Rheinisch-Westfälische Technische Hochschule Aachen

Lehrstuhl für Informatik V
Prof. Dr. Matthias Jarke

Scalability of P2P Based Mobile Web Services Discovery

Diploma Thesis

Hongyan Zhu
Matriculation number: 232341

Jan 15, 2008

Supervisors: Prof. Dr. Matthias Jarke
Prof. Dr. Wolfgang Prinz

Advisor: M.Sc. Satish Narayana Srirama

Declaration

Hereby with my signature, I assure that I have written this master thesis on my own, and all references cited in the thesis are stated veritably and completely, and any citation is referenced to its source truly.

---------- ----------

Hongyan Zhu Date

Erklärung

Hiermit versichere ich, dass die vorliegende Masterarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht wurden.

---------- ----------

Hongyan Zhu Date

Acknowledgements

At first, I would express my appreciation to Prof. Dr. Matthias Jarke and Prof. Dr. Wolfgang Prinz who provide me the opportunity to do my thesis in their department.

I owe many thanks to my advisor M.Sc. Satish Narayana Srirama for several reasons. At first for his decision to give me the chance to do this thesis in his project, then for his effort and support during the whole period of my work and at last for his belief that I could fulfil the task with success when I am sometimes even sceptical with myself. Without his help and encouragement it is not possible for me to accomplish such work in width and depth.

Many friends and classmates generously provide their advice to improve my work and/or to help solve encountered problems during the process of my work. I often feel grateful and fortunate that I have such kind friends and classmates. Among them are Jing Da, ShuJie Hu, Liang Huang, Edmund Julius, Feng Luo, YanHuai Peng, Qian Yang, Ting Zhang, if I am allowed to mention just a few names.

In the end, no words or language could express the deepest gratitude and love I have to my dear parents and brother for their continuous support and encouragement during the past years of my study in Germany. Their supports give me enormous strength to overcome difficulties on the way to my goal and steadfast confidence in what I pursue in my life.

Abstract

An efficient discovery mechanism is the prerequisite for Web Service requestor to deploy a Web Service with success. UDDI registry is widely used for this purpose. However, this approach on the basis of central registry does not suit the highly dynamic and flexible behaviour of mobile peers. In the quest for a most appropriate mechanism for mobile Web Service discovery various approaches are studied. Among them the P2P technology provides a superb alternative that enables Web Services to discover each other dynamically independent of any centralized registries. JXTA technology is a set of open protocols that enable any connected device on the network, from cell phones to PCs and servers, to communicate and collaborate in a P2P manner. Since each peer joining the JXTA virtual network is identified with a unique peer ID, the interoperable communication between peers is even less bounded and independent of firewalls and network address translations (NATs).

On the basis of the past study of mobile Web Service discovery, this diploma thesis intends to design and implement a more efficient discovery mechanism by borrowing some good characteristics of UDDI such as categorization. With the reference of some popular industry categorization standards a categorization hierarchy is built up. It starts with the root group mobile Web Service Group and consists of peer groups of further three levels hierarchically. The built group structure is a first draft to realize the idea of categorization. To construct a complete categorization hierarchy for all possible mobile Web Services is neither realizable for the present stand of this research nor meaningful for the need of the project now. With the categorization structure it is intended to find out whether the critical and core functionality of the P2P solution – discovery could be improved with efficiency and scalability. Based on the available resources the evaluation and scalability test is conducted and come out with the results that mobile Web Service discovery mechanism with categorization reduces the time cost in fact and is scalable with a small scale of peers.

Table of contents

List of Figures

List of Tables

List of Acronyms

ACK:	Acknowledgement
API:	Application Programming Interface
AJAX:	Asynchronous JavaScript and XML
BPEL:	Business Process Execution Language
CDC:	Connected Device Configuration
CLDC:	Connected Limited Device Configuration
DAML:	DARPA Agent Markup Language
DAML-S:	DARPA Agent Markup Language for Service
DARPA:	Defense Advanced Research Projects Agency
DNS:	Domain Name System
EDGE:	Enhanced Data rates for GSM Evolution
EJB:	Enterprise Java Bean
ERP:	Enterprise Resource Planning
ESB:	Enterprise Service Bus
GPRS:	General Packet Radio Service
HTTP:	Hypertext Transfer Protocol
IDE	Intergraded Development Environment
ISO:	International Organization for Standardization
JXME:	JXTA Java Micro Edition
JXTA:	Juxtapose
J2ME:	Java 2 Platform Micro Editon
MCA:	Module Class Advertisement
MIA:	Module Implementation Advertisement
MSA:	Module Specification Advertisement
MWSMF:	Mobile Web Service Mediation Framework
NAICS	North American Industry Classification System
NAT:	Network Address Translation
OASIS:	The Organization for the Advancement of Structured Information Standards
OWL:	Web Ontology Language
PBP:	Pipe Binding Protocol
PDA	Personal Digital Assistant
PDP:	Peer Discovery Protocol
PIP:	Peer Information Protocol
PRP:	Peer Resolver Protocol
P2P:	Peer to Peer
RCP	Rich Client Platform
RDF:	Resource Description Framework
RDV:	Rendezvous
RPC:	Remote Procedure Call
RTT	Round Trip Time
RVP:	Rendezvous Protocol
SOA:	Service Oriented Architecture
SOAP:	Small Object Access Protocol
SMTP:	Simple Mail Transfer Protocol
TCP/IP:	Transmission Control Protocol/Internet Protocol
TTL:	Time to Live

UBR:	UDDI Business Registry
UDDI:	Universal Description, Discovery and Integration
ULD:	Unit Load Device
UMTS:	Universal Mobile Telecommunications System
UNSPSC:	United Nations Standard Products and Services Codes
URI:	Uniform Resource Identifier
URL:	Universal Resource Locator
URN:	Uniform Resource Name
UUCP:	Unix to Unix CoPy
W3C:	World Wide Web Consortium
WS:	Web Service
WSDL:	Web Services Discovery Language
XML:	Extensible Markup Language

1 Introduction

By 2008 the number of global wireless subscribers should reach 2 billion [Gros04] and the trend keeps growing rapidly. At the same time the development of mobile technology enables mobile devices such as PDAs and smart phones to fulfil tasks almost like a personal computer. Web Service is a software system designed to support interoperatable machine to machine interaction over a network. With Web Services applications could communicate with each other directly so as to exchange data or fulfil a task. The broad range of Web Services and the gigantic user base of mobile devices make the combination of Web Services on mobile devices rather attractive due to the new opportunities for mobile operators, wireless equipment vendors, third-party application developers as well as end users.

1.1 Motivation

There are basically two parties engaged in the Web Services deployment scenario: **Web Services requester entity** and **Web Services provider entity**. A Web Service requester entity is an entity, i.e. an end user or an application, who needs some Web Services for its own purpose and wish to deploy it to satisfy this objective. Web Services provider entity is an entity, i.e. a software vendor or an application, who provides some Web Services to those who need them against payment or for free.

A requester entity with the wish to deploy some Web Services stays in, generally speaking, two possible cases about the source (i.e. Web Services provider entity) of Web Services to be needed: with enough knowledge, or without enough knowledge. Enough knowledge about provider entity enables a requester entity perform the deployment completely and with success. However, because of the spontaneous characteristics of Web Services production and publishing in reality, a requester entity seldom already gets enough information about how to deploy some Web Services prior to deployment. It is usually the case that although a requester entity knows **what** he needs, he does not know **where** to get what he needs and **how** to get it. In this case a vital stage prior to deployment comes on the stage: **Web Services Discovery**.

For those requester entities who need Web Services but possesses no much idea about it yet, Web Services discovery is not only an absolutely necessary but crucial step to help a requester entity to get the information about the Web Services provider and then further the Web Services itself. A variety of technologies are emerging with the attempt to perform the Web Services discovery today. Among them are E-speak, UDDI, Salutation, Jini and UpnP, not to mention a very often used approach – search engine such as Google, baidu and yahoo!. Since the application scope under study in this project is mobile field. What interests us is then the performance of **mobile Web Services discovery**.

So which approach best fits the objectives of Web Services requester in mobile application scenario, an environment where nodes life cycle and prerequisites differs significantly from traditional network with static connection and strong storage and processing capacity? What are pros and cons to choose one approach for Web Services discovery and not the other? Are there particular critical benchmarks to measure mobile Web Service discovery performance? If yes, what are they and how to measure the performance of mobile Web Service discovery according to some

appropriate benchmarks? These are questions which arouse much curiosity and are expected to be answered satisfactorily in this thesis.

At present there are three leading viewpoints of how a discovery service should be conceived: as a registry, as an index or as a peer-to-peer system. UDDI e.g. acts as a central service registry and keeps information about Web Services from WS provider in form of WSDL files in a centralized server. However, the centralized UDDI registry is mainly designed for static networks and therefore could not work as an ideal discovery service for mobile networks. In mobile network nodes may join or leave the network at any time, there may be significant latency between the time a Web Service is updated and the reflection of the update in the central registry. By way of peer-to-peer system the problem could be better handled because nodes are designed to communicate with each other more directly without the need of the existence of a central server.

Recent research shows the feasibility of a mobile web service provider for smart phones (Mobile Host) [SrJP06b]. Thus a smart phone is technically able to act both as Web Services requester and provider in P2P networks. Due to the limitations of centralized registry UDDI can not scale well to serve the large number of services that will be provided by the Mobile Host. An alternative for the service discovery is to publish the information about Web Services in form of WSDL documents to the P2P network by using the JXTA/JXME network [Srir06].

1.2 Thesis Objectives

In the previous work of Mobile Web Services Provisioning Project a model is designed and implemented by combining Web Services and JXTA together to make WS discovery more dynamic, flexible and efficient. However, it is not quite clear whether this peer-to-peer model could actually serve the need of mobile web services better or not in comparison with the traditional UDDI registry way. Thus, **to study the features of different approaches for Web Service discovery so as to find a most appropriate alternative for mobile Web Service discovery is the first objective of this thesis**.

Despite the limitations of UDDI discovery for the application in mobile Web Services, it has its advantageous features such as categorization function to enable a more speedy and efficient Web Services discovery. The second objective of this thesis is **to adapt some of the best features of UDDI to P2P approach so as to improve the model suggested in the previous work of mobile Web Services project**. Some paradigms will be provided to borrow the best features of UDDI into P2P approach.

After the first objective is reached it is presented a complete picture of the advantages and disadvantages of the different approaches of discovery in the application of mobile Web Services. With the knowledge resulted from the comparison the advantages of UDDI are used to make P2P approach work better. However, whether the updated P2P approach is really well scalable for mobile Web Services is still to be studied. And **to check the scalability of the P2P approach with P2P performance analysis is the third objective of this thesis**.

1.3 Thesis Topics

1.3.1 Web Service

In recent years, Web Service, an advanced Web technology has been deployed broadly because it helps enhance communication and collaboration interoperability. Web Services enable business to quickly connect applications with each other independent of hardware, operating systems, or programming environments. Software components from different companies, which may probably reside on different infrastructures, can communicate with each other by way of SOAP and HTTP.

As a specifically distributed service Web Service uses XML to code and decode the data and SOAP to transfer it. XML stands for Extensible Mark-up Language. Although XML is much like HTML, it differs from HTML on employment purposes. HTML focuses on how data looks while XML mainly cares about what data is. HTML is designed to display data while XML is made to describe data. SOAP is a simple protocol based on XML for accessing a Web Service over HTTP. It is platform and language independent and allows getting around firewalls on the Internet. To specify the location of Web Service and the operations that the Web Service expose the WSDL file is used. A WSDL file is just a simple XML document and contains the necessary elements to describe a Web Service.

1.3.2 Peer-to-Peer Systems for mobile devices

Since the foundation and popularity of Napster, Peer-to-Peer technology, which is growing toward distributed systems and getting away from monolithic systems, has become a trend over the last decade from the perspective of software engineering, A **peer-to-peer** (or **P2P**) computer network relies primarily on the computing power and bandwidth of the participants in the network rather than concentrating on a relatively low number of servers. P2P networks are typically used for connecting nodes via largely *ad hoc* connections [Wiki07]. Each party has the same capabilities and either party can initiate a communication session in P2P communications model. A pure peer-to-peer network does not have the notion of clients or servers, but only equal *peer* nodes that simultaneously function as both “clients” and “servers” to the other nodes on the network. This model of network arrangement differs from the client-server model where communication is usually to and from a central server. The development of P2P systems has generally experienced three stages: centralized P2P, decentralized P2P and the combination of centralized and decentralized P2P. In the third-generation P2P architecture according to their different functions peers are categorized as **super-peers** and **edge-peers**. Super-peers act as gateway for edge-peers to P2P network and are also used for NAT and firewall traversal.

In our project the open-source JXTA/JXME development framework is used as P2P platform for mobile Web Service. JXTA belongs to the third generation P2P and is independent of the software and hardware platform because it uses XML messages. A JXTA network consists of peers (*edge-peers*) and super-peers (*rendezvous peers* and *relay peers*). As soon as a peer joins the JXTA platform it is embodied a unique logical, network independent ID and finds its closest rendezvous peer. JXTA for J2ME (JXME) is a light version of JXTA for mobile devices and works currently only in Java Environment.

1.4 Thesis Structure

Chapter 2 presents the concept, technology and current achievements related to the thesis. It includes Service-based Architecture (SOA), the architecture and operations of Web Service, as well as standards employed in Web Service such as XML, UDDI, SOAP and WSDL. Then P2P Technology, its development and P2P systems for mobile Web Service are introduced. Finally some projects with some similar research interests to our project as well as regarding scalability test of JXTA protocols are discussed and the previous work about mobile Web Service discovery in our project is reviewed.

Chapter 3 discusses about the concept to map JXTA to UDDI and Web Services and search mechanisms for Web Services. Various mechanisms of Web Service discovery are introduced from different viewpoints: registry, index, dynamic or P2P. Dynamic and P2P approaches are paid special attention since they are much referenced to design the discovery mechanism for mobile Web Services in our project. Then it is about the designing scenario to adapt categorization in UDDI to P2P based mobile web services discovery. After that the idea to invoke Web Services through JXTA pipes is also presented.

Chapter 4 elaborates the implementation of the P2P based mobile Web Service discovery. It reviews the technology and tools used for the implementation stage briefly such as Java Platform, Eclipse SDK, NetBeans IDE and JXTA/JXME. According to the publishing and discovery sequence the implementation is presented from the aspect of the WS provider, JXME MIDlet and JXME proxy.

Chapter 5 works out the scalability evaluation of P2P based mobile Web Service discovery mechanism. The definition of scalability is firstly studied and then the goals of scalability test and methodology for scalability performance are set and built up. With the benchmark suite the scalability test is conducted in the network with different topologies. Then by analyzing the performance results from the test and comparing them with each other it is intended to get satisfactory answers to the questions raised in goals of scalability test.

Chapter 6 makes a summary of the thesis work and discusses the future work.

2 State of the Art

2.1 Service Oriented Architecture (SOA)

2.1.1 Introduction to SOA

Service-Oriented Architecture is one of many popular technologies in the software world on which people can not agree for a widely acknowledged definition. Service Orientation, from the literal interpretation, describes the fundamental design principle for an architecture which inter-relates an application's different functional units, which are called services, through well-defined interfaces and contracts. Services could be accessed independent of underlying platform implementation. The OASIS defines SOA as the following [OASI06]:

A paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. It provides a uniform means to offer, discover, interact with and use capabilities to produce desired effects consistent with measurable preconditions and expectations.

.

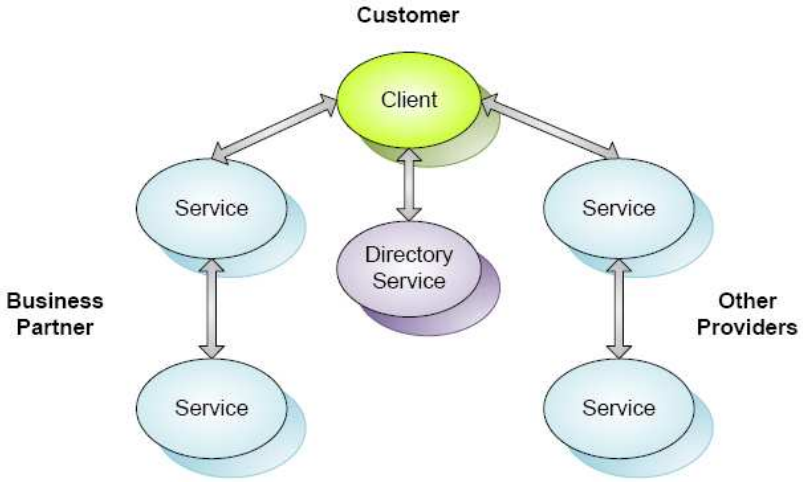


FIGURE 2-1: A SERVICE ORIENTED STRUCTURE [ORT05]

The key characteristic that distinguishes SOA from other technologies lies in loose coupling. Loose coupling means that the client of a service is essentially independent of the service as in **FIGURE 2-1** [Ort05]. A main obstacle in reusing the existing business applications is the uniqueness of specific applications and systems, since the applications are usually coded in different languages, run in different operating framework and deploy different interfaces. It would be rather time-consuming to analyze those different underlying implementations in order to communicate with the existing applications and deploy them for your own use. With SOA the client just needs to communication with the service by a well-defined interface. Neither needs a client know the implementation nor does he care the possible changes since all these odds are just under the control of service provider. To use Google Search Engine on your own homepage, for example, you need little knowledge about its implementation. So long as the search interface remains the same, the search function will work as usual no matter the search engine is under revision or not. In this way, the clients and

loosely-coupled services interact and communicate with each other widespread and with great flexibility.

2.1.2 SOA and Web Service

Many IT professionals foresee the strong trend of SOA, especially Web Service-based SOA, in speeding up the application development process. With SOA IT becomes more agile in responding to the changing business needs. SOA is clearly more a wave of future than just a hot topic. Gartner[1] reports "By 2008, SOA will be a prevailing software engineering practice, ending the 40-year domination of monolithic software architecture." And that "Through 2008, SOA and Web Services will be implemented together in more than 75 percent of new SOA or Web Service projects."

A Web Service is a service that communicates through predefined and commonly acknowledged standard protocols and technologies. All the major software vendors develop their products conforming to these standards so as to keep the communication between clients and services consistent despite the wide spectrum of different platforms and operating environments. When we are considering the reasons that made Web Service the most prevalent approach to implementing an SOA, some of them must be [Ort05]:

- **Reusability**. The functions of an application or a system can be dramatically easier to access as a service in an SOA than in some other architecture. So integrating applications and systems can be much simpler.

- **Interoperability**. The maturing set of protocols and technologies are not only platform, system and language independent, but also work across firewalls, which makes it possible for business partners to share vital services.

- **Scalability**. Applications that use services in a loosely-coupled SOA tend to scale easily than in a tightly-coupled environment because there are few dependencies between the requesting application and the services it uses. In comparison with a tightly-coupled service in some other architecture, which takes numeric value or java objects as input, the service in a web-service based SOA is loosely-coupled and accepts a document as input. The very limited interaction required for a client to communicate with a SOA service allows applications using the service to scale without putting a heavy communication load on the network.

- **Flexibility**. The coarse-grained, document-oriented and asynchronous nature of services in SOA make it easier to evolve with changing requirements than in a fine-grained[2] architecture where different components of an architecture are bound to each other.

[1] **Gartner Inc**. is an information and technology research and advisory firm in USA. See http://www.gartner.com

[2] Fine-grained and coarse-grained architecture mean tightly-coupled and loosely-coupled architecture respectively.

2.2 Web Service

2.2.1 Overview

In this section a very general introduction of Web Service architecture and operations is given. Section 2.2.1.1 includes widespread acknowledged standards of Web Service. And Section 2.2.1.2 tells about the parties and operations involved in Web Service process.

2.2.1.1 Web Service Architecture

Web Service is briefly introduced in **section 2.1** as a frequently used approach to implement SOA. As **FIGURE 2-2** shows us the Web Service approach is based on a maturing set of standards that are widely accepted and used. These widespread accepted standards make it possible for clients and services to communicate and understand each other across a wide variety of platforms and across language boundaries. Among them the standards such as ***Extensible Mark-up Language (XML), Small Object Access Protocol (SOAP), Universal Description, Discovery and Integration (UDDI) and Web Services Description Language (WSDL)*** address the basics of interoperable services. They ensure that a client could find the service he needs and make a request in the way that the service provider understands independent of the languages in which the client and the service are coded or in which platform they reside. In this section, the basic standards XML, SOAP, UDDI and WSDL will be introduced in detail. Other high-level standards in the area of web service security and web service management also need to be adapted in order to make Web Services into mainstream IT practice. The standards organizations such as World Wide Web Consortium (W3C) have also drafted standards in the areas such as WS-Security and WS-BPEL. However, the security and management issues are not the focus of the thesis and thus will not be discussed further in detail.

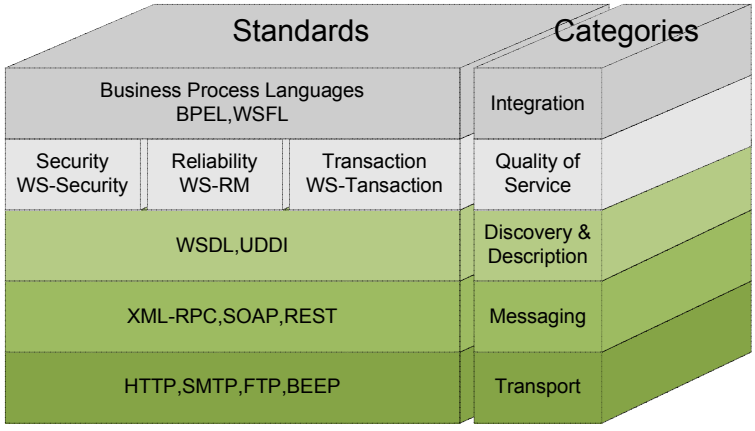


FIGURE 2-2: WEB SERVICE ARCHITECTURE STACK [HALL06]

2.2.1.2 Web Service Operations

The purpose of a Web service is to provide some functionality on behalf of its owner such as a business or an individual. A ***provider entity*** is the person or organization that provides an appropriate agent to implement a particular service. A ***requester entity*** is a person or organization that wishes to make use of a provider entity's Web service. It will use a *requester agent* to exchange messages with the provider entity's *provider agent*.

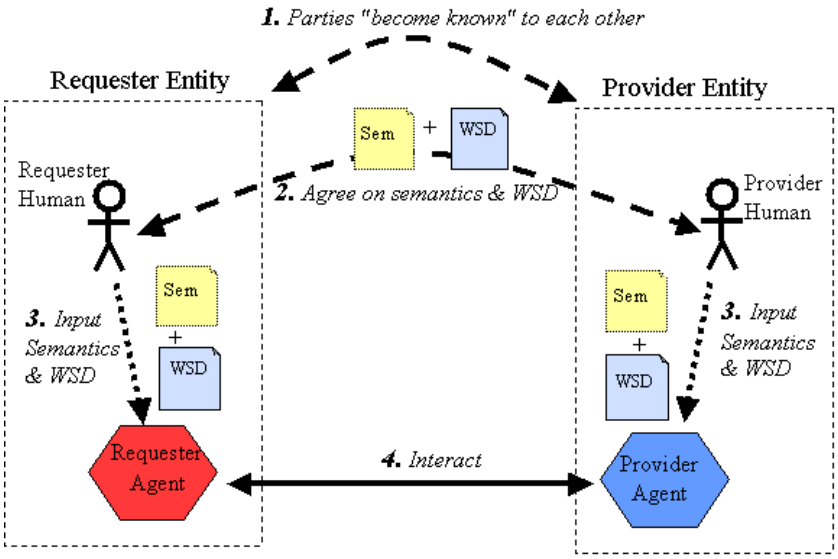


FIGURE 2-3: WEB SERVICES PROCESS [BHMN04]

The process engaging in a web service usually includes the following four stages:

1. The requester and provider entities "become known to each other", in the sense that whichever party initiates the interaction must become aware of the other party. There are two cases accordingly: either the provider or requester agent plays the role of the initiator. The former case that the provider agent somehow has already obtained the address of the requester agent is practically less common and this case is not much of our interest for research either. The typical case is that the *requester* agent will be the initiator with no idea of the address of the requester agent before. A discovery service is then necessary for the requester to locate a suitable service description which contains the provider agent's invocation address. The detailed process of service discovery will be fully described in **Section 3.1**.

