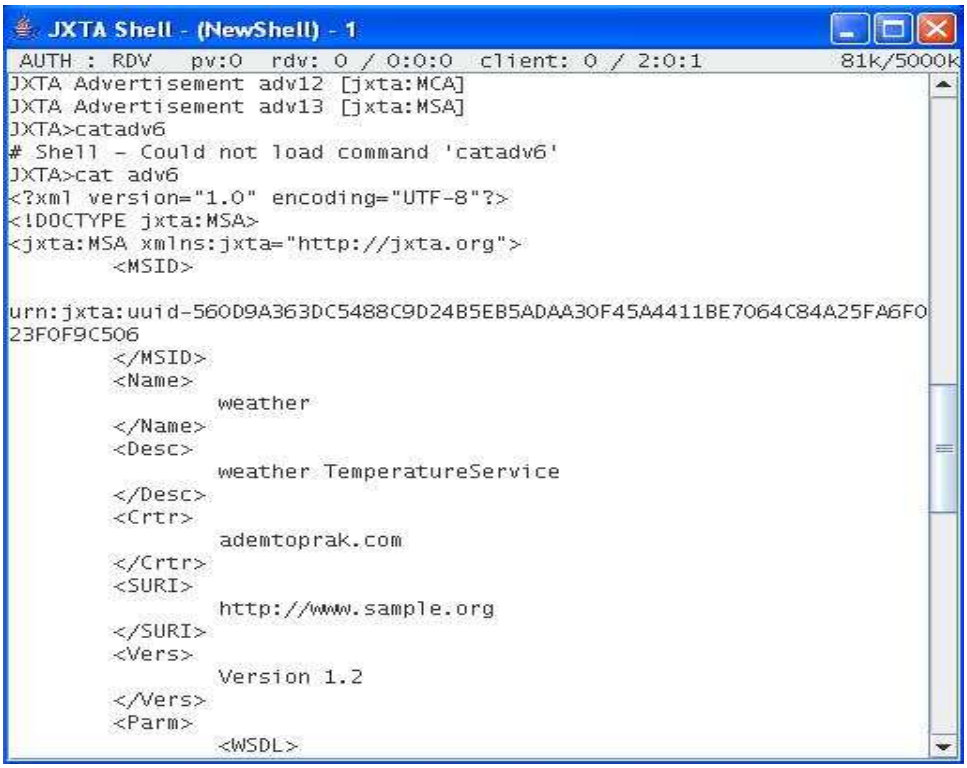
List of Examples

Example 1 : Example of a Module Specification Advertisement.....28

Appendix – Shell and Mobile Application images



JXTA proxy/relay shell screen displaying available advertisements.



JXTA proxy/relay shell screen displaying module spec advertisements content.

Mobile application first image is the configuration screen where IP and port of the proxy/relay peer is entred. Second image displays a box to enter search creteria. Third image shows a result list box where it is allowed to navigate up/down to select a service.

Literature

[1]. S. Srirama, M. Jarke, and W. Prinz. Mobile Web Service Provisioning. In Int. Conf. on Internet and Web Applications and Services (ICIW06). IEEE Computer Society, February 2006.

[2]. SOAP, Simple Object Access Protocol, version 1.1, http://www.w3.org/TR/SOAP

[3]. WSDL, Web Services Description Language, version 1.1, http://www.w3.org/TR/wsdl

[4]. UDDI, The Universal Description, Discovery and Integration, http://www.uddi.org/

[5]. JXTA.ORG. Web: Project JXTA Home. *http://www.jxta.org/*.

[6]. Web Services and JXTA: Companion Technologies http://archive.devx.com/javasr/articles/ohearne/ohearne-1.asp

[7]. SYSTINET. WASP Product Suit. *http://www.systinet.com/products/overview*.

[8]. Connected Limited Device Configuration (CLDC); JSR 30, JSR 139 http://java.sun.com/products/cldc/

[9]. Clay Shirky: "What's P2P and What's Not." http://www.openp2p.com/pub/a/p2p/2000/11/24/shirky1-whatisp2p.html. 11/24/2000

[10]. ICQ.com - community, people search and messaging service! ww.icq.com/

[11]. Skype, Peer to peer voice service. www.skype.com/

[12]. Morpheus P2P file sharing application www.morpheus.com

[13]. Gnutella http://en.wikipedia.org/wiki/Gnutella

[14]. Napster http://en.wikipedia.org/wiki/Napster

[15]. Erkki Harjula1, Mika Ylianttila1, Jussi Ala-Kurikka1, Jukka Riekki2, Jaakko Sauvola:"Plug-and-Play Application Platform Towards Mobile Peer-to-Peer " www.mediateam.oulu.fi/publications/pdf/570.pdf