2. The requester entity and provider entity agree on the service description (a WSDL document) and semantics that will govern the interaction between the requester agent and the provider agent. This does not necessarily mean that the requester and provider entities must communicate or negotiate with each other. It simply means that both parties must have the same understandings of the service description and semantics, and intend to uphold them. There are many ways this can be achieved.

The provider entity may publish and offer both the service description and semantics as take-it-or-leave-it "contracts" that the requester entity must accept unmodified as

conditions of use. The service description and semantics may be defined as a standard by an industry organization, and used by many requester and provider entities [BHMN04].
3. The information in the service description and semantics must either be input to, or implemented in, the requester and provider agents. There are many ways this can be achieved. Whatever the approach is, it must be achieved before the two agents can interact.

4. The requester agent and provider agent exchange SOAP messages on behalf of their owners.

2.2.2 Web Services Standards

Some of the very often used Web Services standards in this project are presented in this section with some details. They include XML, UDDI, SOAP and WSDL.

2.2.2.1 XML

XML stands for ***Extensible Mark-up Language*** and has become the standard to describe data to be exchanged on the web. Unlike HTML, which was designed to display data and focuses on how data looks, XML was designed to describe data and focuses on what data is. And XML describes the content of a document by using tags to “mark up” the content. A tag identifies the information of a document as well as the structure of the information. The grammatical rules of XML are in form of ***XML Schema***. XML Schema specifies e.g. what types of tags are allowed, what type of data is expected in a tag. Because of these strictly formulated rules XML documents are well-formed and easy to be processed.

2.2.2.2 UDDI

As mentioned in **section 2.2.1.2**, when the parties engaged in web services are unknown to each other, web service discovery is usually needed for the requester entity to find the related information about the service he is searching for. ***Universal Description, Discovery, and Integration (UDDI)*** Registry provides a standardized method and is one of the paradigms for publishing and discovering information about Web Services. UDDI is an industry initiative that attempts to create a platform-independent, open framework for describing services, discovering businesses, and integrating business services [Wals02]. UDDI involves four core data models: **businessEntity**, **businessService**, **bindingTemplate** and **tModel**.

- **businessEntity** is the top level UDDI element and it contains the name, description, identifiers and classifications for an organisation entity. Each **businessEntity** may publish one or more **businessService** definitions. These definitions are abstract so they just say what the service does but not how it does it.

- Each **businessService** is linked to one or more **bindingTemplates** that specify where the service is located (URL) and how to invoke it. The same service might have many binding templates if it can be accessed by way of several different protocols.

- Each **bindingTemplate** contains a set of keys that link the binding template to more detailed technical specifications (“**tModel**s”) such as WSDL files, XML schema, or protocol definitions. The **tModels** can be regarded as “reference standards” and may be re-used by several different business entities. For example, a **tModel** that

defined the http transport protocol is likely to be referenced by many **bindingTemplates** [Rior06].

Conceptually, a business can register three types of information into a UDDI registry and they are named in the art of phone registry.

White pages: Basic contact information and identifiers about a company, including business name, address, contact information and unique identifiers. This information allows others to discover your Web Services based upon your business identification.

Yellow pages: Information that describes a Web Services using different categorizations (taxonomies). This information allows others to discover your Web Services based upon its categorization.

Green pages: Technical information that describes the behaviours and supported functions of a Web Services hosted by your business. This information includes pointers to the grouping information of Web Services and where the Web Services are located.

2.2.2.3 SOAP

The structure and meaning of XML tags provides an efficient way to exchange data. However, it is not enough to exchange data over the Web. Some agreed-upon protocols are still needed to format XML document so that the requester has some idea about which part of the document is of main importance and which part is additional.

Small Object Access Protocol (**SOAP**) is such an XML-based protocol to transport data to and from the web server. It provides a way to communicate between applications running on different operating systems with different technologies and programming languages. The basic item of transmission in SOAP is a SOAP message, which consists of a mandatory **SOAP Envelope**, and optional **SOAP Header**, and a mandatory **SOAP Body** (see **FIGURE 2-4**). The Envelope identifies the XML as being a SOAP message and must be the root element of the message. The Body element contains the message payload. The Header element provides an extension hook that allows SOAP to be extended in arbitrary ways [Sour04].

Envelope is the root element of SOAP. It specifies two things: an XML ***namespace*** and an ***encoding style***. The XML ***namespace*** is designed to avoid name clashes by specifying the names that can be used in the SOAP message. The ***encoding style*** identifies the data types recognized by the SOAP message.

Header serves as a container for extensions to SOAP. No extensions are defined by the specification, but user-defined extension services such as transaction support, locale information, authentication, digital signatures etc. could all be implemented by placing some information inside the **Header** element. The **Header** element is typically used to convey security-related information to be processed by runtime components. Children of the Header element may be annotated with the ***mustUnderstand*** and/or ***actor*** attributes.

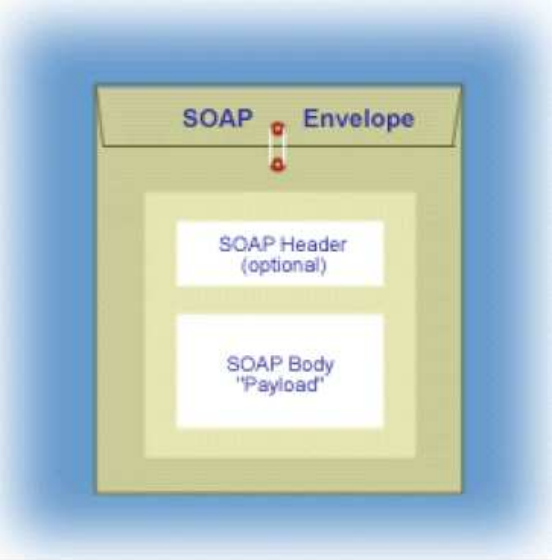


FIGURE 2-4 CONCEPTUAL SOAP MESSAGE STRUCTURE [ORT05]

Body contains the main part of the SOAP message, that is, the payload intended for the final recipient of the SOAP message. In the case of a request message the payload of the message is processed by the receiver of the message and is typically a request to perform some service and sometimes to return some results. In the case of a response message the payload is typically the results of some previous request or a fault. The optional SOAP **Fault** element is used to hold error and status information for a SOAP message and only appears in response messages.

2.2.2.4 WSDL

WSDL stands for **Web Services Description Language** and is an XML grammar for describing a Web Services as a collection of access endpoints capable of exchanging messages in a procedure- or document-oriented fashion [Wals02]. It specifies the location of the service and the operations (or methods) the service exposes. A WSDL document describes a web service as a collection of abstract items called “ports” or “endpoints” and use these major elements:

- **definitions** The definitions element must be the root element of all WSDL documents. It defines the name of the web service, declares multiple namespaces used throughout the remainder of the document, and contains all the service elements in the document.

- **types** The types element describes all the data types used between the client and server. WSDL is not tied exclusively to a specific typing system, but it uses the W3C XML Schema specification as its default choice. If the service uses only XML Schema built-in simple types, such as *strings* and *integers*, the types element is not required.

- **message** The message element describes a one-way message, which is either a single message request or a single message response. It defines the name of the

message and contains zero or more message part elements, which can refer to message parameters or message return values.

- **portType** The portType element combines multiple message elements to form a complete one-way or round-trip operation. For example, a portType can combine one request and one response message into a single request/response operation, most commonly used in SOAP services.

- **binding** The binding element describes the concrete specifics of how the service will be implemented on the wire. WSDL includes built-in extensions for defining SOAP services, and SOAP-specific information therefore goes here.

- **service** The service element defines the address for invoking the specified service. Most commonly, this includes a URL for invoking the SOAP service [Cera02].

2.3 Peer-to-Peer Systems

2.3.1 Introduction to P2P Systems

A **peer-to-peer** (or **P2P**) computer network relies primarily on the computing power and bandwidth of the participants in the network rather than concentrating it in a relatively low number of servers. P2P networks are typically used for connecting nodes via largely *ad hoc* connections [Wiki07]. P2P is a communications model in which each party has the same capabilities and either party can initiate a communication session. Since the foundation and popularity of Napster, P2P has become a trend over the last decade from the perspective of software engineering, which is growing toward distributed systems and getting away from monolithic systems.

A pure peer-to-peer network does not have the notion of clients or servers, but only equal peer nodes that simultaneously function as both “clients” and “servers” to the other nodes on the network. This model of network arrangement differs from the client-server model where communication is usually to and from a central server. A typical example for a non peer-to-peer file transfer is an FTP server where the client and server programs are quite distinct, and the clients initiate the download/uploads and the servers react to and satisfy these requests.

A more technical definition of P2P was put together by Dave Winer of UserLand Software. According to him a P2P system should have the following seven characteristics [BMCM04]:

- User interfaces load outside of a web browser.
- User computers can act as both clients and servers.
- The overall system is easy to use and well integrated.
- The system includes tools to support users wanting to create content or add functionality.
- The system provides connections with other users.
- The system does something new or exciting.
- The system supports “cross-network” protocols like SOAP or XML-RPC.

2.3.2 Development of P2P Systems

The earliest peer to-peer network in widespread use was the Usenet news server system, in which peers communicated with each other to propagate Usenet news articles over the entire Usenet network. Particularly in the earlier days of Usenet, UUCP[3] was used to extend even beyond the Internet. However, the news server system also acted in a client-server form when individual users accessed a local news server to read and post articles. The same consideration applies to SMTP[4] email in the sense that the core email relaying network of Mail transfer agents is a peer-to-peer network while the periphery of Mail user agents and their direct connections is client server.

Some networks and channels such as Napster (See Figure 2-8), OpenNAP and IRC server channels use a client-server structure for some tasks such as searching and a peer-to-peer structure for others. Networks such as Gnutella (See Figure 2-9) or Freenet use a peer-to-peer structure for all purposes, and are therefore referred to as pure peer-to-peer networks, although Gnutella is greatly facilitated by directory servers that inform peers of the network addresses of other peers [Grad05].

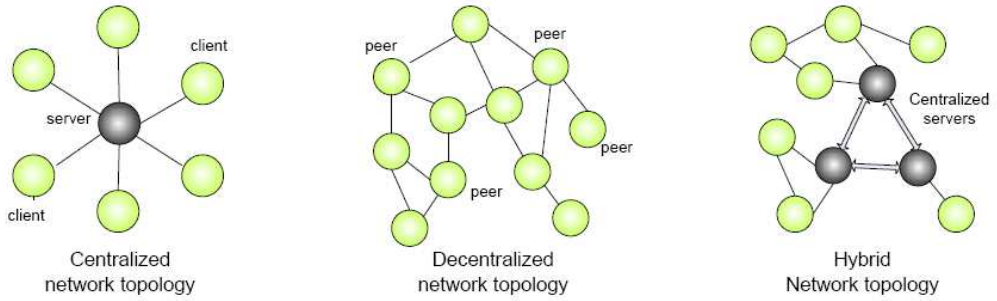


FIGURE 2-5 DIFFERENT NETWORK TOPOLOGIES

2.3.2.1 First Generation: Hybrid P2P

The Internet was originally built as a peer-to-peer system in the late 1960s to share computing resources within the US. The first hosts on ARPANET were connected together as equal peers rather than as client-server. The main users at the beginning were computing researchers who did not need protection against each other, and security break-ins were practically non-existent, making the Internet much less partitioned than it is today. Many peer-to-peer systems were widely used and still in existence today such as Usenet and DNS. However, since 1994 the structure of the Internet changed dramatically with millions of people flocking to the Net. Modem connection protocols were widely used and applications were then targeting slow speed analogue modems. Applications such as web browsers were based on a client-server protocol. The Structure of the Internet made a switch from peer-to-peer to

[3] **UUCP** stands for **Unix to Unix CoPy**. The term generally refers to a suite of computer programs and protocols allowing remote execution of commands and transfer of files, email and net news between computers.

[4] **Simple Mail Transfer Protocol** (**SMTP**) is the *de facto* standard for e-mail transmissions across the Internet.

the client-server model (see **FIGURE 2-5**). A Client-Server system is one in which there is a dominant computer (the Server), that is connected to several other computers with less control (the Clients). Clients can communicate with other clients only through the Server.

Compared with Client-Server model, each peer on the network may act as both a client and server in a peer-to-peer system. Clients in a P2P network can interact freely with other clients without the intervention of a server although sometimes there is the presence of a directory server for look up purposes. Among the applications which are still using peer-to-peer model Napster was one of the most successful. It started with the idea to find a easier way for music listener to share their recordings with each other. Before Napster there were online recordings on the Internet and by using MP3 compression format music tracks could be transferred onto disk files and then published on a website for users to download them by way of FTP. However, one major problem was that up-to-date MP3 files were difficult to find. To solve this problem Napster provided a constant up-to-date central database with index of all shared files (see **FIGURE 2-6**). Users could register with the searchable Napster network name space and find files easily through Napster servers, which had information on registered hosts and MP3 data. The servers dealt with the transfer of files between clients but didn't actually store any of the music themselves. Napster's network protocol created direct peer-to-peer access between clients. Thanks to its simplicity of use by way of peer-to-peer model Napster got a huge success.

If we analyse the hybrid P2P architecture for mobile Web Services discovery, two main shortcomings can not be neglected: single point of failure and lack of scalability. The whole system collapses when server does not work well. The architecture does not fit for the mobile environment where each node may join or leave the network any time. On the other hand centralized systems produce giant communication traffic and storage on server and the system is not well scalable.

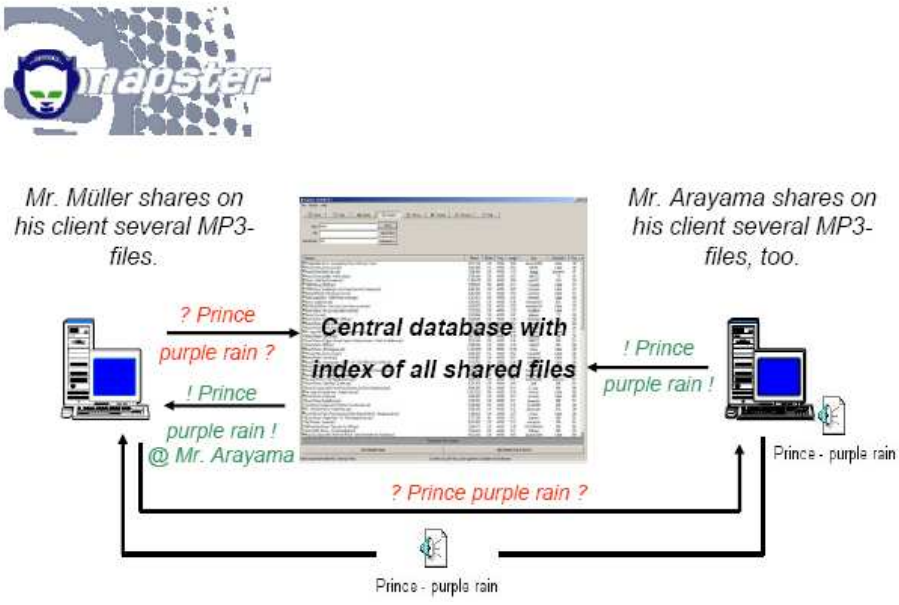


FIGURE 2-6 NAPSTER ARCHITECTURE [WEHR06]

2.3.2.2 Second Generation: Pure P2P

After Napster was involved into legal issues due to copyright problem, the designers of Gnutella learnt from the weakness of Napster P2P model and deploy no central server to store the names and locations of all the available files. Instead, there are many different client applications available to access the Gnutella network. By deploying an extremely simple and clever way Gnutella software distributes a query to thousands of machines very quickly. It works as follows (see FIGURE 2-7):

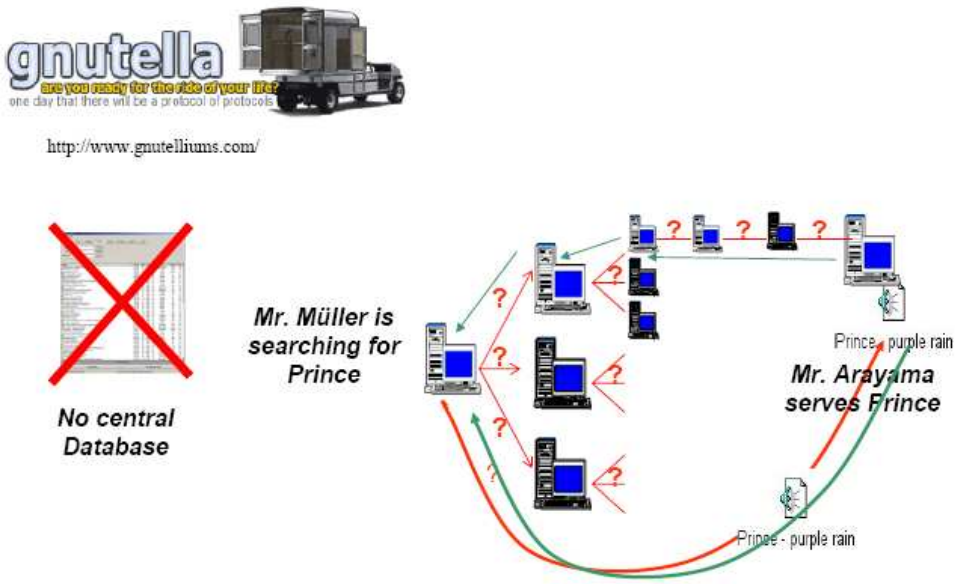


FIGURE 2-7 GNUTELLA ARCHITECTURE [WEHR06]

A requester types the name of song he is looking for. Then his machine knows of at least one other Gnutella machine somewhere on the network. The user's machine sends the song name he typed in to the Gnutella machine(s) it knows about. These machines search to see if the requested file is on the local hard disk. If so, they send back the file name (and machine IP address) to the requester. At the same time, all of these machines send out the same request to the machines they are connected to, and the process repeats. A request has a *TTL* (time to live) limit placed on it. A request might go out six or seven levels deep before it stops propagating. If each machine on the Gnutella network knows of just four others, that means that a request might reach 8,000 or so other machines on the Gnutella network if it propagates seven levels deep.

The principal advantage of pure P2P architecture like Gnutella is its being totally decentralized and each node acts as a servent, i.e. both a server and a client. But there are also some drawbacks with the protocol:

- **Bottlenecks due to different connection Speeds of Users.** Despite their different connection speed of single servents, users on the system act as gateways to other users to find the data they need. When the users on slower bandwidth

machines are acting as connections to those on higher bandwidth, it leads to bottlenecks.

- **The search may not discover the entire network**. It is possible that even though a certain resource is available on the network, it may not be visible to the requester if it is too many nodes away. By way of standard TTL of advertisement and search each client only holds connections to four other clients and a search/init packet is only forwarded five times, which means, only about 4,000 peers are reachable.

2.3.2.3 Third Generation: Mixed P2P

The next generation peer-to-peer applications such as BitTorrent and eDonkey combines the features of the first two generations peer-to-peer systems and tries to solve the problems appeared in the first two generations. By the hierarchical and structured architecture with the concept **super-peer** and **edge-peer**, mixed peer-to-peer architecture makes enhancement to improve the ability to deal with large number of users (see **FIGURE 2-8**). Super peers function as relays for edge peers and other super peers. Super peers are also used as the gateway to the network by edge peers and for NAT and firewall transversal.

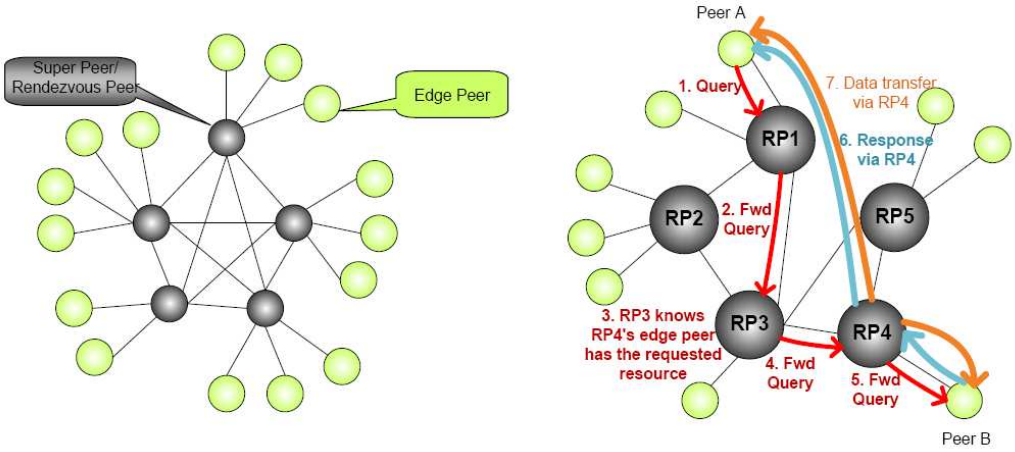


FIGURE 2-8 MIXED P2P AND OPERATING PRINCIPLE OF JXTA [HYAR04]

Open-source JXTA development framework belongs to the third generation peer to peer systems. Originally conceived by Sun Microsystems Inc. in 2001, the development of JXTA is still going on. By establishing a virtual network on top of the IP or non-IP networks, JXTA hides the underlying protocols. As in **FIGURE 2-8**, each Peer is given a unique, network independent logical ID when it joins JXTA platform, finds the closest rendezvous peer and use it as a gateway to peer-to-peer 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. 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.

In **FIGURE 2-8** a peer (e.g. peer A) is requesting for some resource, which is in this case in another peer located in the network behind NAT. When querying a resource, peer A sends a query to its rendezvous peer (e.g. RP1) (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 on 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 peer B (5.). Because B is in a network using NAT, it sends the response to peer A via its relay (and rendezvous) peer RP4 (6.). Then, the data is transferred between peer A and B using RP4 as a relay (7.) [HYAR04].

2.3.2.4 P2P Systems for 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 is no any well known third-generation protocol designed especially for mobile devices.

The principle of third generation architectures as such is suitable for mobile use. Despite the advantages of using third generation protocols in a mobile environment, 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 Java Micro Edition). It works in all MIDP devices.

JXME has two versions: proxyless and proxy-based. JXME proxy-based version uses a proxy and a relay that forward the messages coming from the JXTA network to the mobile device and vice versa. In addition to that, they filter out any unessential messages to save the mobile peer's network and processing resources. Combining small size and a rich set of features seems an impossible scenario for any one P2P platform, so one with many protocols and a possibility for relaying more constrained peer's messages with a more powerful one would look viable. Therefore proxy-based version seems promising. However, communication is not possible in the absence of knowledge where a proxy might exist in the network topology. For more powerful mobile devices the proxyless version is encouraged to be further developed. In the following **Section 2.4** Project JXTA/JXME will be discussed in very detail.

2.4 Project JXTA/JXME

Project JXTA started as a research project incubated at Sun Microsystems. Its goal is to explore a vision of distributed network computing using peer-to-peer topology, and to develop basic building blocks and services that would enable innovative

applications for peer groups. JXTA is short for Juxtapose, which means side by side. It is a recognition that peer to peer is juxtapose to client server or Web based computing — what is considered today's traditional computing model. The juxtaposed entities are any connected computing device on the network to communicate, including PC workstations and servers, cell phones and PDAs and groups of computers. Project JXTA is developed to make it easier to establish temporary associations between systems and groups and conform to three core principles:

- to use familiar technology and standards where possible,
- to seek the input of industry experts,
- to encourage open development.

JXTA works on the basis of protocols instead of API. Therefore JXTA is independent of operating systems, hardware or language. Some basic functions of peer-to-peer networking are creating, finding, joining, leaving and monitoring groups, talking to other groups and peers, and sharing content and services. The functions are performed by exchanging XML advertisements and messages between peers. Furthermore, JXTA enables interoperable P2P applications with a wide range of capabilities, including:

- establishing peer groups among users of various devices that can communicate easily across firewalls;
- the ability to find peers on the networks – even across firewalls;
- simplified file sharing;
- automatically detecting new Web site content;
- remote monitoring of peer activities;
- accessing deep Web data
- providing secure communication[JXTA04b].

2.4.1 JXTA Architecture

In **FIGURE 2-9** we have an overview about JXTA architecture. The simple structure indicates the design concept behind it: the whole structure should be as thin as possible so as to be provided as much interoperability as possible to encourage the innovation from developers and contributors. The JXTA software architecture consists of three layers:

Core Layer:

- Security

In order to support different levels of resource access, JXTA peers operate in a role-based trust model, in which an individual peer acts under the authority granted to it by another trusted peer to perform a particular task.

- Peer groups, Peer pipes, Peer monitoring

JXTA core is also called platform layer and encapsulates the minimal and essential primitives that are common to P2P networking. The functions for the peer group contain mechanisms for joining peers to groups or removing from them. The functions for the peer pipe contain communication mechanisms among other peers for transferring data, content and code independent of protocol. The functions for the peer monitoring contain mechanisms to control behaviour and activities for each peer [JXTA05].

Services Layer
The services layer includes network services that may not be absolutely necessary for a P2P network to operate, but are common or desirable in the P2P environment. By using the primitive functions from core layer, this layer provides higher-level functionality e.g. searching and indexing, directory, storage systems, file sharing, distributed file systems, resource aggregation and renting, protocol translation, authentication, and PKI (Public Key Infrastructure) services. For the specific purpose the developer could also create his own services by implementing the abstract component.

Applications Layer
The applications layer includes implementation of integrated applications, such as P2P instant messaging, document and resource sharing, entertainment content management and delivery, P2P Email systems, distributed auction systems, and many others [JXTA05]. In this layer, both JXTA service and JXTA core layer components are used to implement P2P applications. Since the entire system is modular-like, the developer can create variety of applications by implementing the abstract component for his need.

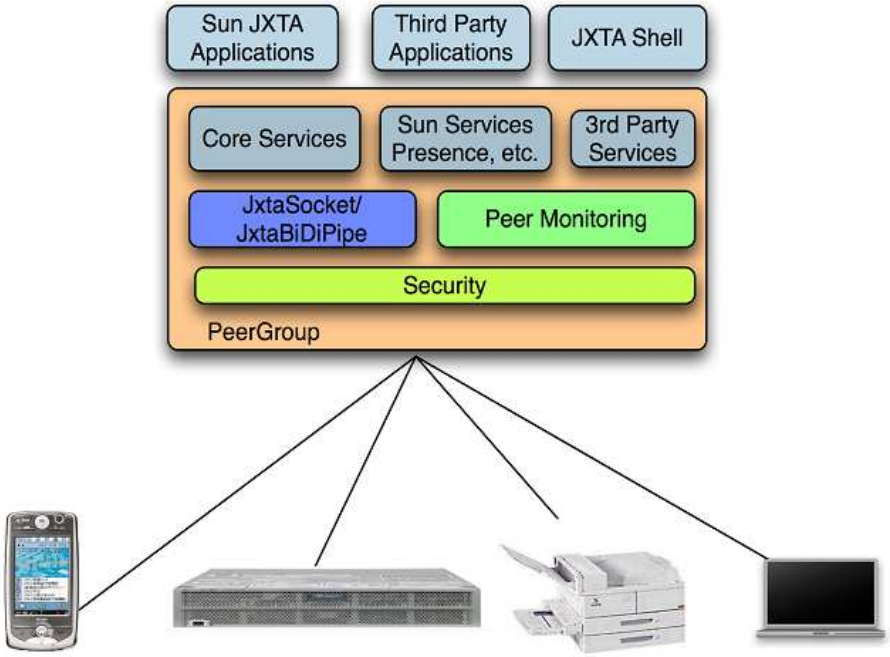


FIGURE 2-9 JXTA ARCHITECTURE [JXTA07]

2.4.2 JXTA Concept

JXTA technology is a set of simple, open peer-to-peer protocols that enable any device on the network to communicate, collaborate, and share resources. JXTA peers create a virtual, ad hoc network on top of existing networks, hiding their underlying complexity. In the JXTA virtual network, any peer can interact with other peers,

regardless of location, type of device, or operating environment — even when some peers and resources are located behind firewalls or are on different network transports. Thus, access to the resources of the network is not limited by platform incompatibilities or the constraints of hierarchical client-server architecture. JXTA technology espouses the core technology objectives of ubiquity, platform independence, interoperability, and security. JXTA technology runs on any device, including cell phones, PDAs, two-way pagers, electronic sensors, desktop computers, and servers. Based on proven technologies and standards such as HTTP, TCP/IP and XML, JXTA technology is not dependent on any particular programming language, networking platform, or system platform and can work with any combination of these.

2.4.2.1 Peers and Peer Groups

Any device with a digital heartbeat that implements one or more of the JXTA protocols could be a peer. Each peer is uniquely identified by a Peer ID and operates independently and asynchronously from other peers. In order to set up point-to-point connections, the network interface is published by the peer as the peer Endpoint. However, the direct connection between two peers is not obligatory and intermediary peers serve the purpose to route messages for peers which are separated due to physical network connection or network configuration.

Each peer belongs to Net Peer Group by default. Peers could self-organize into peer groups. The motivations to establish a peer group are e.g. to create a secure encodings and/or monitoring environment. The peers within a peer group agree upon a common set of services. Not all services within a peer group must be implemented by each peer. A peer just needs to implement services which are useful for him and use the implementation of the uncritical services provided by the default net peer group. JXTA provides a core set of peer group services such as:

- **Membership Service**, which is used by the peer in the current peer group to accept or reject the application from a new group.
- **Access Service**, which is for one peer to validate requests from another peer within a peer group.
- **Discovery Service**, which serves peer members to search for peer group resources.
- **Pipe Service**, which is used to create and manage pipe connections between the peer group members [JXTA04].

2.4.2.2 IDs

Peers, peer groups, pipes and other JXTA resources need to be uniquely identifiable. Unique IDs are generated randomly by the JXTA J2SE platform binding. A JXTA ID uniquely identifies an entity and serves as a canonical way of referring to that entity. Currently, there are six types of JXTA entities which have JXTA ID types defined: peers, peer group, pipes, contents, module classes, and module specifications. URNs are used to express JXTA IDs. URNs are a form of URI that are intended to serve as

persistent, location- independent, resource identifiers. Like other forms of URI, JXTA IDs are presented as text.

An example JXTA peer ID is:
`urn:jxta:uuid-59616261646162614A78746150325033F3BC76FF13C2414CBC0AB663666DA53903`

An example JXTA pipe ID is:
`urn:jxta:uuid-59616261646162614E504720503250338E3E786229EA460DADC1A176B69B731504`

2.4.2.3 Pipes

Pipes are virtual communication channels that are used to send messages by JXTA peers. They support the transfer of any object such as binary code, data strings, and Java based objects. The pipe endpoints are referred to as the input pipe and the output pipe. Two basic modes of communication, point to point and propagate, are offered by JXTA pipes as in **FIGURE 2-10**.

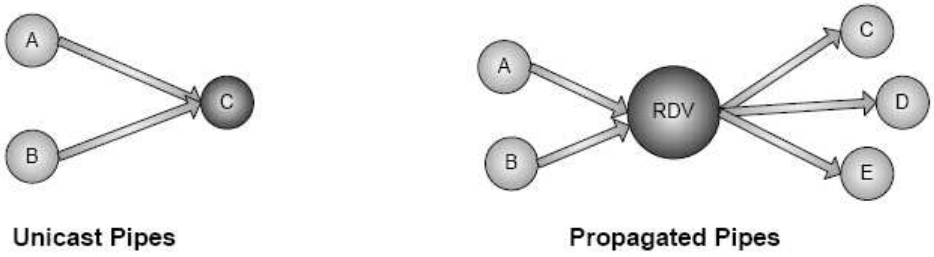


FIGURE 2-10 JXTA PIPES [JXTA07]

2.4.2.4 Modules

JXTA Modules are an abstraction used to represent any piece of “code” used to implement behaviour. The module does not specify what the “code” is and leaves the implementation of the behaviour to module implementations. With the generic abstraction form modules allow a peer to instantiate a new behaviour independent of platform or languages. The ability to describe and publish platform-independent behaviour is essential to support peer groups which are composed of heterogeneous peer. The module abstraction includes a **module class**, **module specification**, and **module implementation** (see **FIGURE 2-11**) [JXTA04a]:

- **Module Class**

The module class is primarily used to advertise the existence of a behaviour. The class definition represents an expected behaviour and an expected binding to support the module. Each module class is identified by a unique ID, the **ModuleClassID**.

- **Module Specification**

The module specification is primarily used to access a module. It contains all the information necessary to access or invoke the module. For instance, in the case of a service, the module specification may contain a pipe advertisement to be used to communicate with the service. A module specification is one approach to provide the functionality that a module class implies. There can be multiple module specifications for a given module class. Each module specification is identified by a unique ID, the **ModuleSpecID**. The ModuleSpecID contains the Module Class ID, indicating the associated module class. A module specification implies network compatibility. All implementations of a given module specification must use the same protocols and are compatible, although they may be written in a different language.

- **Module Implementation**

The module implementation is the implementation of a given module specification. There may be multiple module implementations for a given module specification. Each module implementation contains the **ModuleSpecID** of the associated specification it implements. Modules are used by peer group services, and can also be used by stand-alone services. JXTA services can use the module abstraction to identify the existence of the service (its **Module Class**), the specification of the service (its **Module Specification**), or an implementation of the service (a **Module Implementation**). Each of these components has an associated advertisement, which can be published and discovered by other JXTA peers.

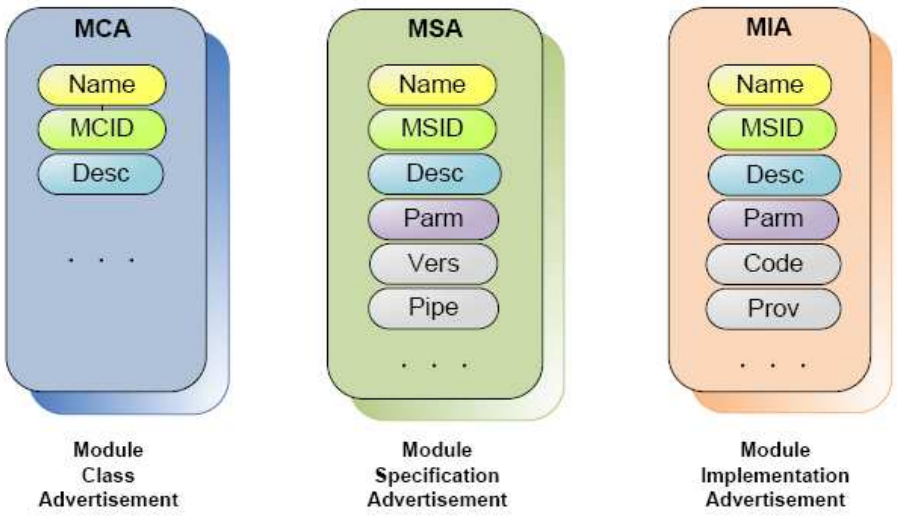


FIGURE 2-11 JXTA MODULE

2.4.2.5 Advertisement

Advertisements are language-neutral metadata structures resource descriptors in form of XML documents to represent network resources in JXTA network. With advertisements, the problem of finding peers and all their different types of resources can be reduced to a problem of finding advertisements describing those resources.

As the basic unit of data exchanged between peers to provide information on available services, peers, peer groups, pipes, and endpoints, advertisements can be used to virtually describe anything: source code, script, binary, classes, compiled JIT code, Java objects, EJB, J2EE containers. Project JXTA standardizes advertisements for the following core JXTA resource: peer, peer group, pipe, service, metering, route, content, rendezvous-peer, endpoint, transport.

On the basis of these advertisements developers can add unlimited amount of additional information for their own purpose. By way of related advertisements peers cache, publish and exchange advertisements to discover and find available network resources.

2.4.3 JXTA Protocols

JXTA defines a series of XML message formats, or ***protocols***, for communication between peers. Peers use these protocols to discover each other, advertise and discover network resources, and communication and route messages.

As in **FIGURE 2-12** there are six JXTA protocols [JXTA04]:

- **Peer Discovery Protocol** (PDP) is used by peers to advertise their own resources and discover resources from other peers. Each peer resource is described and published using an advertisement. PDP is the default discovery protocol for all user defined peer groups and the default net peer group. The current Project JXTA J2SE platform binding uses a combination of IP multicast to the local subnet and the use of rendezvous peers, a technique based on network- crawling.

- **Peer Information Protocol** *(*PIP) is used by peers to obtain status information such as uptime, state, recent traffic, etc. from other peers once the location of a peers are clear. This information is useful for commercial or internal deployment of JXTA applications. The PIP ping message is sent to a peer to check if the peer is alive and to get information about the peer. The ping message specifies whether a full response (peer advertisement) or a simple acknowledgment (alive and uptime) should be returned.

- **Peer Resolver Protocol** (PRP) enables peers to send a generic query to one or more peers and receive a response or multiple responses to the query. Queries can be directed to all peers in a peer group or to specific peers within the group. Unlike PDP and PIP, which are used to query specific pre- defined information, this protocol allows peer services to define and exchange any arbitrary information they need.

- **Pipe Binding Protocol** (PBP) is used by peers to establish a virtual communication channel, or **pipe**, between one or more peers and to bind two or more ends of the connection.

- **Endpoint Routing Protocol** (ERP) is used by peers to find routes to destination ports on other peers. Route information includes an ordered sequence of relay peer IDs that can be used to send a message to the destination.

- **Rendezvous Protocol** (RVP) is a mechanism by which peers can subscribe or be a subscriber to a propagation service. Within a peer group, peers can be rendezvous peers or peers that are listening to rendezvous peers. RVP allows a peer to send messages to all listening instances of the service. The RVP is used by the Peer Resolver Protocol and the Pipe Binding Protocol to propagate messages.

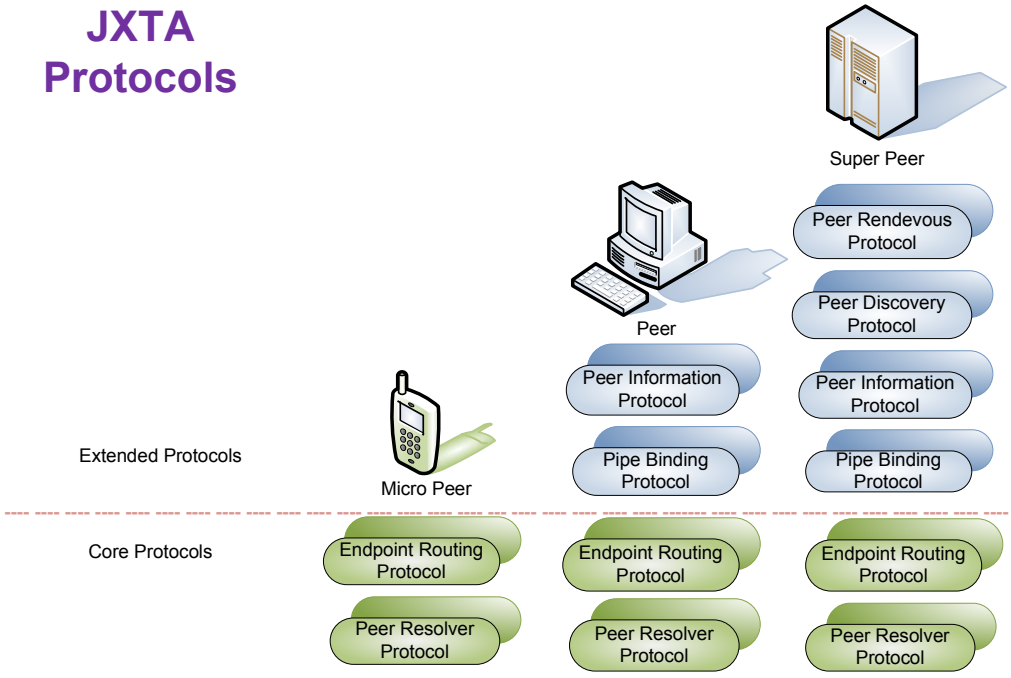


FIGURE 2-12 JXTA PROTOCOLS [MANN04]

2.4.4 JXTA Peer Category

According to the function and feature the peers in JXTA network are divided into following four categories:

- **Minimal edge peer.** A minimal edge peer can send and receive messages, but does not cache advertisements or route messages for other peers. Peers on devices with limited resources (e.g., a PDA or cell phone) would likely be minimal edge peers.

- **Full-featured edge peer.** A full-featured peer can send and receive messages, and will typically cache advertisements. A simple peer replies to discovery requests with information found in its cached advertisements, but does not forward any discovery requests. Most peers are likely to be edge peers.

- **Rendezvous peer.** A rendezvous peer is like any other peer, and maintains a cache of advertisements. However, rendezvous peers also forward discovery requests to help other peers discover resources. When a peer joins a peer group, it automatically seeks a rendezvous peer. If no rendezvous peer is found, it dynamically becomes a rendezvous peer for that peer group. Each rendezvous peer maintains a list of other known rendezvous peers and also the peers that are using it as a rendezvous. Each peer group maintains its own set of rendezvous peers, and may have as many rendezvous peers as needed. Only rendezvous peers that are a member of a peer group will see peer group specific search requests. Edge peers send search and discovery requests to rendezvous peers, which in turn forward requests they cannot answer to other known rendezvous peers. The discovery process continues until one peer has the answer or the request dies. Messages have a default time-to-live (TTL) of seven hops. Loopbacks are prevented by maintaining the list of peers along the message path [JXTA04].

- **Relay peer.** A relay peer maintains information about the routes to other peers and routes messages to peers. A peer first looks in its local cache for route information. If it isn't found, the peer sends queries to relay peers asking for route information. Relay peers also forward messages on the behalf of peers that cannot directly address another peer (e.g., NAT environments), bridging different physical and/or logical networks Any peer can implement the services required to be a relay or rendezvous peer. The relay and rendezvous services can be implemented as a pair on the same peer.

2.5 Related Projects

2.5.1 Related Projects about P2P-based Web Service discovery

Two related projects are introduced to have some idea about the research in P2P-based Web Service discovery. It is interesting that project WS-Talk integrates Web Services and semantic Web standards in JXTA P2P environment. And the project in **Section 2.5.1.2** intends to develop a service discovery system for ubiquitous peer-to-peer networks which is (programming) language and network independent and flexible as well.

2.5.1.1 WS-Talk

The WS-Talk project consortium consists of both industrial and academic initiatives. The Web Service (WS)-Talk interface Layer is a structured natural language interface for the inter-service communication that extends service virtualisation to strengthen consumer self-service. While providers will concentrate more on the technical levels of activation and communication within a service network, the users, i.e. the service consumers, will form ad-hoc collaborations between services at the semantic level that suit their own specific needs. Web Service (WS)-Talk layer is presented as a structured-language interface for Web services. This "open building block" can be implemented by both the service designers who as providers are more concerned with

the architecture of the underlying service model and the service consumers who as users will seek to specify Web services as solutions to specific problems [CZM05].

What mainly interests us in WS-Talk project is the integration of Web Services and semantic Web standards in JXTA peer-to-peer environment. Several reasons are mentioned for the use of JXTA technology:

- Their previous work for Web Service and JXTA coexistence and cooperation clearly shows how the JXTA native-mode XML description can function seamlessly with SOAP and WSDL as used by Web Services.

- Search and resource discovery functionality for Web Services is supported, through self-description of the Web Services in XML, and by implication of SOAP and WSDL.

- Through XML self-description, free text – hence natural language – querying is supported.

- JXTA is based on open source, Java code, and has a large developer community.

- In line with some other peer-to-peer systems JXTA supports and favours stable ontology. (Cf. P2P systems for music delivery and the ontology used in such areas.)

- JXTA supports decentralized peers and thus is flexible and robust [CZM05].

The objective is to merge (i) Web Services for description, (ii) semantic web for search and support of ontology, and through ontology, natural language, and (iii) peer-to-peer networking because it provides a convenient transport layer. Therefore the protocols used will be WSDL for Web Service description; DAML which has now become OWL for ontology description; SOAP, for access and transfer; all based on JXTA using pipes or sockets as a transport layer.

Since they are looking for a way to combine different workflows, **Business Process Execution Language** (**BPEL**), which is used to model the behaviour of both executable and abstract process, becomes a natural choice. BPEL for Web services is an XML-based language designed to enable task-sharing for a distributed computing or grid computing environment – even across multiple organizations – using a combination of Web services [BPEL05]. And BPEL serves the need of the project well to support the “open building block”. However, with the progress of their research, it is not sure whether BPEL without P2P works better (or worse). One reason is that it is found that the searching capabilities of JXTA did not well suit their purposes because of the asynchronous way in which the queries were replied (i.e. two equal queries give as response two different answer). What is sure is that a pure P2P is not needed. And the further research and implementation in respect to P2P ceased to be executed then.

2.5.1.2 Service Discovery in Peer-to-Peer Networks

The goal of this work is to develop a service discovery system for ubiquitous peer-to-peer networks that is:

- Programming language independent. It should be possible to write peers in different languages.

- Network independent. No assumptions should be made about network topologies or protocols, since these factors are likely to be highly variable in real life ubicomp situations.

- Flexible. It should be possible to use any sort of device or service with our system.

The approach deployed in the research allows platform-independent interaction between devices represented by JXTA peers, and the ability to describe services with rich semantic information to enable advanced service discovery. It shows how the JXTA peer-to-peer framework can be enhanced with semantic service descriptions written in DAML-S and WSDL (see **FIGURE 2-13**). This allows any kind of device to be represented as a JXTA peer. Furthermore, Java support classes are created for reasoning about these DAML-S descriptions using JTP [Elen03]. It is also shown how platform-independent communication between JXTA peers (and thus devices represented as JXTA peers) can be done, using SOAP over JXTA. By writing a number of sample devices and providing a simple graphical user interface to their service discovery framework. What this amounts to is a service discovery system for ubiquitous peer-to-peer networks that is platform independent and highly flexible.

JXTA has proved to be very flexible due to its use of open XML messages and advertisements, and non-commitment to any particular standard for remote communication. This made it easy to integrate other technologies with it (namely DAML-S, WSDL, SOAP, and JTP). Also, the combination of technologies and languages which is chosen to use, Java, DAML-S, WSDL, SOAP, JTP works very well. Reasoning using an inference engine and semantic device descriptions allows powerful service discovery. However, the whole web services concept is rather heavy-weight. The implementation requires *jar* files weighing in at over 12 MB in all, and most of these are due to using SOAP and WSDL. Even if some of this could be reduced, (optimized) using text-based XML messages will always incur an overhead as compared to binary messages [Elen03].

The related research about P2P-based Web Services discovery is discussed. In the next section **2.5.2**, some research of scalability test of JXTA protocol is to be reviewed so as to get some reference and determine the most appropriate type(s) for scalability test in this thesis.

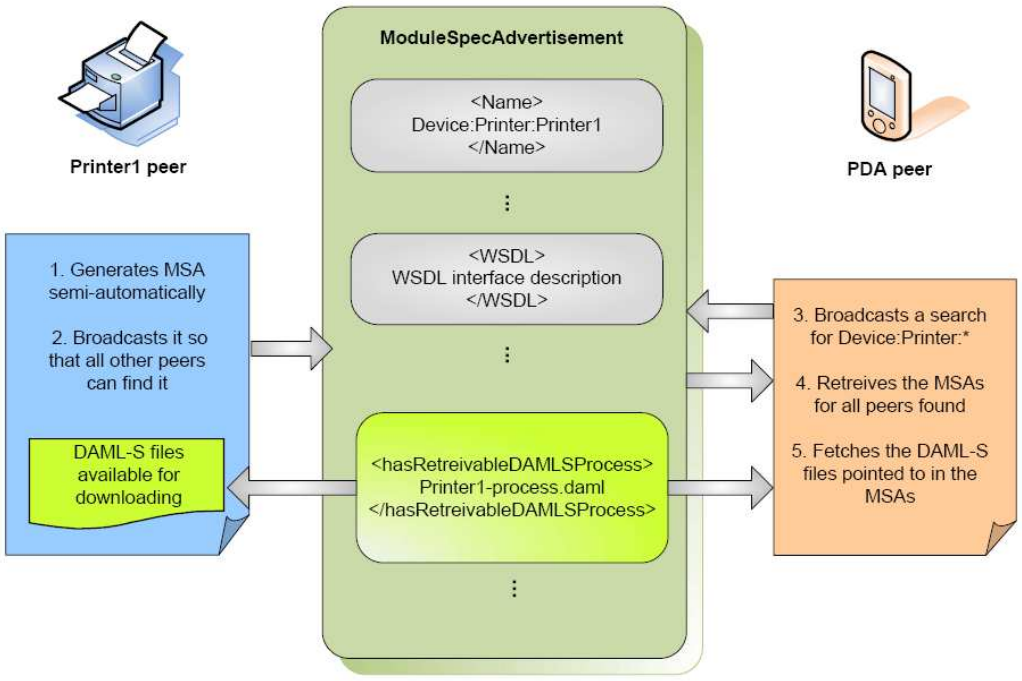


FIGURE 2-13 HOW TO USE MODULE SPECIFICATION ADVERTISEMENT [ELEN03]

2.5.2 Related Projects about JXTA Framework Scalability Test

Three projects regarding the performance research of JXTA framework are discussed in this section. The broadest research to measure the scalability capability of JXTA is conducted by the JXTA Benchmarking project of JXTA developer community [DoYP02]. The goal of the project is to construct a test harness which can be set up by people relatively unfamiliar with JXTA, operated more or less automatically (scripted), which reports on network performance measures of interest to the JXTA community. The second reference project [HaDe03] intends to complement the JXTA Benchmarking project by focusing on the performance evaluation of typical peer operations and consequences for the peer network, the user and the developer. It covers the trade-off between peer startup latency and the maintenance of the local cache, the throughput limits of pipes, the measurement of the core JXTA communication concept in a LAN environment. The third project is initiated by trying to answer a common and unanswered question on the JXTA mailing lists: How many rendezvous peers are supported by JXTA in a given group? With the focus on a de facto standard of P2P programming – JXTA specifications, a detailed, large-scale, multi-site experiment is conducted by using the nine clusters of the French Grid'5000 test bed [CCDD05]. Because our intention is to build a suitable benchmark suit for scalability test of P2P based mobile Web Service discovery, we are interested in the effect which is brought both by peer operations and by the scale of the network. The study of these two projects gives us a rational reference on these aspects.