[16]. Jeff Schneider: Convergence of Peer and Web Services 07/20/2001 http://www.openp2p.com/pub/a/p2p/2001/07/20/convergence.html

[17]. Hajamohideen, Shafeer Huddain(March 2003): A model fro Web Service and Invocation in JXTA. http://www.ti5.tu-harburg.de/publication/2003/Thesis/haja03/haja03.pdf

[18]. Java P2P Unleashed www.samspublishing.com/title/0672323990

[19]. JXTA v2.0 Protocols Specification http://spec.jxta.org/nonav/v1.0/docbook/JXTAProtocols.html#id926357

[20]. BayTSP Corporation: Combating Online Software Piracy in an Era of Peer-to-Peer File Sharing by http://www.baytsp.com/downloads/WhitePaperFinal.pdf

[21]. Backlund Norberg, Mia; Taaveniku, Terése: A Web Service Architecture in Mobile Ad hoc Networks, http://epubl.ltu.se/1402-1617/2005/141/LTU-EX-05141-SE.pdf

[22]. UDDI Explorer http://www.codeproject.com/cs/webservices/UDDIExplorer.asp

[23]. http://en.wikipedia.org/wiki/Tf-idf

[24]. Project JXTA for J2ME - Extending the Reach of Wireless With JXTA Technology, www.jxta.org/project/www/docs/JXTA4J2ME.pdf

[25]. Bernard Traversat, Ahkil Arora, Mohamed Abdelaziz, Mike Duigou, Carl Haywood, Jean-Christophe Hugly: Project JXTA 2.0 Super-Peer Virtual Network. www.jxta.org/project/www/docs/JXTA2.0protocols1.pdf

[26]. JXTA v2.3.x: Java Programmer's Guide Apr 7, 2005 www.jxta.org/docs/JxtaProgGuide_v2.3.pdf

[27]. S. Srirama, "Concept, implementation and performance testing of a mobile Web Service provider for Smart Phones", Master Thesis, RWTH Aachen University, Jun. 2004

[28]. Web Services Architecture http://www.w3.org/TR/2002/WD-ws-arch-20021114/

[29]. Inverse Document Frequency http://www.soi.city.ac.uk/~ser/idf.html

[30]. Apache Lucene home page http://lucene.apache.org/

[31]. Lucene in Action (In Action series) by Erik Hatcher and Otis Gospodnetic.

[32]. Meet Lucene By Otis Gospodnetic and Erik Hatcher http://www.developer.com/java/other/article.php/10936_3490471_1

[33]. Wikipedia the free encyclopedia http://en.wikipedia.org/

[34]. Lucene : a tutorial introduction to full-text indexing in Java. http://www.jroller.com/page/wakaleo/?anchor=lucene_a_tutorial_introduction_to

[35]. Knudsen, Jonathan: Wireless Java™: Developing with Java™ 2, Micro Edtion, Apress, 2001.

[36]. de Jode, Martin: Programming Java 2 Micro Edition on Symbian OS, A Developer's Guide to MIDP 2.0, Apress, 2004

[37]. MIDlet Basics, https://www6.software.ibm.com/developerworks/education/wi-kxml/section3.html (registration (currently free) required to access the site)

[38]. J2ME Wireless Toolkit User's Guide, http://java.sun.com/j2me/docs/wtk2.2/docs/UserGuide-html/

[39]. Eclipse is an open source community http://www.eclipse.org/

[40]. NetBeans Sun open-source platform http://www.netbeans.org/

[41]. NetBeans Mobility Pack http://www.netbeans.org/products/mobility/

[42]. NetBeans Profiler http://www.netbeans.org/products/profiler/

[43]. Srirama, S. (2006) 'Publishing and Discovery of Mobile Web Services in Peer to Peer Networks', International Workshop on Mobile Services and Personalized Environments (MSPE'06), November, Aachen, GI. pp. 99-112.

[44]. Michael Yuan Mobile P2P messaging, Part 2: Develop mobile extensions to generic P2P networks http://www-128.ibm.com/developerworks/wireless/library/wi-p2pmsg2/ 01 Jan 2003

[45]. JXTA-SOAP is a package which allows SOAP communication over the JXTA Peer-to-Peer network. http://soap.jxta.org/

[46]. Sumi Helal, Nitin Desai, Varun Verma and Choonhwa Lee "Konark – A Service Discovery and Delivery Protocol for Ad-Hoc Networks" www.icta.ufl.edu/projects/publications/konark_wcnc2003.pdf