2.5.2.1 JXTA Benchmarking Project

JXTA Benchmarking project is conducted at Sun Microsystem, Inc's Wireless Engineering Laboratory. Two types of testing: vertical and horizontal, are performed. Vertical test measures performance of JXTA core and JXTA services on a given standalone peer. Horizontal changes are major fundamental changes in design and topology to get broad performance and scalability improvements of at least an order of magnitude as defined in the scalability design under platform. In the order of priorities JXTA Benchmarking efforts can be broadly classified into the following categories:

1. Discovery & Resolver
2. Load/Stress Tests
3. Round trip Timings for messages
4. Single peer stand alone performance test
5. A real life application

Among the categories we could see discovery is the core functionality of any p2p solutions and no exception for JXTA. It stands with the highest priority in the row because the performance and scalability of discovery services determines the performance and scalability of JXTA platform to a large extent. Since discovery process mainly consists of parsing the local cache of advertisements, checking the indices, propagation of discovery messages/query and resolving responses, the project proposes the following scenarios to conduct discovery test as in **FIGURE 2-14**:

- **Single Peer:**

  P1<->P1

  On a given peer, it measures the time it takes to discover an advertisement. The test is repeated by varying the number of advertisements in its local cache.

- **Two Peers on same subnet:**

  P1<->P2

  On a given peer, it measures the time it takes to discover an advertisement by propagating queries to other peers in a subnet. The test needs to be repeated by varying number of advertisements in a subnet. Queries are generated from P1 and advertisements are to be found on P2.

- **Discovery through one Rendezvous:**

  P1<->R1

  On a given peer, it measures the time it takes to discover an advertisement by propagating queries to a rendezvous peer. The number of advertisements is also needs to be varied on R1.

- **Discovery through chain of two or more Rendezvous:**

  P1<->R1<->R2…

In this scenario, P1 generates queries to R1. But the advertisements to be found are not cached on R1. Instead, they are cached on other rendezvous, R2 or R3. So queries are further propagated amongst Rendezvous.

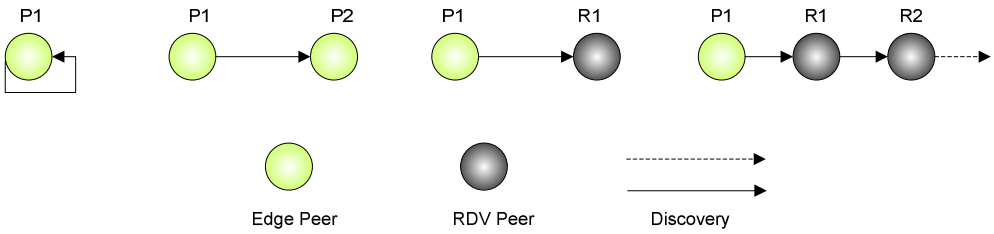


FIGURE 2-14 DISCOVERY SCENARIOS

The Benchmarking project aims to define the JXTA benchmarks and collect performance and scalability measurements as the platform development progresses. The next project [HaDe03] we will present intends to extend the existing work of JXTA Benchmarking project by studying typical peer operations and consequences.

2.5.2.2 The Cost of Using JXTA

The proposed performance model of JXTA in [HaDe03] consists of the following components and metrics:

1. Latency of typical peer operations
2. message round-trip time (RTT)
3. message and data throughput
4. Rendezvous query throughput
5. Relay message throughput

For our interest the first two components in the performance model are most related to our discovery scalability test. Typical peer operations shown in **FIGURE 2-15** are based on the review of two JXTA applications: a P2P forum system[5] and MyJXTA [JuWR99].

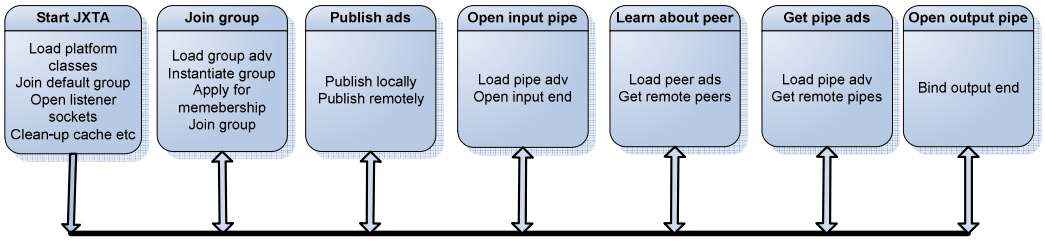


FIGURE 2-15 TYPICAL JXTA PEER OPERATIONS [HADE03]

[5] http://www.gnutella.com/.

Most of the high-level JXTA peer operations are actually sets of distinct basic steps, as shown in **FIGURE 2-15**. From the performance perspective, the developers and users should be interested in the cost of these operations due to the following reasons: high cost of the basic steps involved to complete an operation, frequency of steps or operations performed during the peer's lifetime in the network. To simplify the evaluation, only the costs of the typical high-level operations are measured in [HaDe03], and the typical high-level operations in the relative order a peer performs them upon startup include:

- **Start the JXTA platform**, to initialize the environment for running JXTA protocols and services.
- **Join a peer group**, to enjoy a more secure and efficient environment according to user preferences or common peer services,.
- **Publish own advertisements**, to make peers aware of the presence and available resources.
- **Open an input pipe**, to receive messages from peers.
- **Learn about other peers,** who participate in the same group and share common resources.
- **Obtain pipe advertisements**, to discover available communication channels of other peers.
- **Open output pipe,** to send messages to other peer(s).

Two types of benchmarks are set and studied while analyzing peer's typical operations. They are **startup benchmark** and **Round-trip time benchmark**. Since these two types of benchmarks provide a suitable way to describe the different performances of discovery process with or without categorization mechanism, they will be taken into our benchmark suit in the scalability test of this thesis

2.5.2.3 Performance Scalability of the JXTA P2P Framework

In another study [ACDJ06] JXTA peerview and discovery protocol is evaluated by detailed, large-scale, multi-site experiments using the nine clusters of the French Grid'5000 test bed [CCDD05] with various JXTA-C overlay configurations. The peerview protocol is used by RDV peers to organize them by synchronizing their views of each other. The goal of discovery protocol is to find resources within the group.

With the number of test RDV peers up to 200, the test of discovery protocol shows that adding up to 50 RDV peers does not significantly increase the discovery time. From 50 to 200 RDV peers, the discovery time grows linearly. The result is explained by the result of peerview protocol test. The property for stabilization of the peerview can not be satisfied when a large number of RDV peers are used, which causes a linear cost with respect to the number of RDV peers. The impact of "noise" on the discovery time is also studied by a total of 5000 fake advertisements published by 50 edge noisiers, i.e. 50 edge peers. The maximum overhead is measured when the number of RDV peers *r* is 5. Then for values of *r* up to 150, the overhead slightly decreases. For values

of r between 150 and 200, publishing fake advertisements does not have any more effect on the discovery time.

The available test environment of our project is limited to a small number of RDV peers. And the performance analysis of [ACDJ06] provides us a practical and valuable reference about the scalability JXTA discovery protocol. Together with the other two projects reviewed in the **Section 2.5.2.1** and **2.5.2.2** it sets the basis for the design and build-up of benchmark suit of scalability test in this thesis.

2.6 Previous Work

The previous work in our project is reviewed in this section. At first it is about the feasibility of mobile Web Service provisioning from mobile Host. Then a model is designed to merge Web services and P2P technology on mobile and other resource constrained devices.

2.6.1 Mobile Web Service Provisioning

The introduction of Third and Interim Generation mobile communication technologies in the cellular domain like UMTS, GPRS/EDGE increased the speed of wireless data transmission significantly. The drastic increase of processing power and device capabilities enables better applications and usage of mobile devices in different application domains. From the aspect of technical development, time is mature to enable mobile devices to act not only as Web Services requestor but also as Web Services provider. In the previous research in Mobile Web Services Provisioning Project, feasibility of mobile Web Service provider for the Smart Phones is analysed.

The basic architecture of the mobile terminal as Web Service provider can be established with the Web Service provider (”Mobile Host”) being implemented on the smart phone. 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 tunnelling are diverted and handled by the Web Service handler. Even though the Web Service provider is implemented on the Smart Phone, the standard WSDL can be used to describe the services, and the standard UDDI registry can be used for publishing and un-publishing the services [SrJP06a].

From the introduction of Web Services in **Section 2.2**, we know the basic architecture for Web Services is built upon Service Requestor (Client), Service Provider and Service Registry. The basic structure of mobile Web Services host is similar (see **FIGURE 2-16**). The Mobile service provider publishes its Web Services with the service registry. The service requestor later searches (”Find”) the UDDI registry for the services, and the UDDI compatible service registry refers the respective WSDL for the service. The service requestor accesses the described Web Service, using SOAP.

In the research, the prototype of Mobile Host was set up and it was extensively tested considering performance aspects in different real-time working environments/conditions. The evaluation clearly 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 [SrJP06b].

FIGURE 2-16 BASIC ARCHITECTURAL SETUP OF MOBILE HOST [SRJP06A]

2.6.2 Mobile Web Services Discovery in JXTA/JXME

As introduced in **Section 2.3**, the mobile Web Services discovery paradigm by way of centralized UDDI registry has many drawbacks and does not really suit the need of Web Services discovery process in mobile environment. 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.

This stage of research intended to find a better solution for the Web Services discovery on mobile and normal hosts. It provided 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 requestors via P2P technology are presented. The aim of this network is to develop a distributed service discovery mechanism. JXTA's P2P provides perfect solution for service discovery and communication among mobile users as "peers".

By mapping JXTA modules to WSDL and UDDI illustrated in **FIGURE 2-17**, WSDL can be represented with three modules of JXTA, which are Module Class, Module Specification and Module Implementation, and advertisement of these modules represents UDDI behaviour.

To invocate the Web Services, JXTA-SOAP model, Proxy model and Port forwarding model are studied. After comparison Port forwarding model is chosen to be the most appropriate one to suit the research need. More about this Web Services invocation will be introduced in **Section 3.3**.

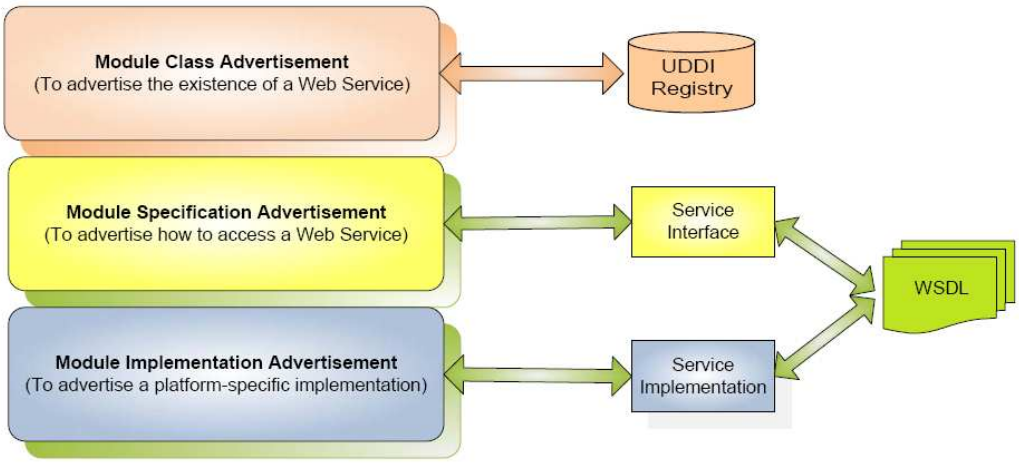


FIGURE 2-17 TO MAP JXTA MODULES WITH WEB SERVICES [SRJP06A]

2.7 Summary

SOA is more a wave of future than just a hot topic. With SOA IT becomes more agile in responding to the changing business needs. The features such as reusability, interoperability, scalability and flexibility of web services make it the most prevalent approach to implementing an SOA. P2P system is one approach for mobile Web Service discovery and has evolved through three generations: Hybrid P2P, pure P2P and mixed P2P. The third generation mixed P2P is nowadays used for mobile environment. JXTA technology could be used to create P2P based applications based on Java technology. It is designed with a simple structure so as to be provided as much interoperability as possible to encourage the innovation from developers. In the previous work of our project the feasibility of mobile Web Service provisioning is proved and in the search for a distributed service discovery mechanism. JXTA provides superb solution for service discovery and communication among mobile users as “peers”.

3 Conceptual Design

3.1 Mobile Web Services Discovery

While introducing web services operations in **Section 2.2.1.2** on the first stage we take it for granted that the requester entity and provider entity have known each other. In reality it is usually not the case. It often happens that a requester entity needs to initiate a service but has no idea about the provider agent it wishes to engage, a requester entity may then need to discover a candidate. Discovery is "the act of locating a machine-processable description of a Web service that may have been previously unknown and that meets certain functional criteria" [BHMN04]. A discovery service is a service that facilitates the process of performing discovery. It is a logical role, and could be performed by either the requester agent, the provider agent or some other agent.

While networked applications follow the static install model, mobile applications follow the discover, lease, deploy and discard model. To deploy application is the purpose, but to discover applications which meet certain functional criteria is the first and a very crucial stage of the whole process. Without effective discovery mechanism every single stage after discovery would be impossible to implement. Without efficient discovery mechanism the efficiency of the whole process is in question.

Web Services Discovery takes place in the first stage of the whole web services process, when it is necessary. Discovery is the prerequisite for a functional web service process and is the key focus of our project research at present. In **Section 2.2.1.2** all the four stages are illustrated. Now the first stage of web services discovery will be studied in detail. We will do further analyse at first about the process of discovery service in **Section 3.1.1** and then conceive discovery service from different viewpoints in **Section 3.1.2** so as to find the suitable discovery approach for the mobile web service discovery research in our project.

3.1.1 Web Services Discovery Process

Let's read the **FIGURE 3-1** about the web service process again but focus just on the first stage to see how a requester entity and a provider entity could know each other.

a. The discovery service somehow obtains both the Web service description ("WSD" in **FIGURE 3-1**) and an associated functional description ("FD") of the service. The functional description is a machine-processable description of the functionality (or partial semantics) of the service that the provider entity is offering. It could be as simple as a few words of meta data or a URI, or it may be more complex, such as a Tmodel in UDDI or a collection of RDF, DAML-S or OWL-S statements.

This architecture does not specify how the discovery service obtains the service description or functional description. Various possible paradigms will be introduced in the following **Section 3.1.2**. If the discovery service is implemented as a registry such as UDDI, then the provider entity may need to actively publish the service description and functional description directly to the discovery service. If the discovery service is implemented as a search engine, then it might crawl the Web and collect service descriptions wherever it finds them, while the provider entity have no idea about it.

b. The requester entity supplies criteria to the discovery service to select a Web service description based on its associated functional description, capabilities and potentially other characteristics. One might locate a service having certain desired functionality or semantics; however, it may be possible to specify "non-functional" criteria related to the provider agent, such as the name of the provider entity, performance or reliability criteria, or criteria related to the provider entity, such as the provider entity's vendor rating [BHMN04].

c. The discovery service returns one or more Web service descriptions (or references to them) that meet the specified criteria. If multiple service descriptions are returned, the requester entity selects one, perhaps using additional criteria.

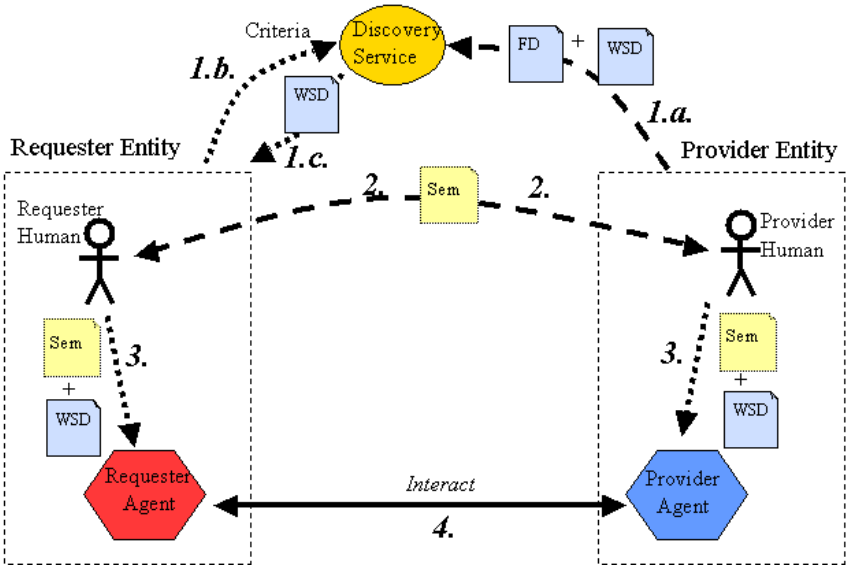


FIGURE 3-1 WEB SERVICES DISCOVERY PROCESS [BHMN04]

3.1.2 Web Services Discovery Viewpoints: Registry, Index, dynamic or Peer-to-Peer?

At present, there are three leading viewpoints on how a discovery service should be conceived: as a *registry*, as an *index*, or as a *peer-to-peer* system. By analyzing characteristics of each of them, we would like to find out which paradigm suits our need to the most extent.

3.1.3 The Registry Approach

A *registry* is an authoritative, centrally controlled store of information. Publishing a service description requires an active step by the provider entity: it must explicitly place the information into the registry before that information is available to others. The registry owner decides *who* has authority to place information into, or update, the registry and *what* information is placed in the registry. Others cannot independently

augment that information. UDDI is often seen as an example of the registry approach, but it can also be used as an index [BHMN04].

Centralized registries may be more appropriate in more static or controlled environments where information does not change frequently. And the usual mechanism to publish Web Services with a UDDI registry is far from an ideal alternative for mobile environment on which we are concentrating. The possible great amount of Web Services to publish by Mobile Hosts would easily lead to bottlenecks and become single points of failure. The dynamic nature of mobile network throws a second doubt on deploying UDDI registry for the mobile nodes which can join or leave at any time and switch from one operator to another. In order to keep the binding information up-to-date the Mobile Hosts have to publish Web Services every time when they change the operating network. The spontaneous moving nature of nodes results in bulk of out-of-date advertisements on one side, and a waste of registry resources on the other. Even so, to keep the published Web Services information very up to date is still not guaranteed and stays a hard nut to crack.

3.1.4 The Index Approach

In contrast with a registry, an *index* is a compilation or guide to information that exists elsewhere. It is not authorities and does not centrally control the information that it references. Publishing is passive: the provider entity exposes the service and functional descriptions on the Web, and those who are interested (the index owners) collect them without the provider entity's specific knowledge. Anyone can create their own index. When descriptions are exposed, they can be harvested using Web spiders[6] and arranged into an index. Multiple organizations may have such indexes [BHMN04]. The information contained in an index could be out of date. However, since the index contains pointers to the authoritative information, the information can be verified before use. An index could include third-party information. Different indexes could provide different kinds of information — some richer, some sparser. Free-market forces determine which index people will use to discover the information that they seek.

It is important to note that the key difference between the registry approach and the index approach is not merely the difference between a registry itself and an index in isolation. Indeed, UDDI could be used as a means to implement an individual index: just spider the Web, and put the results into a UDDI registry. Rather, the key difference is one of control: Who controls *what* and *how* service descriptions get discovered? In the registry model, it is the owner of the registry who controls this. In the index model, since anyone can create an index, market forces determine which indexes become popular. Hence, it is effectively the market that controls what and how service descriptions get discovered [BHMN04].

Perhaps the most well known type of discovery service used is the web search engine. Search engines, such as Google[7] and Altavista[8], create massive indices of the millions of web pages they crawl; allowing users to query for pages relevant to their

[6] A Web Spider is a program or automated script which browsers World Wide Web methodically. Spiders are mainly used to create a copy of all the visited pages for later processing by a search engine that will index the downloaded pages to provide fast searches. Here spiders are used to gather specific types of information from Web pages.
[7] www.google.com
[8] www.altavista.com

interests using pattern matching. Search engines have progressed from only cataloguing web pages to allowing searches on images, Usenet newsgroups, media files, and more. To assist in searching for information on specific topics, sites such as Yahoo![9] Provide directories of all web sites sorted into subject-related categories and sub-categories. This allows users to use the directory service like a Yellow Pages for services, or to narrow the subject scope of the index for their pattern match search. Though search engines and directory services work well with static web services and information, they are not suitable for searching for dynamic services. Because of the enormous amount of information, web search engines cannot maintain a fresh index of all websites [MaKr02]. Though the most popular sites may be crawled more often, the average website index may be months old [BaCH00] [LaGi99]. Directories with months-old stale information are not going to be suitable for locating web services with lifespan of minutes or hours.

By distributing the index of content and services across many different servers, the time and cost of maintaining the index in each server's scope is greatly reduced. Web services can register and update the directory servers handling their most specific categories. A large distributed directory service must determine where queries are sent and how to efficiently route them.

3.1.5 Dynamic Approach

The wish to enable Web Services discovery in an ad-hoc fashion arouse the extensive research around the topic Dynamic Service Discovery. Among the prominent service discovery mechanisms available, E-speak and UDDI (Universal Description, Discovery and Integration) are designed specifically for discovering web services, while Salutation, Jini, and UpnP (Universal Plug and Play) are geared toward services furnished by hardware devices (printers, faxes, etc.). All can be used to discover software services (web services) either by design or when provided with the appropriate wrappers. Some of them are explained here.

3.1.5.1 E-speak

E-speak, Hewlett-Packard's e-services initiative, is an open services platform. It is designed to let e-services and smart devices communicate with one another in a secure and manageable way regardless of their physical location, platform, system management policy, development environment or device capabilities.

Through a powerful query mechanism clients could discover service dynamically from E-speak engines and from external advertising depots. The query of clients is constructed with the attributes of services. A client inquires the E-speak engine of matching services. A query usually contains a constraint and may contains no preferences at all or more than one preferences. The constraint specifies a condition that services of interest much satisfy. If the E-speak engine finds services that match constraints, preferences are used by the engine to make an order of results. Three preference operators **min**, **max** and **with** are defined. The **min** takes an expression and orders services in ascending order of the value of the expression. On the opposite, the **max** orders services in descending order of the value of the expression. The **with** operator has an condition and a weight expression. If the condition is evaluated

[9] www.yahoo.com

to true, then the weight is added to the total weight of resource. If the condition is evaluated as false, then nothing is done. After all the evaluation is finished, the services are ordered ascending or descending according to their total weight. The number of results to be returned is decided by arbitration policy. Three operators are used: **first, all** and **any.** The **first** operator has an integer variable in the form of "first n" which indicates that the first n results should be returned. The **all** operator means that all the results are to be returned. And **any** operator enables to return any service which is found.

The following diagram explains the lookup process more clearly. From Repository services which match the constraint construct N2. If the user specifies preference(s), N2 will be ordered according to the preference(s) and construct N3. If no preference is given, N2 remains the same, i.e. N3 equals N2. If the user specifies arbitration policy, N3 will be filtered and construct N4. If no arbitration policy is specified, N3 remains the same, i.e. N4 equals N3. N4 indicates the services to be returned to the user [GKLS00].

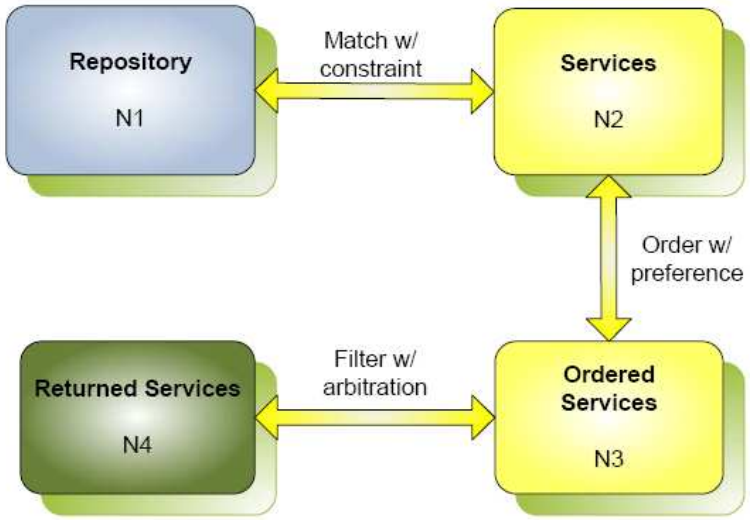


FIGURE 3-2 E-SPEAK LOOKUP PROCESS [GKLS00]

3.1.5.2 Salutation

Salutation is a service discovery and session management protocol developed by the Salutation Consortium. Salutation was created to solve the problems of service discovery and utilization among a broad set of appliances and equipment in an environment of widespread connectivity and mobility. The architecture also enables application, services and devices to search other applications, services or devices for a particular capability, and to request and establish interoperable sessions with them to utilize their capabilities.

The Salutation protocol aims to integrate different devices into a network by supplying them with a standard communications and API specification for discovering the capabilities of other entities in a network. As shown in **FIGURE 3-3**, this architecture is based on a model called the Salutation Manager (SLM), which is similar to the lookup service in Jini and functions as a service broker for services in the network. The

Salutation architecture enables transaction between function units representing essential features of a service (e.g. fax, print, scan etc). Each functional unit is composed of descriptive attribute record. Another important entity called transport manager that isolate SLM from particular transport protocol and provide SLM reliable communication channels independent transport layer. SLM may sit one more than one Transport Manager attaching to different network, and provide a transport-independent interface to Server and Client applications [YuAg03].

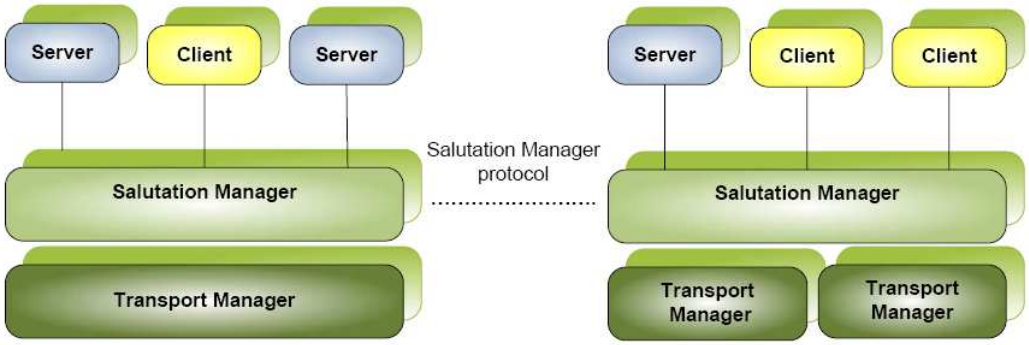


FIGURE 3-3 SALUTATION ARCHITECTURE [YUAG03]

3.1.5.3 Jini

Jini is a coordination framework by Sun with the goal to turn the network into a flexible tool for clients to find the resources they need. The clients refer both human users and computers. A service protocol, which is a set of interfaces written in Java, serves for the services to communicate with each other.

The discovery service of Jini proceeds in the following steps:

- A device (client) intends to look for a service and publish advertisements.
- A Lookup Service (server) runs instances of the discovery service which listen for multicast requests form discovering entities.
- The discovering entity performs a multicast that requests references to the lookup service.
- The lookup server calls a remote method on the discovering entity's exported object instance passing a remote reference to its lookup service as the parameter.

By extending Java application environment Jini enables code and data to move between machines. Autonomous devices who could be aware of the existence of each other and be able to coordinate to fulfil tasks if needed construct a federation. To realize the coordination lookup service is used to find and resolve services. Lookup Service keeps dynamic information about these autonomous devices and plays a key role in the framework. Every device must find one or more Lookup services before joining a federation. Lookup Services could also be assigned with group names. A device may join the group according to group name of Lookup services. After finding the lookup service which is of its interest, a device could register to provide lookup

service information about itself or query for information about other devices [Cals99]. So far all discovery mechanisms are based on the assumption that a registry by a service proxy is always available, which is not realistic for dynamic ad hoc networks. A discovery mechanism on the basis of centralized registry will not function well for Web Services distributed in ad hoc networks. The following two approaches VISR and Konark are designed to meet the needs of distributed Web Service discovery.

3.1.5.4 VISR

VISR (View based Integration of Web Service Registries) is developed as a P2P architecture for distributed Web Service registry. They transform the Web Service brokerage model to a distributed model with implicit Web Service brokerage to provide accurate Web Service registry entries. From **FIGURE 3-4** we could have a conceptual overview of VISR's Web Service registry paradigm. Every Web Service provider is at the same time broker requestor, and every peer is part of the virtual Web Service registry.

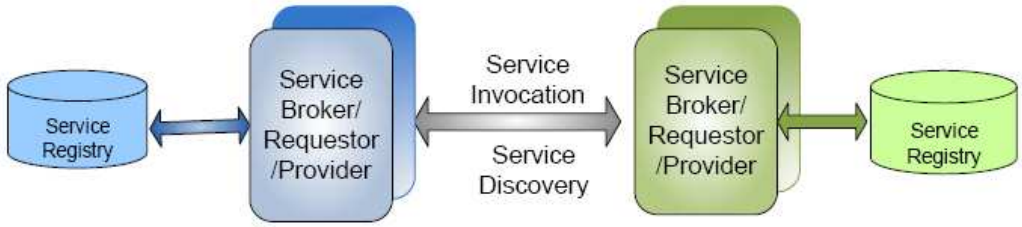


FIGURE 3-4 VISR'S WEB SERVICE REGISTRY PARADIGIM [DUTR06]

Registry information is differentiated into three types in order to provide a flexible data model. They are VISR peer profiles, view profiles and service profiles. Peer profiles are used to identify WS providers. View profiles provide context information about WS and serves to organize the global WS registry content within WS communities. Service profiles tells about WS description in a lightweight.

Regarding WS discovery VISR not only supports keyword based match but also makes improvement by employing the structure of VISR service profiles as potential matching criteria. Besides, Xpath expressions are also supported to select Web Services [DuTr06].

3.1.5.5 Konark service discovery protocol

Similar to VISR, Konark also enables each device to act both a server and as a client simultaneously. A WS requestor uses *active pull* mechanism for WS discovery, while a WS provider advertises their Web Services periodically by the so called *passive push*. For large dynamic ad hoc network Konark has an extended incremental discovery protocol, i.e. Service gossip protocol. By listening to service advertisement, request and response messages in the network, each peer gossips its knowledge about services minus the network's knowledge. In this way, single peer gets the overview of

the whole network. Because repeated gossip of the same information is avoided, this incremental discovery algorithm is supposed to effectively reduce network traffic for service discovery [LHDV03].

3.1.5.6 UpnP

The UpnP is also an industry initiative headed by Microsoft to enable simple and robust connect ivy among stand-alone devices and PCs from many different vendors. UpnP networking usually consists of six phrases: Addressing, Discovery, Description, Control, Eventing, and Presentation. Discovery comes after devices get their addresses.

For service discovery UpnP uses Simple Service Discovery Protocol (SSDP). By SSDP a ***device*** control point declares its presence to other devices and discovers others. Unlike Jini, in which a lookup directory service is necessary for discovery, SSDP could work without directory service. HTTP over multicast (HTTPMU) and HTTP over unicast (HTTPU) are used for advertising. The advertising message contains service type, name and location, an URLs that identifies the advertising service and point to an XML file that provides a description of advertising services. When a ***control point*** joins the network, it publishes discovery queries in form of messages by way of HTTPMU. All devices listen to the standard multicast address for these messages and must respond if any of their embedded devices or services matches the search criteria in the discovery message by way of HTTPU. When the device is removed or wants to leave the network it send bye-bye message to declare its no more availability.

3.1.6 P2P-based Web Service Discovery – the chosen Mechanism for mobile Web Service Discovery

Peer-to-Peer (P2P) computing is not a brand-new technology, since the earliest peer to-peer network in widespread use was the Usenet news server system, in which peers communicated with one another to propagate Usenet news articles over the entire Usenet network. But P2P technology has become a hot topic and begun to draw popular attention since the foundation of Napster in 2000. A P2P computer network relies primarily on the computing power and bandwidth of the participants in the network rather than concentrating it in a relatively low number of servers. P2P networks are typically used for connecting nodes via largely *ad hoc* connections. More information about P2P technology and development is already discussed in **Section 2.3**.

In web services discovery P2P provides an alternative that does not rely on centralized registries; rather it allows Web services to discover each other dynamically. Under this view, a Web service is a node in a network of peers, which may or may not be Web services. At discovery time, a requester agent queries its neighbours in search of a suitable Web service. If any one of them matches the request, then it replies. Otherwise each queries its own neighbouring peers and the query propagates through the network until a particular hop count or other termination criterion is reached.

Peer-to-peer architectures do not need a centralized registry, since any node will respond to the queries it receives. P2P architectures do not have a single point of failure, such as a centralized registry. Furthermore, each node may contain its own

indexing of the existing Web services. Finally, nodes contact each other directly, so the information they receive is known to be current. (In contrast, in the registry or index approach there may be significant latency between the time a Web service is updated and the updated description is reflected in the registry or index.)

The reliability provided by the high connectivity of P2P systems comes with performance costs and lack of guarantees of predicting the path of propagation. Any node in the P2P network has to provide the resources needed to guarantee query propagations and response routing, which in turn means that most of the time the node acts as a relayer of information that may be of no interest to the node itself. This results in inefficiencies and large overhead especially as the nodes become more numerous and connectivity increases. Furthermore, there may be no guarantee that a request will spread across the entire network, therefore there is no guarantee to find the providers of a service [BHMN04].

Because of their respective advantages and disadvantages, P2P systems, indexes and centralized registries strike different trade-offs that make them appropriate in different situations. P2P systems are more appropriate in dynamic environments in which proximity naturally limits the need to propagate requests, such as ubiquitous computing. Indexes may be more appropriate in situations that must scale well and accommodate competition and diversity in indexing strategies.

3.2 Adapting Categorization to P2P-based Discovery

In order to make discovery process for mobile web services more efficient and accurate, various mechanisms are considered. To adapt ***categorization*** in UDDI to P2P based mobile web services discovery is one of them with vital priority because the ability to attribute metadata to services registered in UDDI, and then run queries based on that metadata is absolutely central to the purpose of UDDI at both design time and run time. That's why categorization is arguably the most important feature of UDDI.

UDDI's raison d'être is for the purpose of description and discovery and, as such, the ability to perform searches and queries based on properties and attributes is critical. If data cannot be found or understood, that data is functionally non-existent or worse, misleading. Data is worthless if lost within a mass of other data. Even if an entity *is* discovered, if the user cannot determine the context about the entity – how it is supported, who owns it, what it does, etc. – the user cannot effectively interact with the entity. In other words, being able to distinguish and differentiate data is *as important as* being able to find data [Janu02].

This section consists of two parts. In the first part we will have a close reading to see how UDDI specification handles categorization. The second part elaborates a conceptual model to deploy categorization in P2P based discovery.

3.2.1 Categorization in UDDI Business Registry

Besides the ability to mark UDDI registration data with identifiers, another design goal of UDDI is the ability to assign category information. Without categorization, locating

data within a UDDI registry would prove to be very difficult. Especially for the discovery of previously unknown businesses, services, bindings or service types, it is indispensable that the corresponding UDDI registration data is marked with a set of categories that can universally be searched on. For example, the Universal Standard Products and Services Classification (UNSPSC), a set of categorization codes representing product and service categories, can be used to specify a business' product and service offering in a more formalized way [UDDI04].

3.2.1.1 Simple categories

All four main UDDI data structure types, which are **businessEntity**, **businessService**, **bindingTemplate** and the **tModel**, provide a structure to support attaching categories to data. By providing a placeholder ***categoryBag*** for attaching categories to these data structures, any number of categories can be used for a variety of purposes. The information about categories is added to the UDDI data structure types by using a ***categoryBag***. A ***categoryBag*** consists of one or more ***keyedReference.*** Each keyedReference contains three attributes:

- **tModelKey**: uniquely identifies the tModel that represents the category system
- **keyName**: human readable name of the category system, and when the actual category is coded, a human readable rendition of the value
- **keyValue**: the actual category code within the specified category system

The ***find_tModel*** call is used to find available resource of UDDI. By setting

```
keyValue = ”categorization”
```

all category systems registered with a specific UDDI registry according to some recommended policy to recognize category systems could be found. An alternative is to search by keywords under *Utility tModels and Conventions,* the general keywords taxonomy of UDDI. In this case the keyword to be searched should be set to `keyValue` accordingly.

The fragment of an XML document in **Appendix B** presents a simple example of the usage of ***categoryBag*** in one of the UDDI data structure types, **businessEntity** .

3.2.1.2 Grouping categories

When the use of single categories is not enough to describe an entity, the relationship between single categories could be used. ***keyedReferenceGroup*** serves for this purpose to contain a set of ***keyedReference***. In this way an entity could be described with more details [UDDI04].

By setting
```
keyValue = ”categorizationGroup”
```

and using ***find_tModel*** call all category group systems that are registered within a UDDI registry that follows the recommended policy for recognizing category group systems could be found. The fragment of an XML document in **Appendix B** presents an example of the usage of ***keyedReferenceGroup*** in ***categoryBag.***

3.2.2 How to Deploy Categorization in P2P based Discovery

3.2.2.1 Design of implementation of categorization in JXTA/JXME network

As one can see, in UDDI categorization helps the Web Services requester to find the needed services more quickly and efficiently because the searching scope of instances could be narrowed immediately with a given standard. With the same idea it could also help to speed up the web services discovery in JXTA environment. For the ULD such as mobile devices the better related searching results makes even more sense than devices with no much processing and memory limits.

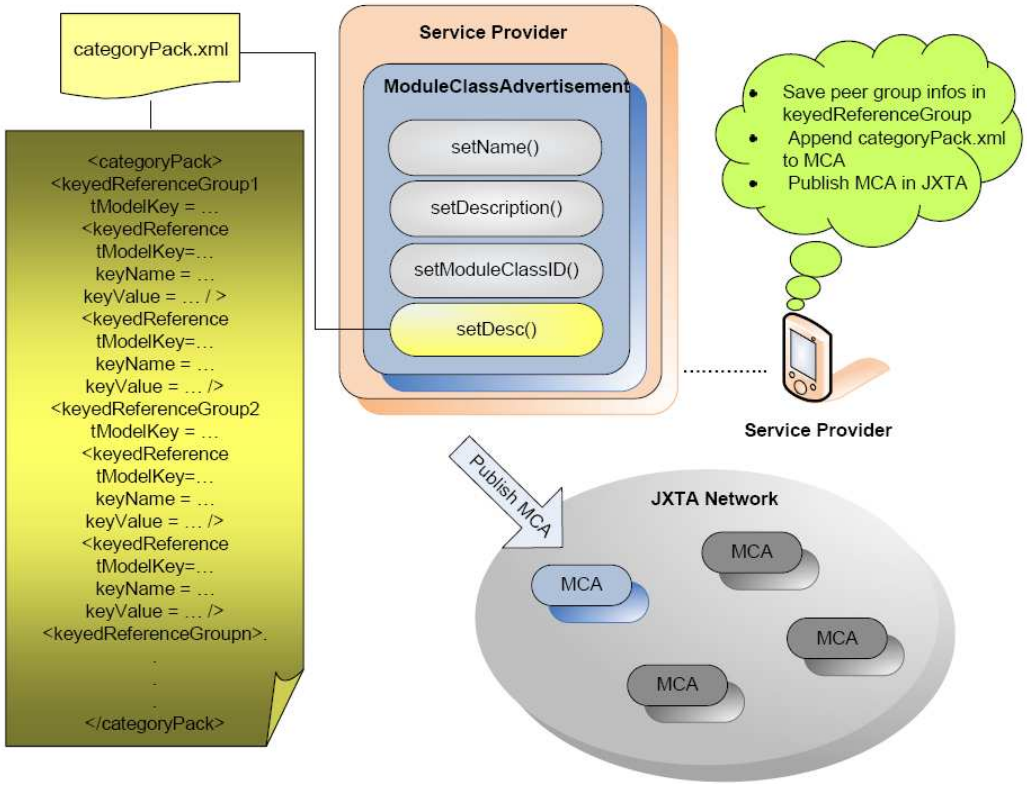


FIGURE 3-5 TO ADAPT CATEGORIZATION TO JXTA MODULE [ELEN03]

In order to deploy categorization in our project, we also need a placeholder to attach information about categories to Module Advertisements from Service Provider. We call the structure which contains category information ***categoryPack***. It introduces peer groups by way of ***keyedReferenceGroup*** similar as the ***categoryBag*** in UDDI. A peer group could belong to one or more ***keyedReferenceGroup***.

The ***categoryPack*** will be defined and appended in Module Class Advertisement as in FIGURE 3-5. When a peer starts the JXTA, declares its existence and gets its unique mobile ID in the JXTA network. At the same time it publishes its category information and will join in some (one or more) related PeerGroup(s) according to the categorization (see FIGURE 3-6).

The super peer of each PeerGroup caches the peer ID and `categoryPack` of each peer in the PeerGroup. When a service requester searches for a service, the request

will be sent to the related super peer(s) at first. The super peer looks through the cache and sends the request to the still available peer IDs further.

Each peer who receives the request from its super peer conducts the search process in its local cache by searching for the MCAs according to the keyword given by the mobile requestor. An MCA maps the general description of Web Services under the same group. Since more than one Web Services could possess the same general description but different detailed description and content, one MCA could possibly match more than one MSAs. Actually when an MSA is to be published in some peer group(s), it should search for the most appropriate MCA in the chosen peer group(s) to match by the category information provided by ***keyedReferenceGroup*** in MCA.

If the number of found MSAs is small, then the searching results could already be accepted and sent back to the service requester. Otherwise some deep search mechanism such as Lucene will further filter and improve the searching results. In this way, it makes possible that the most related services could be found at the first filtering stage.

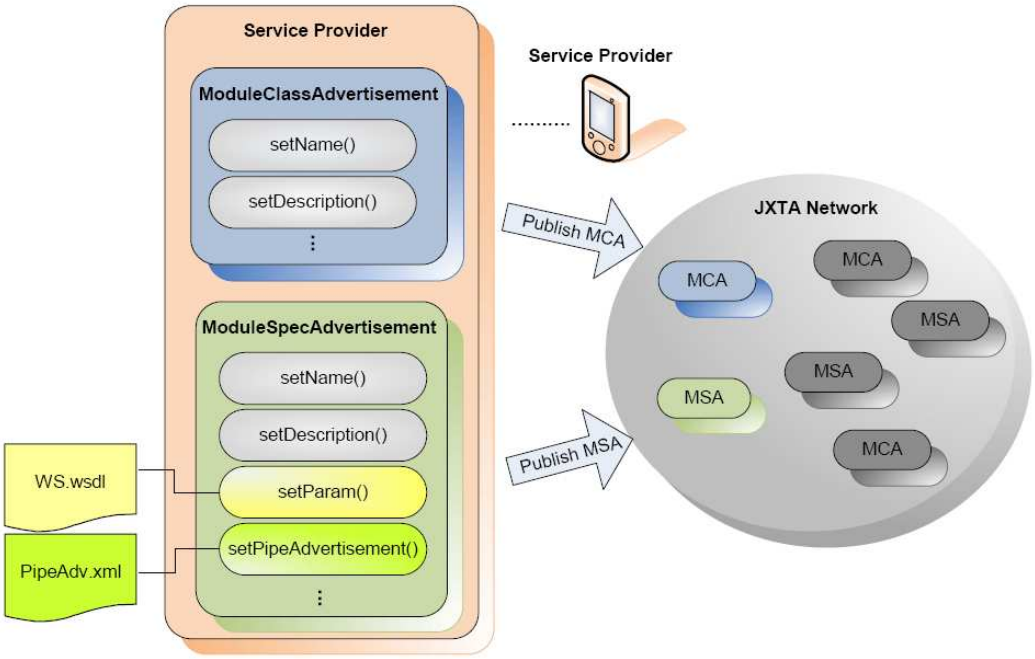


FIGURE 3-6 TO PUBLISH CATEGORY INFORMATION INTO JXTA

3.2.2.2 Categorization structure of mobile Web Services

One of the main focuses of this thesis is to improve efficiency of discovery service with the introduction of a hierarchical peer group structure, which categorizes the mobile Web Services in the JXTA/JXME network. With the reference of some popular industry categorization standards *North American Industry Classification System* (NAICS) and *United Nations Standard Products and Services Code* (UNSPSC), we design a categorization structure of mobile Web Services in **FIGURE 3-7.** Neither have

we the ambition to build up a complete categorization hierarchy nor do we intend to include all possible existing mobile Web Services into this hierarchy because those are not our intension. The group structure which we build is a first draft to realize the idea of categorization. It could be complemented and/or corrected by the mobile Host who would publish some Web Services but find no satisfactory groups in our categorization. In **FIGURE 3-7** we define each peer group with a group name e.g. `profScienTecGroup` and a brief group description, e.g. professional, scientific and technical services for `profScienTecGroup`. The groups on the same level have the same background color, e.g. all groups with green background belong to the third level.

On the top level is mobile Web Service group with the peer group name

`mWSGroup`.

On the second level eleven peer groups are defined and direct child groups of mWSGroup, they are

`accomCaterGroup, artCultureGroup, educationGroup, entertainmentGroup, FinanceInsuranceGroup, healthGroup, informationGroup, profScienTecGroup, repairGroup, rentLeaseGroup, searchGroup.`

On the third level there are thirty-three peer groups, which are

`accomGroup, caterGroup, artSportGroup, coLearnGroup,informalLearnGroup, jitLearnGroup,locBaseLearnGroup,museumGroup, mobiLearnGroup,ambuGroup, profitEntertainGroup,gameGroup,bankingGroup,paypalGroup,dentistGroup, firstAidGroup,homeCareGroup,internetGroup,telecomGroup, geoMapGroup, videoaudioGroup,publishGroup,softGroup,draftingGroup,designGroup, enginneringGroup, autoRepGroup,commNavEquipGroup, realEstateGroup, rentalGroup, videoSearchGroup,picSearchGroup,musicSearchGroup.`

On the fourth level there are twelve peer groups, which are

`casinoGroup, lotteryGroup, locBaseServGroup, rssGroup, newspaperGroup, directoryGroup,periodicalGroup,bookGroup,parkingGroup, internetAccGroup, guideServGroup, autoRentGroup`

Each peer group in the hierarchy except leave peer groups on the fourth level has one or more child groups, e.g. `mobiLearnGroup` has child groups `informalLearnGroup, locBaseLearnGroup, coLearnGroup, jitLearnGroup.`

The application of hierarchical categorization tree will be implemented by way of a shell command in JXTA platform. In both the Windows Command Prompt window and a UNIX shell, a number of built-in commands could be used to perform some simple operations. The JXTA shell features a mix of both simple and complex commands. It is an interactive application, which enables the user to have a look at the JXTA environment and perform operations. Like any other shell, the JXTA shell issues a prompt (**JXTA>**) on which the user could give input of shell commands. The shell command then sends the corresponding output on the screen. For example, ***whoami*** command is used by JXTA to show either the peer information of the current user, or the peergroup advertisement of the group currently logged into. In order to publish the hierachical categorization information into the JXTA network, we will implement such a shell command ***category*** so that all the peer groups on **FIGURE 3-7** will be created

with the hierachical structure and published into JXTA network when the user type the command at the JXTA shell.

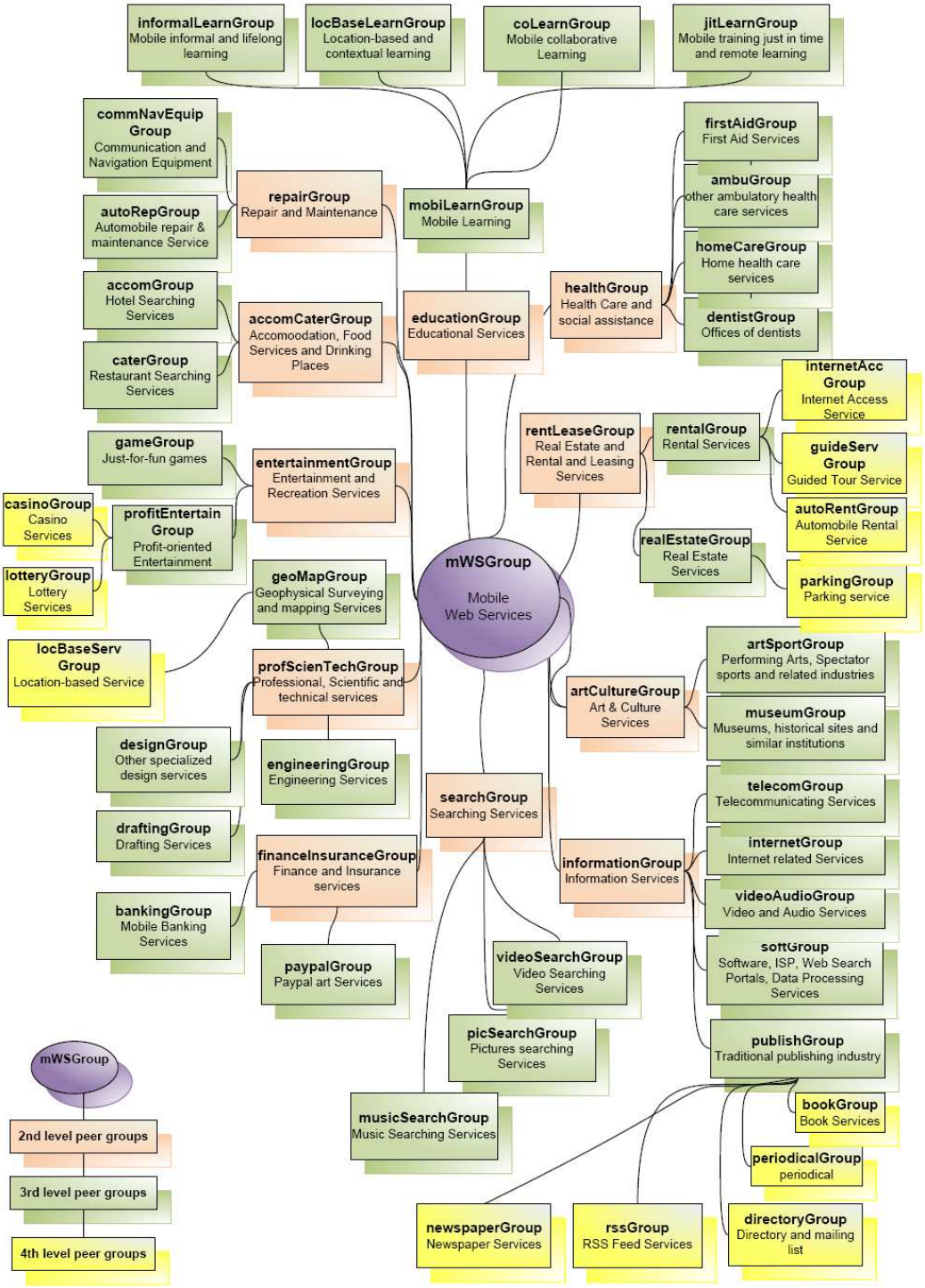


FIGURE 3-7 MOBILE WEB SERVICES CATEGORY HIERARCHY

3.3 Web Services Invocation

When Web services are searched, found and established at mobile devices, it is then the time to invoke them. 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. In previous work Port Forwarding model is deployed, which is using JXME pipes to pass messages among peers. This model works as follows:

To receive incoming messages over the JXTA network it is required to create a pipe. A server creates a pipe using peer ID, which is attached to **Module Spec Advertisement** while the service provider application pushes the MSA service into JXTA network.

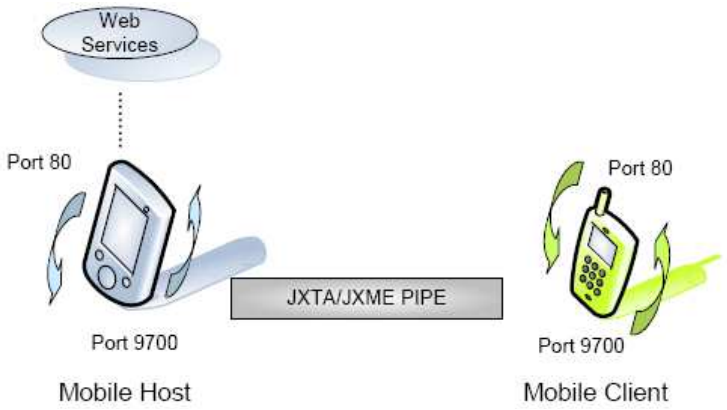


FIGURE 3-8 PORT FORWARDING MODEL [TOPR06]

A client side 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.

Because in JXTA environment any peer could be server as well as client, we will consider the implementation in one side. The pipe application consists of three main methods:

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

After a pipe is created with peer ID, it is required to send a listen request message to the JXTA relay peer. The relay peer starts listening messages. 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.

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.

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 behaviour. 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.

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. Thus a full invocation process is done.

3.4 Summary

Different approaches and technologies are studied and compared to find out a best alternative for mobile Web Service discovery. Among registry, index, dynamic and P2P paradigms we think P2P best suits this need because of its flexible and dynamic nature. However, we also consider to borrow good properties of UDDI while developing our P2P based discovery mechanism. Categorization is considered a key feature of UDDI in Web Service discovery process and is therefore included in our development. A hierarchical structure of peer groups is designed for JXTA/JXME network with the assumption that it could bring efficiency as well as scalability to discovery performance.

4 Implementation

This section presents the detailed implementation of mobile Web Service Discovery with categorization paradigm. At first the implementation environment and tools are introduced in **Section 4.1**. Then we review the overall process of implementation through a sequence diagram. From the aspect of different participants into the discovery process we divide the narration of implementation into three parts: service provider, mobile service client and JXME proxy.

4.1 Implementation Environment and Tools

Java is deployed as the main developing language throughout the implementation of this work from the aspect of both server and client, although the role of server and client could be played by the same device. In JXTA platform, through which P2P based mobile Web Services discovery, is realized, Java is also used instead of other languages thanks to its platform independence, object-oriented methodology and its built-in support for networking. Java 2 Micro Edition is no doubt the version chosen to develop client side user interface because it is running on mobile devices. In addition, JXME, i.e. JXTA Micro Edition API is deployed to enable the function of connecting and communicating within JXTA platform.

Eclipse is chosen as the developing framework on the aspect of service provider because of its being **Rich Client Platform**[10]**.** Due to the same reason NetBeans IDE is used to develop service requester application. The screen show function of NetBeans provides convenient overview about the mobile interface. It is the second reason why NetBeans is used on the client side.

4.1.1 Java 2 Platform and J2ME

The Java 2 Platform is a computing platform from Sun Microsystems which can run applications developed using the Java programming language and a set of development tools. Java has three kinds of versions: Java 2 Standard Edition (J2SE), Java 2 Enterprise Edition (J2EE) and Java 2 Micro Edition (J2ME). These versions are developed for different environments and for different devices.

J2ME platform is a collection of Java APIs for the development of software for small devices with limited memory, display and power capacity 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. To serve this goal modular structure is developed, different kind of profiles and configurations are defined. J2ME has two configurations: CLDC (Connected Limited Device Configuration) for personal intermittent network connections and CDC (Connected Device Configuration) for continuous network connections.

[10] A **Rich Client Platform (RCP)** is a piece of software consisting of the following components: a core,a standard bundling framework, a portable widget toolkit, file buffers, text handling, text editors, a workbench (views, editors, perspectives, wizards), Update Manager.

For our purpose to implement a Web Services discovery environment for mobile devices the combination of CLDC and MIDP (Mobile Information Device Profile) would be the best alternative, just as shown from the **FIGURE 4-1**. CLDC 1.0 and MIDP 2.0 are the configuration and profile used in the thesis and they are introduced in more detail in next section.

4.1.1.1 CLDC 1.0

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 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

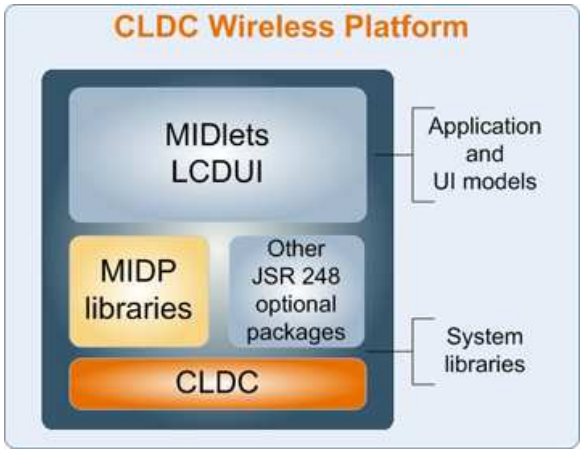


FIGURE 4-1 CLDC WIRELESS PLATFORM[11]

4.1.1.2 MIDP 2.0 and MIDlet

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.

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,

[11] http://java.sun.com/javame/technology/index.jsp

MIDP provides a complete Java runtime environment that leverages the capabilities of handheld devices and minimizes both memory and power consumption.

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 [DeJo04].

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 [MID03].

4.1.2 Eclipse SDK 3.2

Eclipse is an open-source software framework originally written in Java. But it is far more than a Java IDE. Its main focus is to build an extensible development, runtime and application framework to manage software along its complete lifecycle. Eclipse SDK provides an IDE with a built-in Java compiler and neat presentation of Java source files. To manage a huge project such as JXTA, such kind of support is absolutely necessary. Eclipse provides a set of convenient tools for creating, compiling, testing and modifying our project. With usage of workspace external files could be created and refreshed, it further supports our purpose to access and process external files. In addition, Eclipse serves our need to create platform-independent web application in respect to every aspect. Therefore we chose Eclipse SDK (Software Development Kit) as developing platform especially for Proxy/Relay service provider.

4.1.3 Sun wireless Toolkit and Sony Ericsson SDK

The Sun Java Wireless Toolkit is used for test together with Eclipse in the beginning of requestor implementation .The Sun WTK 2.5.2 provides a set of tools for creating Java applications that run on devices compliant with the Java Technology for the Wireless Industry (JTWI, JSR 185) specification and the Mobile Service Architecture (MSA, JSR 248) specification. It consists of build tools, utilities, and emulators.

4.1.4 NetBeans IDE and NetBeans Mobility Pack

During the implementation process on the mobile Web Services requester side of this thesis NetBeans plays the main role both as platform to develop J2ME application and

as an integrated development environment (IDE). The NetBeans IDE is an open-source integrated development environment written entirely in Java using the NetBeans Platform and supports development of all Java application types.

4.1.4.1 NetBeans Platform and NetBeans IDE 6.0

With NetBeans applications could be developed from modules, i.e. a set of modular software components. The Java classes which are edited to interact with NetBeans APIs and a manifest file declaring its module identity construct a Java archive file and each such Java archive file is a module. Extensibility is a very advantageous feature of modules because it simplifies the development for the third party developer who would work further on the applications on the basis of modules. Besides, the platform provides services such as user interface and setting management, storage management, window management etc. to help developers focus on the logic part of developing task.

4.1.4.2 NetBeans Mobility Pack

The main reason for us to choose NetBeans to develop mobile Web Service requester application is NetBeans Mobility Pack. Not only does it integrate support for MIDP 2.0 and CLDC 1.1, but the functions and tools it provides greatly simplifies and clarifies the logic specifications of codes while editing, testing and debugging applications for J2ME platform. One example is the screen functions which presents not only source code, but Screen Design and Flow Design. Since both screen design and flow design are on one side routine coding, on the other side needs visual feedback for the source coding while editing, the tools enormously eases the editing process. The developer could then more focus on logics of coding task.

4.1.5 JXTA and JXTA Java Micro Edition (JXME)

As introduced in **Section 3**, we assume a P2P based discovery mechanism would be a better alternative for our mobile applications. JXTA is designed to enable the refactoring of many applications in a P2P environment rather than being a library for creating concrete applications. JXTA aims to provide services and infrastructure for P2P applications. Its group concept e.g. serves to organize randomly distributed peers and embodies them common entities and services. The routing and communicating mechanism frees the peers from a variety of usual barriers of traffics such as firewalls. Those principles exactly respond the requirement and need of our project. We need such a truly distributed system for mobile devices and at the same time super peers to perform the resource-consuming heavy load tasks such as keeping files, processing messages etc.

The JXTA Java Micro Edition (JXME) provides a JXTA compatible platform on resource constrained devices using the Connected Limited Device Configuration (CLDC) or the Mobile Information Device Profile 2.0 (MIDP), or Connected Device Configuration (CDC). The devices are ranged from smart phones to PDAs. By way of JXTA Java Micro Edition platform, the CLDC/MIDP/CDC devices can participate in the JXTA network and communicate with other JXTA devices[JXME05][12].

[12] https://jxta-jxme.dev.java.net/

In **Section 2.4**, we have reviewed the key concepts and architecture of JXTA, including three simple software layers: core layer, service layer, application layer, the relationship between peers and groups, the form of different advertisements and IDs, six protocols with respective functional domain, pipes for sending and receiving messages, as well as concept of module as a media to represent any behaviour of applications.

In this section, we will firstly go behind the concept facet and present how JXTA API support us with implementation from Web Service provider's aspect. Then what interests us is how JXTA Java Micro Edition contributes to enable a light weight implementation from mobile Web Service requester's aspect possible.

4.2 Overall Publishing and Discovery Implementation

Although our focus in the thesis is discovery process of mobile Web Services, we could not present a complete workflow of discovery process without an overview of publishing process. This **section 4.2** provides a bird's eye view of mobile Web Services producing and consuming behaviours with the help of a sequence diagram. It starts from bootstrap of the mobile P2P network and ends with the invocation of the found Web Service as shown in **FIGURE 4-2**:

- **Bootstrap**

First of all, Mobile P2P network needs to be established and category information is published into the P2P network. The peer from MWSMF (Mobile Web Service Medium Framework) joins JXME network as a super peer. A mobile service requestor starts JXME and joins P2P network via MWSMF.

- **Publishing**

Some Web Services are developed by some developers. The developers join some peer group(s) according to the category information and publish available Web Services in the P2P network. A mobile Host deploys a Web Service. The mobile Host joins P2P network by connecting with a super peer via MWSMF and chooses one or more groups to join according to category information.

- **Requesting**

A mobile service requestor needs some Web Service and chooses a peer group, in which the service requestor believes some Web Service is most possible to be found, and provides a keyword. With provided group name and keyword by service requestor the JXME proxy starts the discovery process.

- **Searching**

When the JXME proxy of the service requestor received search request. The keyword based search starts in its local cache and will be forwarded to all the known Rendezvous peers and edge peers if necessary. The chosen peer group has to be found at first. Then MCAs based on the given keyword are searched in the chosen group.

If some keyword matching MCAs are found, the MCA IDs are extracted to search for the matching MSAs to MCAs. If the result list of MSA is not very long, i.e., the number of found MSAs does not surpass the threshold set by service requestor, then the searching process is over and the result list of MSAs are ready to be sent back to service requestor. If the number of found MSA is greater than the threshold, then deep search mechanism will be deployed to shorten the result list.

If no MCA is found in the provided peer group by service requestor, then a depth-first search for MCAs proceeds in all child groups of the given peer group till some MCA is found or till all its child groups are visited.

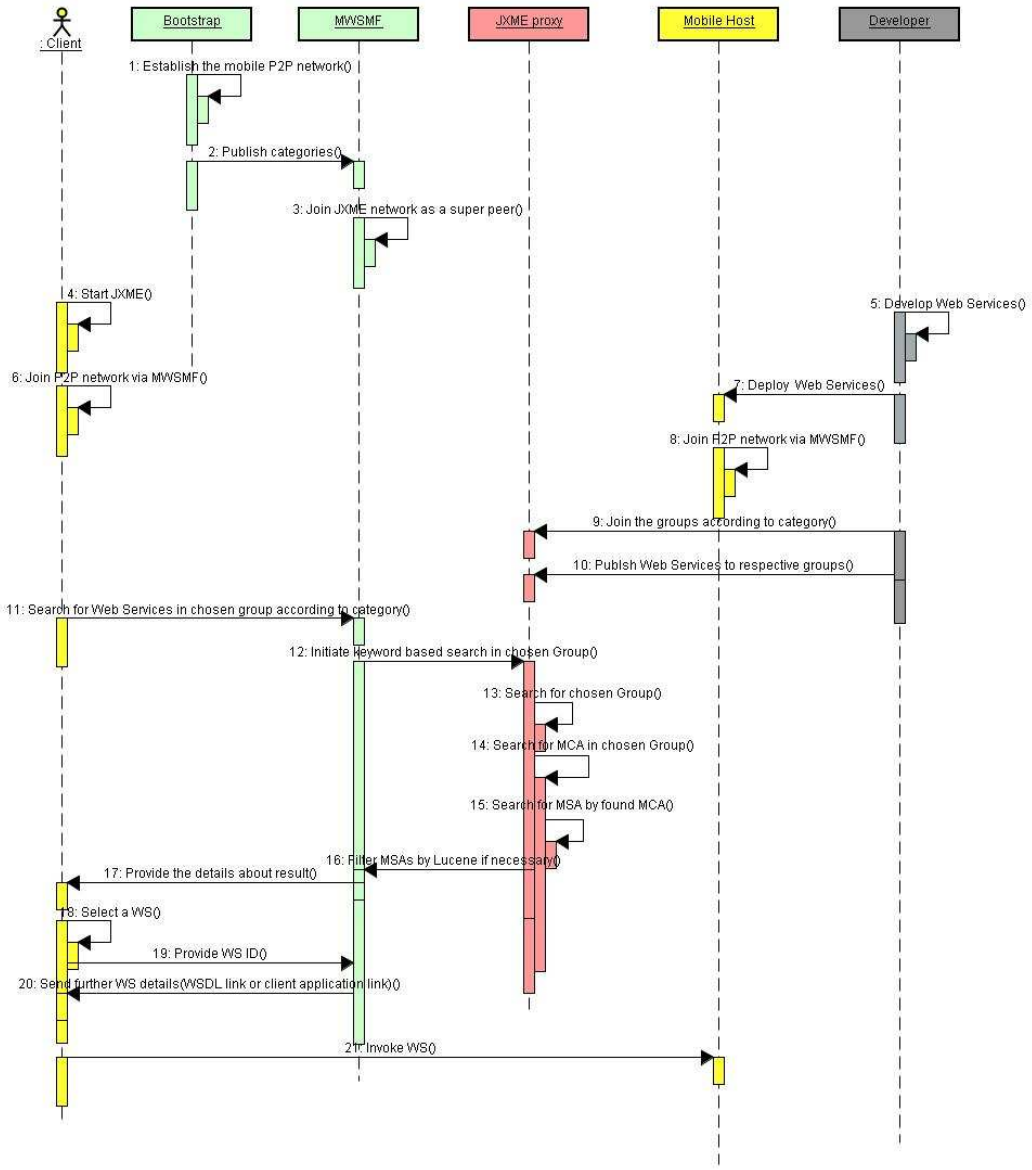


FIGURE 4-2 PUBLISHING AND DISCOVER OF MOBILE WEB SERVICE [SRIR06]

- **Choosing from found MSAs**

When the result list of MSAs are sent to service requestor, it should choose one from them according to the provided key information such as name and description of Web Services on the mobile screen. By scrolling up and down, the requestor chooses one which interests her/him the most. Now the requestor needs further information about how to invocate the Web Service. Therefore it sends a message to the service provider expecting more detailed information.

- **Invoking**

Service provider received the request for invoking information of Web Services and sends the requestor the necessary information such as link of WSDL file of Web Services or application link. With the information service requestor could then invoke the Web Service.

When considering how to present the implementation process, we decide on one side to follow the sequence phrases which are just mentioned above, on the other side take the participants into the discovery process as standpoints. In the following detailed presentation about implementation discovery process, we would therefore generally introduce the process into three sections, which are service provider (**section 4.3**), service requestor (**section 4.4**) and JXME proxy (**section 4.5**).

4.3 Service provider Implementation

In this section we start to introduce the implementation from the viewpoint of Service provider. JXTA/JXME is the network to enable P2P communication possible, therefore a peer has to be a member of JXTA/JXME network, i.e., to install JXTA, do configuration and log in the network in form of a shell before it intends to perform any action in the communication process. Usually a service provider proceeds the followings phrases starting from connecting to some super peer in JXTA :

- Connect to Super Peer
- join JXTA
- get Peer ID
- Publish category information by way of MCA
- Join respective Peer Group(s)
- Publish available Web Services in form of MCAs and MSAs
- Ready to handle request messages from mobile client

Service provider is any entity who possesses some Web Services and is ready to share it with those who need it against cost or for free. To enable the requestor know the existence of some Web Services, some kind of broker is needed, who functions to bridge the supplier and consumer so that both parties are informed for further possible information exchange. Who plays the role of broker is the critical benchmark for discovery approach.

In **Section 3.1** we have introduced four types of discovery approach form different viewpoints: registry, search engine, dynamic and peer-to-peer. In the first three viewpoints the role of broker is played mostly by a third party as provider and

requester. Take one example of registry approach, UDDI, e.g. the single central registry as well as the broker is UDDI index. An example of search engine, Google, e.g. has the web server as broker. Although dynamic discovery approach makes progress with the broker stub model, a central broker must still often be available.

The principal feature of P2P approach which differs itself from all the other three viewpoints is that **NO** third-party broker has the necessity to exist in order to convey data about available Web Services in the network. By multicasting or broadcasting advertisements a service provider announces the availability of Web Services within TTL (Time To Live) periodically. In our improved version of P2P discovery approach with the introduction of categorizing mobile Web Services into hierarchical groups we have the hypothesis that searching cost should be reduced and the discovery mechanism should be more scalable. In order to support our hypothesis we will do scalability test and performance analysis in **Section 5**.

We start to introduce the implementation from the very beginning of the hide-and-find game. The narration begins with how to start JXTA platform, including how to start and configurate a JXTA shell, followed by categorization implementation, the main focus of this thesis. After that comes the publishing of advertisement of mobile Web Services. In the end we will know how the shell command "**category**" is created so that every peer could join the hierarchical group structure as soon as it joins the JXTA network. To give a brief bird eye's view, the process takes place in following stages:

4.3.1 To Start JXTA

4.3.1.1 To Start a JXTA shell

Before starting a JXTA shell, some elementary work has to be done. All the advertisements published by the peer must be saved in local cache, the so-called `JXTA_HOME`, which is nothing but a directory in the hard disk. `JXTA_HOME` could be set in system configuration of operating systems or in the codes. By system configuration the advertisements of all peers on the same computer has to be put in the same directory. This is not what we want to see, since it could lead to wrong discovery results if the advertisements lie in the directory of the peer, which they should not belong to. For this reason we set the `JXTA_HOME` every time according to our need before we start JXTA platform in codes. Conventionally, the newly created advertisements as well a file called `PlatFormConfig` is saved in a directory named `.jxta`. With the function `System.setProperty()` we set then `.jxta` the directory as the home directory to store files created by the system.

Every time when a shell is newly started, files of advertisements are created and saved in the above mentioned directory. Obsolete Advertisements could affect some operations such as searching time in discovery process. It is therefore necessary to keep the cache tidy before we start the JXTA platform. With function `clearCache()` the cache, to be more exact, the files in `.jxta/cm` are cleared by deleting every time before the JXTA platform starts.

To enable our application to be a peer and a member in JXTA network, we firstly create a default peer group by calling the `NetPeerGroupFactory()` and `getInterface()`. This group provides many useful services such as to discover

and create new groups. To get some service from this group, e.g. discovery service, just call the function `NetPeerGroup.getDiscoveryService()`. Generally, all JXTA applications belong to this group by default.

4.3.1.2 To Configurate a JXTA shell

JXTA configurator is presented at the first time of application execution or when the user wishes to reconfigurate the peer. Before starting the JXTA platform it needs to be configurated by providing the following information:

1. Basic settings:
   a name and password for the peer.

2. Advanced settings:
   Service settings (relay, rendezvous, JXME proxy),
   Connection information settings (protocol, IP Address, Port etc.)

3. Rendezvous/Relay settings

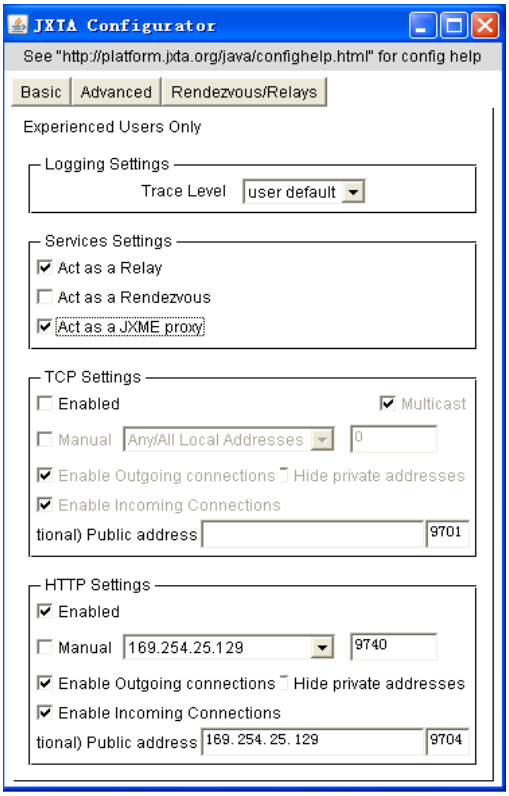


FIGURE 4-3 JXTA CONFIGURATOR

In **FIGURE 4-3** is a screen shot of JXTA configuration advanced setting. Detailed screenshots about configurator settings could be found in **Appendix D**. The configuration is needed either at the first time of application or is chosen to be done by the user to change the settings for some new usage.

One of the main focuses of this thesis is to improve efficiency of discovery service with the introduction of a hierarchical peer group structure, which categorizes the mobile Web Services in the JXME network. With the reference of some most popular industry service categorization standard, we design a categorization structure of mobile Web Services in **FIGURE 3-7.** However, we do not have the ambition to build up a complete categorization hierarchy. The group structure which we build is a first draft to realize the idea of categorization. It could be complemented and/or corrected by the mobile Host who would publish some Web Services but find no satisfactory groups in our categorization. In the next section we will introduce how the categorization of mobile Web Services is implemented in JXTA platform.

4.3.2 Categorization implementation

Without categorization mobile Hosts declares the existence of a Web Service by just publishing the relevant advertisements in the NetPeerGroup by default. To categorize Web Services is actually to publish Web Services in respective groups. These groups must be created at first before mobile Hosts do anything with them. In **section 4.3.2.1** we will introduce how peer groups are created and in **section 4.3.2.2** we will introduce how to publish web services in chosen groups.

4.3.2.1 To Create Peer Group

The following information is necessary to create a new peer group: name, description, and who its parent group is. Every group should have its name and short descriptions about its scope according to JXTA protocols. Remember that the every newly created group should be a member of the group structure in **FIGURE 3-7**, so we need to know which group the newly created group belongs to. For this reason we need parent group information to create a new group. By the same function `createGroup()` all the groups, no matter on which level, could be created one by one.

We start with the top parent group in **FIGURE 3-7**: mobile We Service Group. It is the first level of the hierarchy and all other groups belong to this group. Its parent group is default peer group `NetPeerGroup`. We name the group `mWSGroup` and describe it by `"This is a mobile Web Service Group."` All the other groups are built in the similar way by `createGroup()`. The details about the implementation of the whole categorization structure are in **section 4.3.3.**

In `createGroup()` we call the function `newGroup()` of the parent group to get an instance of `netPeerGroup`. Here peergroup implementation Advertisement is also needed, we get it by calling `getAllPurposePeerGroupImplAdvertisement()`. After a new peer group is created with the instance of netPeerGroup and the necessary parameters, it is then the time to declare its existence to the JXTA network. The declaration is done through another type of dvertisement, `PeerGroupAdvertisement` and we get it by calling `getPeerGroupAdvertisement()` from the newly created group. Every JXTA peer group has some ready available services, such as discovery service. One of the functions of discovery service is publishing advertisements in the local cache so as to enable other peers to find them or to other peers in the network. A newly created peer group needs to announce its presence therefore discovery service is necessary. It is called by `getDiscoveryService()`.

JXTA employ **modules** to provide information about services and applications available in a peer group. Three types of modules are defined for peers to discover modules: a Module Class Advertisement (MCA), a Module Specification Advertisement (MSA), and a Module Implementation Advertisement (MIA).

An MCA exists only to announce the existence of a class of module. Here an MCA is needed to announce the existence of a peer group. More detailed features about peer group are saved in `categoryPack` from an XML document. And this `categoryPack` is added to MCA to announce the existence of a class of module and introduce the basic features of this peer group. To create an MCA for our newly created peer group, we call `newAdvertisement()` from `AdvertisementFactory` by providing `ModuleClassAdvertisement.getAdvertisementType()`. Group name and description are set to MCA. Besides, an unique ID is created by calling newModuleClassID `()` from `IDFactory` for each MCA. This ID will play a key role in later phrase of discovery. When the MCA is readily built, it is published to other peers by `remotepublish(mca)` of discovery service of the peer group.

Till now, a new, self-defined group is built and has announced its presence in the network. Every group in our categorization diagram is born in principle in this way. Later in **section 4.3.3** we will further present how the whole categorization structure is implemented. Now we just suppose that the whole structure is already built up and it is the time for mobile Hosts to publish their Web Services.

In this section we introduced how to create a new peer group and announce its existence to other peers. When the groups are available, it is then possible for mobile Host to publish Web Services they would provide. **Section 4.3.2.2** gives an overview of how Web Services publishing is implemented.

4.3.2.2 Web Service publishing implementation

Parameters needed to publish Web Service

Web Services, which is in form of WSDL file, will be published into JXTA network by MCA and MSA. MCA declares the existence of the Web Service. And MSA provides metadata of the Web Service. We need three kinds of variables before we start to implement Web Service publishing required elements for MCA, required elements for MSA and the peer group which the Web Services belong to. Required elements for MCA include `mcaName` and `mcaDescription` , required elements for MSA include `msaName, msaDescription, msaVersion, msaCreator, msaURI,` while `chosenGroupName, discoveryChosenGroup` are about chosen peer group.

For example, we intend to put a Web Service about weather forecast in the group mobile Web Service Group, i.e. `mWSGroup,` it could have `"weather"` as its `mcaName` , `"weather service"` as its `mcaDescription` , `"Weather Forecast Database NOAA"`as `msaName`, `"Weather Forcast"` as its `msaDescription`, *`mWSGroup`* as its `chosenGroupName`.

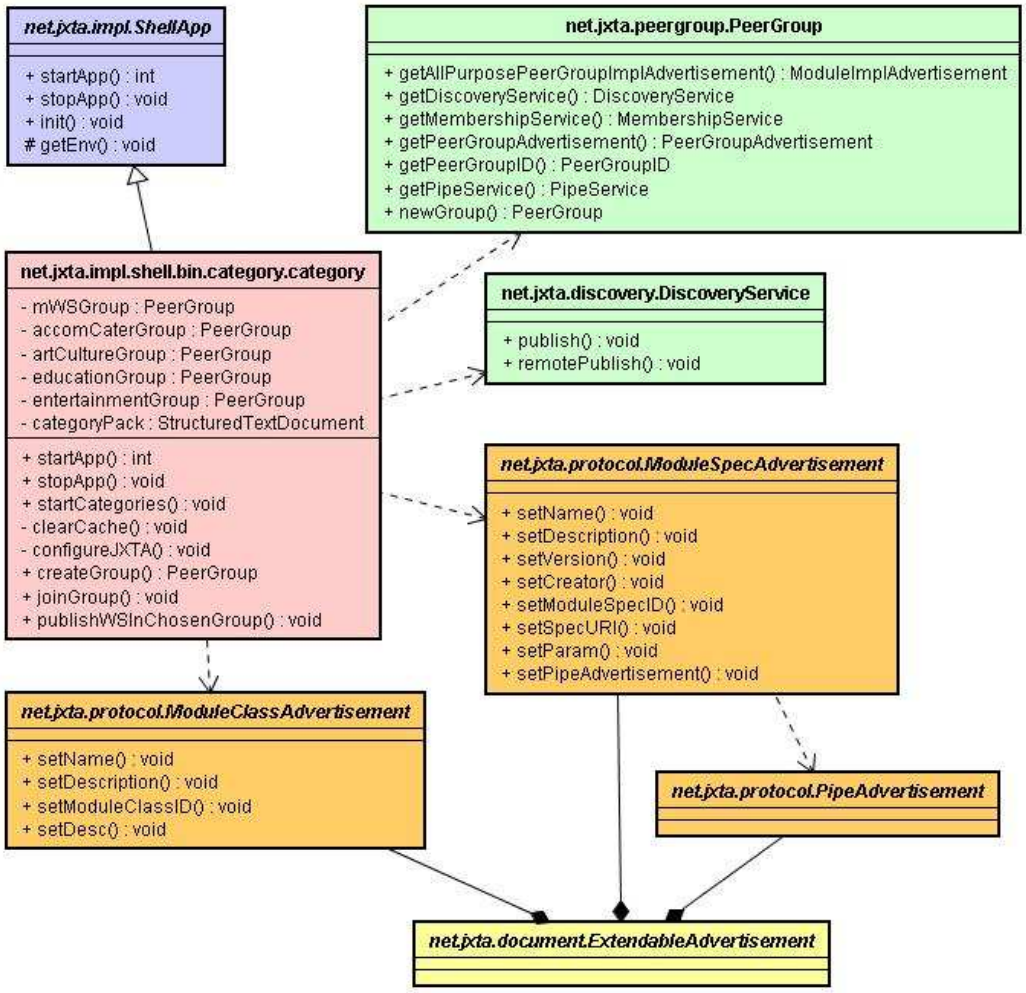


FIGURE 4-4 SERVICE PROVIDER CLASS DIAGRAM

To Find or Construct MCA of Web Service

The first attempt to get a matching MCA for MSA of Web Services is to search in the local cache of chosen peer group by the given MCA name. If there already exists MCA with the given name, then we could get the MCA ID by `getID()` at once. If no MCA with the expected name is found, then such an MCA should be created. After the MCA is created, get an MCA ID by `IDFactory.newModuleClassID()`. MCA ID is needed to build MSA ID because MSA ID is built on the basis of MCA ID. One MCA could have more than one corresponding MSA, while each MSA matches just one MCA (see example in **Appendix B-1**).

To Construct MSA of Web Service

Now the elements are ready to create MSA associated with the Web Service. Just as the creation of MCA we use `AdvertisementFactory.newAdvertisement()` to get `ModuleSpecAdvertisement` at first. Then the information about MSA from the variables such as `msaName`, `msaDescription`, `msaVersion`, `msaCreator`, `msaURI` are set to `ModuleSpecAdvertisement`. With MCA ID a MSA ID is created

by `IDFactory`. The description of Web Service is in a WSDL document. This file could be found in a URL or a directory in the hard disk. We suppose that the Web Services are in the hard disk and get a structured document from the WSDL document And we set the WSDL document in MSA by `setParam()`(see example in **Appendix B-2**).

To Attatch a pipe advertisement to MSA

A pipe advertisement indicates the location, instead of IP, of Mobile Host. A client must use the same pipe advertisement to take to Mobile Host. The pipe advertisement is also set to MSA. When a client finds the modules it needs, it will extract pipe advertisement from MSA and contact mobile Host according to the location information from pipe advertisement. Like other types of advertisements `PipeAdvertisement` is also created by `AdvertisementFactory`. We need to get `peerID` so as to set it as `PipeID`. The `peerID` actually indicates the location of mobile Host and a client contacts a mobile Host by this unique ID for communication. There are three types of message transfer: unicast, propagate, and secure unicast. For communication between single peers we use `UnicastType`. After the needed qualities are set to `PipeAdvertisement`, we write the advertisement in a XML file and append the advertisement to the MSA which it belongs to (see example in **Appendix B-2**).

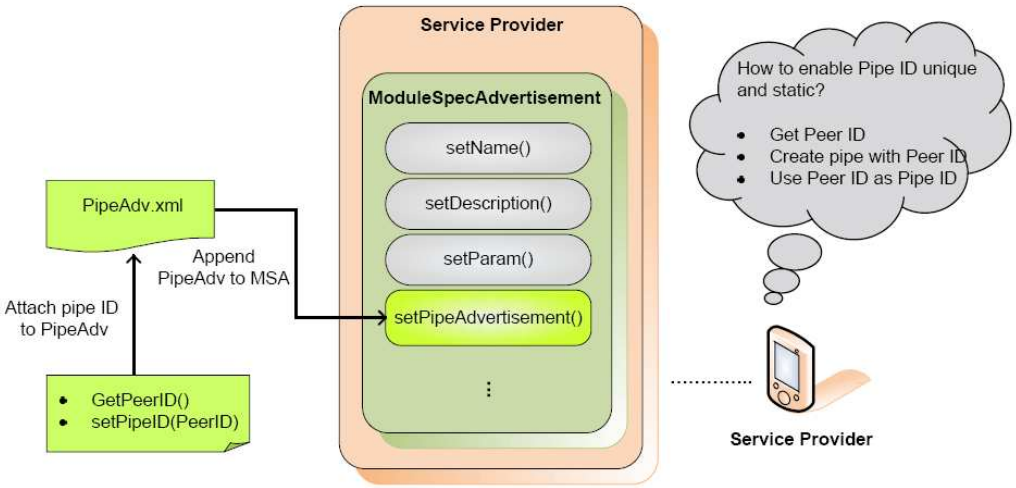


FIGURE 4-5 TO ATTATCH A PIPE AD TO MSA

Time to Publish MSA

Till now the required content of MSA is complete. It is the time to publish MSA. In order to publish MSA in the chosen group we must use the discovery service of that group. As usual, `publish()` and `remotepublish()` are used for local and remote publishing.

In **Section 4.3.2** we present single steps about how to create a new peer group so as to build categorization structure and how to publish Web Services. The actual deployment of this categorization mechanism is not done until the mobile Host user

gives the command on the shell of JXTA peers. We name this command `category`. In this section **4.3.3** we will introduce how the shell command `category` is implemented and how the hierarchal structure of categorization is built up.

4.3.3 Shell command Implementation

JXTA project is designed to create tools that would have familiar look&feel to developers. JXTA shell is such an interactive application that enables direct access to the JXTA network in the same way the Unix shell provides direct access to operating systems. Available commands include e.g. `peers` to discover peers, `join` to join a peer group, `search` to discover jxta advertisements. The shell is extensible and new command could be added with ease. Next we will present how our new self-defined command is created and added to JXTA shell application.

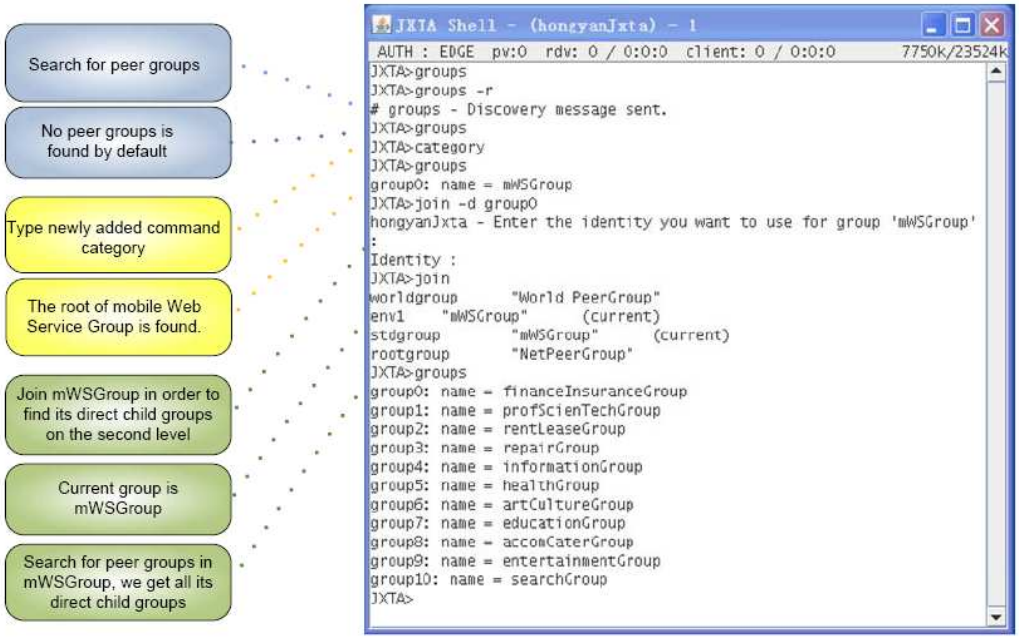


FIGURE 4-6 SHELL COMMAND CATEGORY SCREENSHOT

How to create a Shell command

It is required that a new command should extend `ShellApp`. `ShellApp` is a framework for new commands that contains two methods: `startApp()` and `stopApp()`. These two methods must be overridden when a new command is created. When the users enters a command on the JXTA shell, `startApp()` is called. Therefore the primary work of the new command should be contained in the `startApp()` or called from `startApp()`. We name the new command for categorization hierarchy `catergory`, and all the work of hierachical group creation is contained in `startApp()`. When the work in the command is finished, `stopApp()` will be called to perform any necessary housekeeping.

How to build the hierarchy

The top level of our hierarchy is `mWSGroup`, i.e., mobile Web Service group. On the second level there are eleven groups: `accomCaterGroup`, `artCultureGroup`, `educationGroup`, `entertainmentGroup`, `FinanceInsuranceGroup`, `healthGroup`, `informationGroup`, `profScienTecGroup`, `repairGroup`, `rentLeaseGroup`, `searchGroup`. They are all child groups of `mWSGroup`. On the third level there are again thirty-three groups, which belong to different eleven groups on the second level. On the fourth level we create twelve groups and they take some of the thirty-three groups on the third level as their parent groups.

The service sphere of each group could be interpreted from the name of most groups. A brief description of each sphere could be found in the categorization diagram in **FIGURE 3-7**. `profScienTecGroup`, e.g. is about the professional scientific and technical web services. And `accomCaterGroup`, is about accommodation and catering web services. By way of Map tool `TreeMap` in java we could create all the child groups with the same parent group with one `TreeMap`. The implementation of `TreeMap` is based on red-black tree. The keys or the pairs are in sorted order if you get the results presented.

4.4 JXME Mobile Client Implementation

4.4.1 JXME Client Application

There are two versions for JXME: proxy version and proxyless version. Till now no stable proxyless version is available yet therefore we still use proxied version.

- Search WS by category
- Find the related Peer Group(s)
- Send request to Super Peer
- Deep research mechanism to improve discovery results at middleware framework
- Get the service description
- Invoke the service

4.4.2 Search in Group Request implementation

After the mobile user chooses the peer group, in which the Web Services it requires, and a keyword for searching, The method `searchInGroup()` of JXME Class `peerNetwork` is called (see **FIGURE 4-7**). The original version of searching in JXME class has no group element, so we added this method with group element to suit our need to search Web Services in some group of the categorization hierarchy. In this method, a message consisting of seven elements is built up and is sent as a request to the JXME proxy with which it is connected. These elements include information about the type of request, request ID, the chosen group and the keyword from the user.

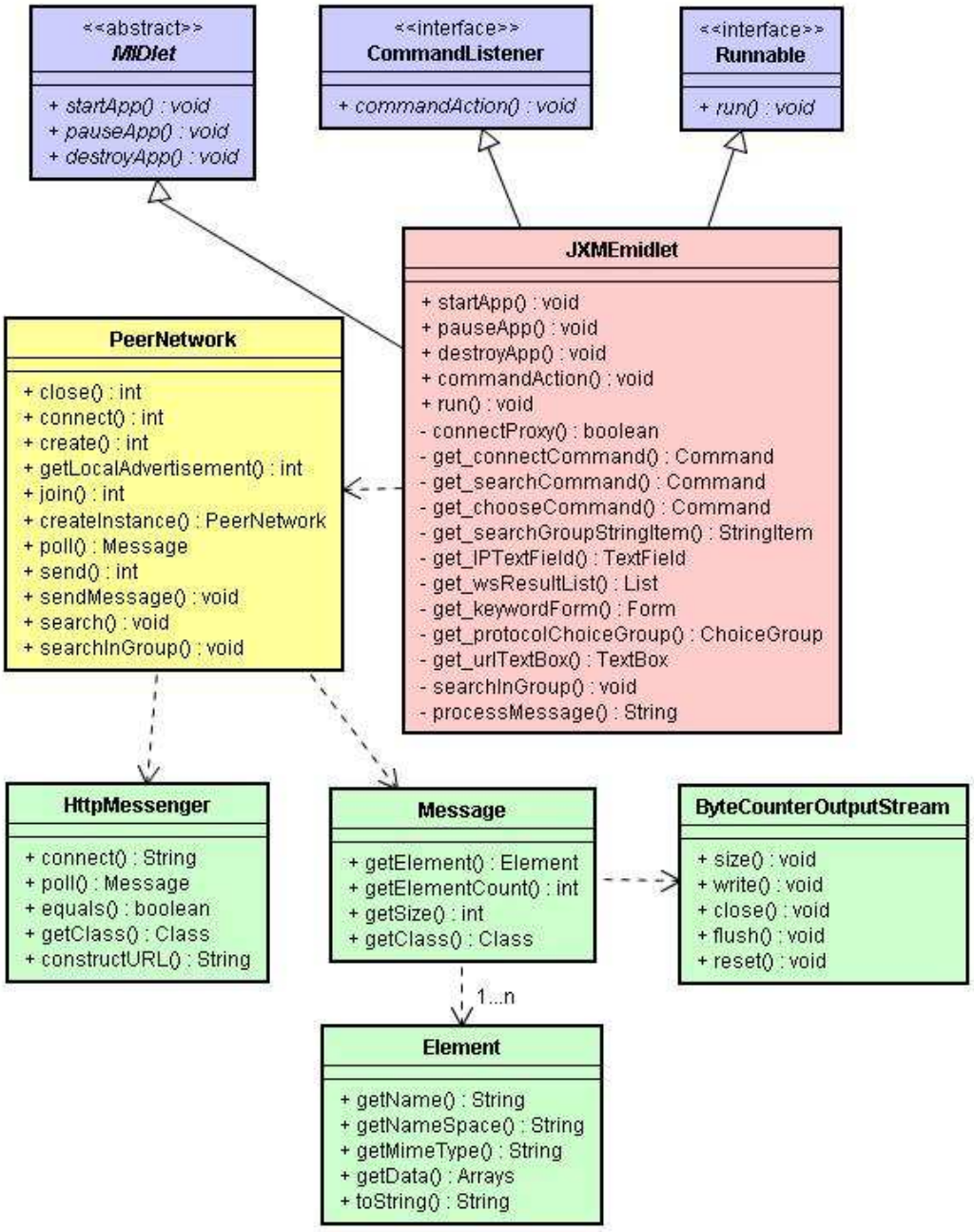


FIGURE 4-7 JXME MIDLET CLASS DIAGRAM

When the JXTA peer who is acting as proxy for this JXME midlet receives incoming messages, it processes them with the method `processIncomingMessage()`, This method about how to process the message and how the Web Services are searched will be covered in **section 4.5** when we introduce JXME proxy implementation. Now let's talk about the searching result. With method `poll()` JXME MIDlet keeps watching incoming message at a time interval. If the searching in JXME proxy is successful it will send message back to inform JXME MIDlet of searching result about Web Services. The message contains information about all the MSAs found by

searching mechanism. From them the MSA names are picked out and printed out to the screen of JXME MIDlet. The MSA names are listed one by one for mobile users to choose. In **section 4.3** we introduced that MSA name represents the name of Web Services. Therefore based on the name of available Web Services the user could choose the one which suits its need the best.

4.5 JXTA Proxy/Relay Implementation

The task of JXME Proxy is to listen to the request from JXME midlet and searches the Web Services according to the criteria of mobile client. In JXTA platform the class ProxyService fulfils the task to deal with messages. In this class with method `processIncomingMessage`() JXME proxy processes the incoming messages by the type of request and lets the respective method deal with the request. In this section we will cover how the JXME proxy handles with request to search Web Services in chosen peer group and the shadow and deep searching mechanism during the discovery process.

4.5.1 Processing message

As introduced in **section 4.4.2**, we add a new type of request message SearchInGroupRequest on the side of JXME midlet to satisfy our need of searching message in Group. There is a type of search request message of JXTA platform, but it can not serve our need to not only general search but search in some certain group of the catergorization structure. When the SearchInGroupRequest message is sent by the mobile client and arrived at JXME proxy, a matching method must also be written to deal with this SearchInGroupRequest, since such a method does not exist in the original JXTA platform. We write `handleSearchInGroupRequest()` to handle the searchInGroupRequest.

The method `handleSearchInGroupRequest()` at first extract the elements needed for searching the matching Web Services with the given value of Advertisement and chosen group. In this method three subfunctions `searchGroup()`,`searchMCAinGroup()`, `searchMSAbyMCA()` are called. Accordingly the whole searching task is fulfilled in three phrased.

4.5.2 Searching Web Service in Group implementation

This section describes implementation about how to search Web Service in peer group in three stages. At first the given peer group is to be found. After that the related advertisements (MCA) are searched in the found peer group. Then it goes on with the search for MSA(s) according to the information by the found MCA(s).

4.5.2.1 Searching peer group

In the first phrase the given group is searched. It starts with the comparison whether the searched group is mWSGroup, i.e. the parent group of all the other groups. If the answer is yes, then the search is ended and mWSGroup is returned. If the searched group is a child group of mWSGroup, then the method `searchGroupInGroup()` is

called in order to return the exact chosen peer group.

The search for the right group proceeds by depth-first search, i.e. child of a group is visited earlier than the neighbour of this group on the same level.

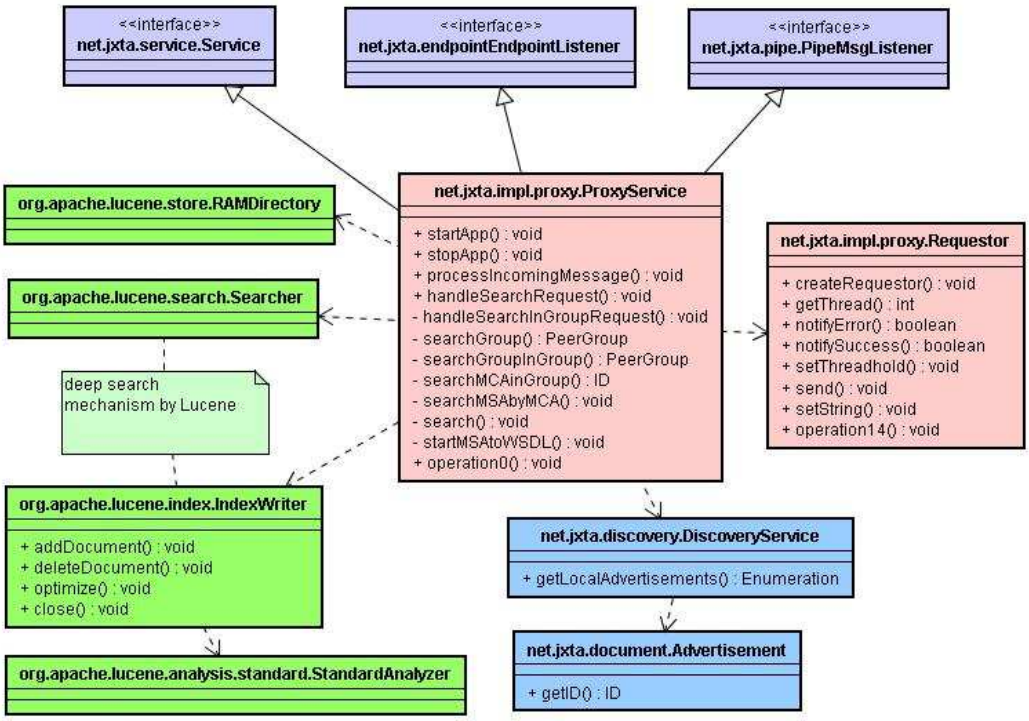


FIGURE 4-8 JXME PROXY CLASS DIAGRAM

4.5.2.2 Searching MCA(s) in peer group

In the second phrase Module Class Advertisement with the given value is to be searched in the found peer group from the first phrase. What we are really searching for is Module Spec Advertisements of the relevant Web Services since the description WSDL document is appended to MSA. But in order to make the searching process more efficient, we look for MCAs with the given value at first. Since each MCA relates to one or more MSAs, if we find some MCA which satisfies the given value, it means at once that we have found one or more MSAs with the given value. The advantage of the searching mechanism by way of MCA instead of direct search of MSAs is the simplified searching complexity of logarithms instead of linear cost. In a large P2P network this simplification is very meaningful to improve the scalability of the whole searching algorithm.

When no MCA meeting the prerequisites of mobile client is found, the search for MCA will go further into all the child groups of the given group and will not halt until some MCAs with the given value are found in some child group of given group or all the leave groups of the categorization tree are visited. . This is realized by recursively calling the method `searchMCAinGroup()` by depth-first search.

When some MCAs meeting the prerequisites of mobile client are found, they are

returned and the searching process goes to the next phrase.

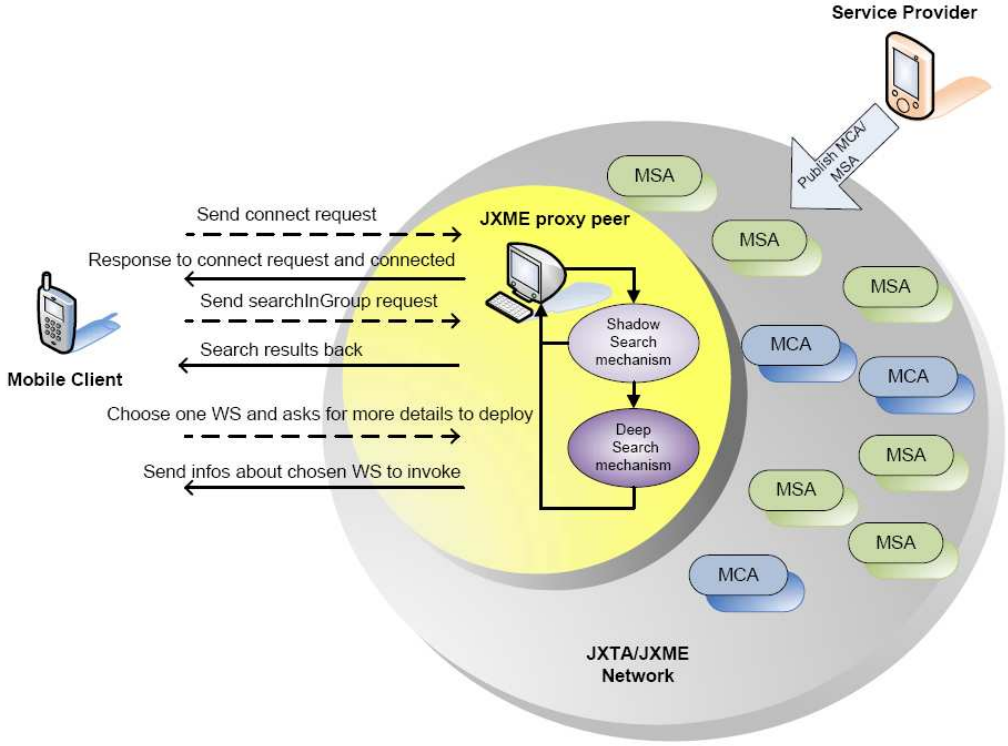


FIGURE 4-9 SEARCHING WEB SERVICES IN PEER GROUP OVERVIEW

4.5.2.3 Searching MSA(s) by MCA

In the third phrase Module Spec Advertisement are searched through MCA ID. The length of MCA is 49 digits. The MSA ID is created based on the corresponding MCA ID by deleting the last two digits and adding some different digits. Therefore, to search for MSA on the basis of its MCA we just need to delete the last two digits of MCA ID and add a wildcard symbol.

- **Shadow search implementation**

In the searchMSAInGroup message from mobile client an upper limit of number of MSAs to be sent back is set. Therefore the number of MSAs which will be finally sent back should not be bigger than this number. If the number of found MSAs is under the limit, the searching task is successfully finished and the found MSAs could be sent back to mobile client. We call this search process **shadow search**.

- **Deep search implementation**

When the number of found MSAs is over the set limt, the number of found MSAs should be further reduced by deep search mechanism of Lucene. The deep search is implemented in the earlier phrase of our project, therefore we will just provide a rough overview.

All the found MSAs are pushed into a document container. Lucene can hold the container in a storage device or in a random access memory RAM, An analyser for general purpose Standard Analyser is used in this project. Two parameters are needed to create a document. The first parameter the MSA ID and is got from Module Spec Advertisement. The second parameter is the content of WSDL document about Web Services which is appended to MSA in the element of param.

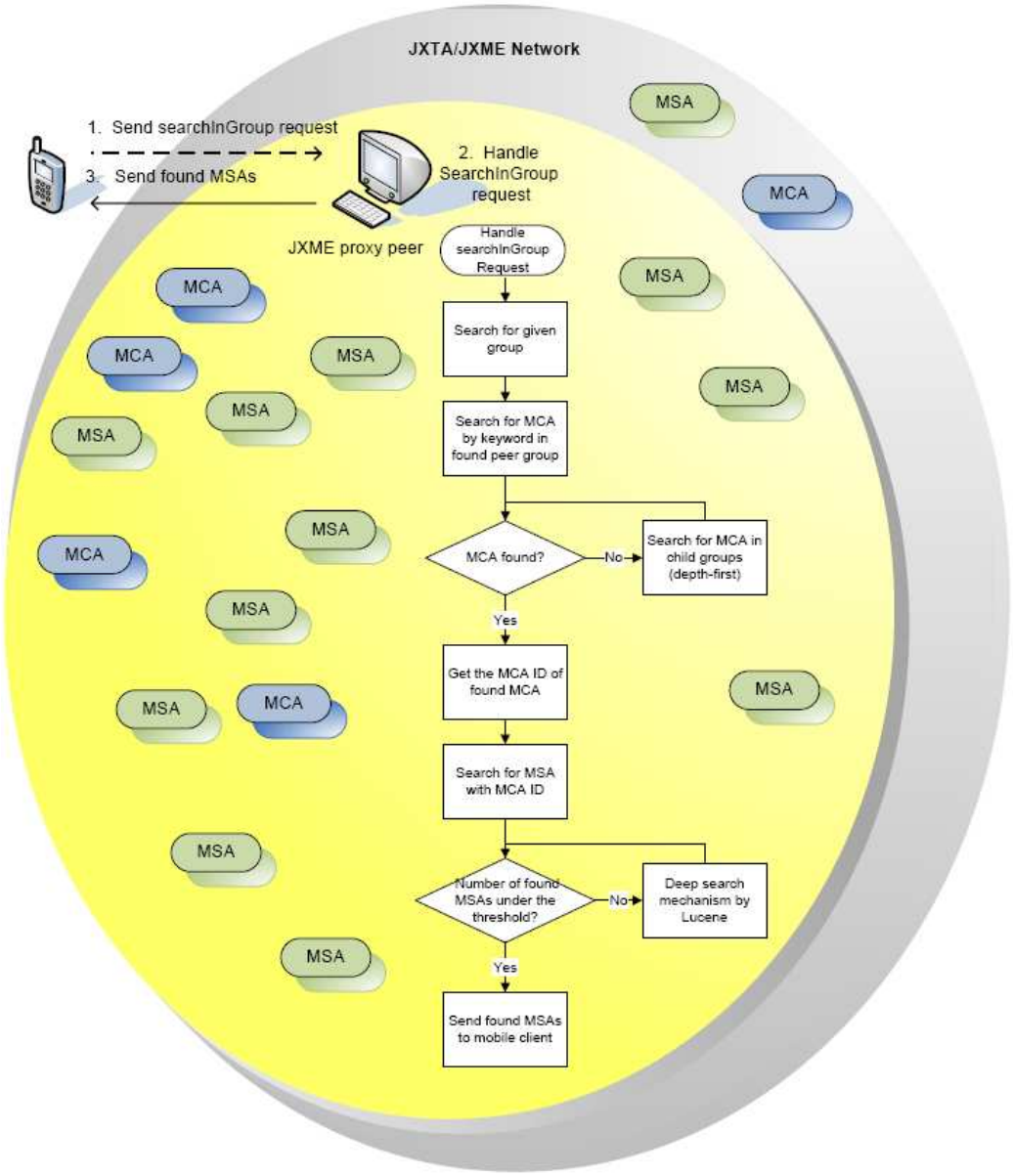


FIGURE 4-10 SEARCHING WEB SERVICES IN PEER GROUP FLOWCHART

4.6 Summary

From the points of view of participants in the publishing and discovery process, the implemented is in detail presented. For Web Service provider, it should publish its available Web Service into some chosen peer groups instead of into NetPeerGroup by default. A mobile Web Service requestor needs to select a peer group in addition to a keyword for searching. The JXME proxy peer process the incoming message of searchInGroup then accordingly by shadow and deep search mechanism.

5 Evaluation

For network performance scalability is an acknowledged desirable feature. The more heterogeneous a network is, the larger the scale of a network is, the more value and impact scalability has on the network traffic and the more meaningful is scalability for a network. In respect to P2P system, scalability is especially a most critical criterion for performance evaluation. In this section we are going to do research on the definition of scalability at first so as to properly denote the meaning of scalability on our project. Then by studying and comparing with related research we will set up the methodology of performance analysis for our projects. After that the result of experiments to test the scalability will be reported in order to study the characteristic of our P2P based discovery mechanism.

5.1 What is scalability?

Generally speaking, we say a system is scalable if it is able to accommodate an increasing number of elements or objects and/or to process growing volume of work gracefully. Gracefully here means with no dramatic overload. In other words, scalability is an attribute that the enlargement of a system does not affect the performance much compared to that of original system before enlargement.

Some defined scalability with two attributes: the ability to function well as it is changed in size or volume in order to meet a user need and the ability not only to function well in the rescaled situation, but to actually take full advantage of it [Scal04]. The second attribute is actually a further idealized improved quality based on the first attribute.

To evaluate the scalability of a system many aspects of factors must be taken into consideration. Different factors lead to different types of scalability to study. Scalability is under some circumstances up to the type of data structures and algorithms for implementation. The structures with extendable characteristics e.g. are entitled inherently with the better scalability than those of fixed size. This type of scalability is called **structural scalability**. Functions of a system are realized and influenced positively or negatively with the existence of data structures on the cost both of space and of time. From this aspect we could analyze the performance of a system by **space scalability** and **time scalability**. Sometimes inherent wastefulness due to frequently repeated actions of a system also lead to poor scalability The presence of access algorithms may result in suboptimal scheduling of resources too. In this case a system may runs well if the load is not heavy. Once the amount of load increases, its performance deteriorates dramatically. Load is therefore another consequential criterion for performance analysis and we call the system whose performance does not suffer much from the increase of load with **load scalability**.

In **Section 2.5.2**, the related work of scalability test of JXTA protocol is reviewed and JXTA discovery protocol is paid special attention. With those references it is time now to set the goals (**Section 5.2**) and design the appropriate methodology (**Section 5.3**) for the scalability test for the task in this thesis.

5.2 Goals of the Scalability Test of mobile Web Service Discovery Mechanism

The goals of evaluation phrase of this thesis especially the scalability test are trying to answer the following questions during the design and implementation phrase:

1. Is categorization discovery mechanism more **efficient** than non-categorization discovery?

2. If yes, **how efficient** is categorization discovery mechanism compared with non-categorization discovery?

3. Is discovery by implemented categorization mechanism **scalable**?

In order to answer the questions reasonably, a persuasive performance model for the test is necessary. The model should take the most probable related factors into consideration and include a detailed plan about single actions to proceed. To measure the performance an appropriate benchmark suit is also inevitable. Therefore, in the following research we have these goals:

- Develop a performance model and Benchmark Suite of non-categorization Discovery Mechanism

- Develop a performance model and Benchmark Suite of categorization Discovery Mechanism

- Conduct the tests according to the performance models and Benchmark Suite.

- Analyze the scalability of Discovery with categorization by comparing the results of Discovery with categorization with those of non-categorization Discovery.

To accomplish the goals we proceed the test in two phrases:

- Establish the respective performance model and benchmark suite for JXTA/JXME discovery mechanism with and without categorization.

- Interpret and analyze the performance results.

Section 5.3 contains details of the developments of performance model and benchmark suite. And **Section 5.4** covers the comparison, interpretation and analysis of performance results.

5.3 Methodology for Scalability Performance

5.3.1 Performance Model

To evaluate the performance of a complicated system like JXTA/JXME, we need to establish a model which covers all the relevant components. As introduced in **Section 2.4.3**, JXTA consists of six protocols. Among them **discovery protocol** interests us at most. In addition, different types of peers: Edge Peer, Relay Peer, Rendezvous Peer and JXME Proxy must be considered in our test. Because JXTA project is still under development and no standardized performance metrics are available, we take the above introduced projects as reference and conduct the experiment on the basis of peer operations in the process of discovery. With reference to related research the main metrics in our test are the **startup benchmark** for pre-discovery stage and **round-time trip benchmark** for discovery stage.

- **Startup benchmark in pre-discovery stage**

The minimum time a user has to wait for the initial response from the JXTA platform after starting an application is startup time. Startup time may be up to many factors. Platform configuration, size and location of the advertisement cache, the number and type of advertisement count as the primary factors. In addition, the peer configuration of rendezvous and/or relay settings may also affect startup time because of possible remote connections involved but this should not be counted as startup time, therefore it is not counted in the startup time. To make sure that only the JXTA platform JVM time is measured, the measurement of startup time should be inside a Java class.

We have presented in detail the implementation of discovery mechanism before. In order to observe the effect of categorization on discovery process, it is reasonable to also study the default discovery process of JXTA and compare this default discovery without categorization with our categorized discovery mechanism. The study consists of two stages: pre-discovery and discovery. Because the configurations of two different discovery mechanisms are to a large extent different from each other, it is reasonable to illustrate the respective happening by these two discovery mechanisms in order to observe the difference. To measure the difference we use a startup benchmark because it reflects the key time cost in pre-discovery stage.

The pre-discovery stage must be illustrated from the aspects of mobile WS requester, and WS provider and/or broker. In the simplest scenario where JXME Proxy/Relay peer acts also as WS provider, it has to conduct the actions in **FIGURE 5-1** to enable later discovery process possible:

- **Start JXTA**, Load platform classes, join default group of JXTA, i.e. Net Peer Group, open socket listener as Proxy peer and clear up local cache.
- **Publish WS,** create and publish an MCA and MSA to announce the availability of some WS.
- **Ready for Request**, Keep listener open to be ready to process incoming message request for WS search.

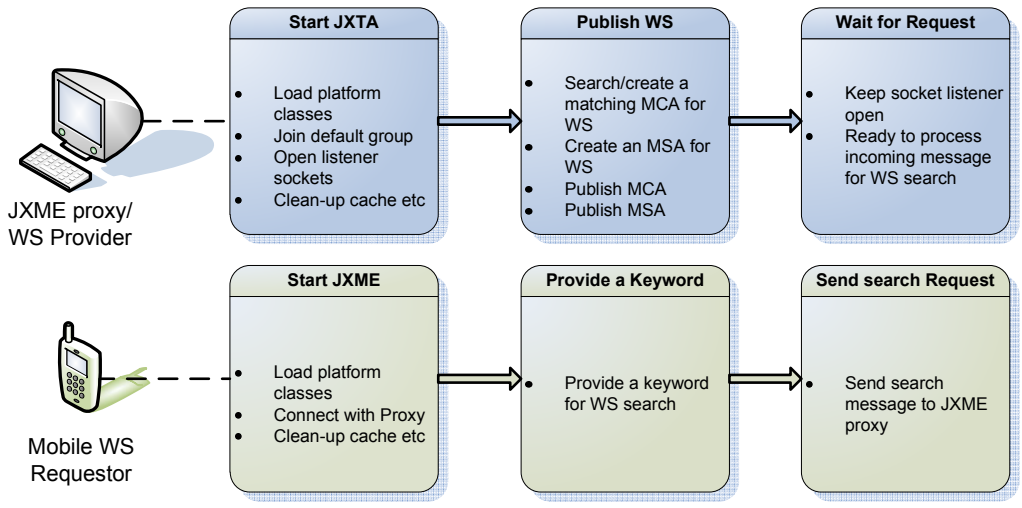


FIGURE 5-1 NON-CATEGORIZATION DISCOVERY MECHANISM OPERATIONS

Accordingly a WS requester conducts the following actions to search for a WS:

- **Start JXME**, load platform classes, connect with its Proxy and clear local cache
- **Provide a keyword** for WS search
- **Send search request** message to its Proxy peer.

In categorized discovery mechanism, one more act has to be taken both by WS provider (in the simplest scenario) and mobile WS requester, that is, to publish category information and to choose groups under categorization. For a WS provider it will create all the groups for categorization designed in **Section 3.2** (see **FIGURE 3-7**) and then publish WS in one or more appropriate group(s). For a mobile WS requester it has to choose a peer group in addition to a keyword for search request as shown in **FIGURE 5-2**.

To create groups for categorization and publish categories of peer groups is a critical peer operation. Therefore time cost to create and publish categories is to be measured and compared with default startup time of a Proxy Peer. Startup timestamp becomes then the Benchmark for pre-discovery stage.

- **Round-trip time benchmark in discovery stage**

Round-Trip Time (RTT) is the time it takes from sending a message by a sender to receiving acknowledgement (ACK) responded by a receiver. It is an elementary metric for a communication protocol. To receive ACK polling mechanism is used because pipes are asynchronous and unreliable. A polling timeout is also set to prevent a peer from waiting endlessly for an ACK which may be dropped on the way of message transfer. One factor which may affect RTT is overhead to compose and process message since JXTA protocols are in form of XML. Because of the TCP and HTTP transports underlying the pipes some overhead may also occur during message transmission.

By way of Message a mobile WS requester sends request for WS search and with the same way a WS provider responds to the request. A JXTA message is a unit of data transfer over the JXTA pipes. In discovery stage message Round-Trip Time is therefore usually taken as an elementary metric to evaluate peer operations and communication. In our test message RTT is measured on a mobile WS requester from the moment to send search request to its Proxy and to the time point when a response message is successfully received by polling messages within time interval.

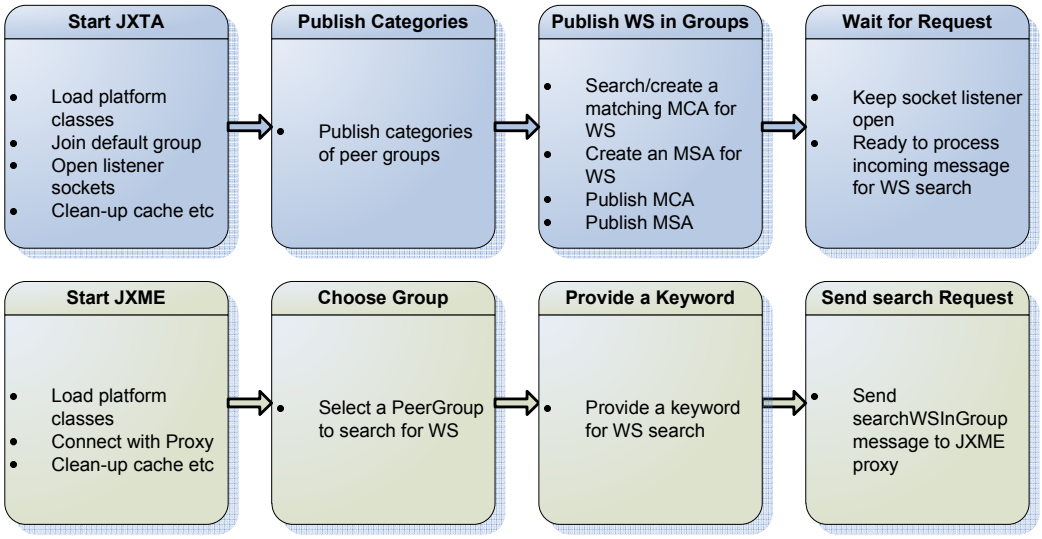


FIGURE 5-2 CATEGORIZATION DISCOVERY MECHANISM OPERATIONS

5.3.2 Benchmark Suite

According to the proposal by JXTA Benchmarking project, our discovery test should be conducted in these scenarios:

- one mobile Peer (JXME) and one Proxy/Relay Peer (JXTA)
- one mobile Peer (JXME) and one Proxy/Relay Peer (JXTA), the relay peer connects to one RDV (JXTA)
- one mobile Peer (JXME) and one Relay Peer (JXTA), the relay peer connects to more than one RDV (JXTA)

There are two versions for a JXME mobile peer to connect with JXTA peers: proxy-based and proxyless. Although proxyless version may take less latency in message transmission, no final stable version is available by the time of this research. So we still deploy the proxy-based version and connect JXME mobile peer with a JXTA peer who acts as JXME proxy. We are using the stable release JXTA 2.4.1. According to this release, a JXME proxy should be simultaneously a relay peer. So in the initial configuration, the JXTA peer which will play the role of JXME proxy should be set as both **JXME proxy** and a **relay Peer**.

With the intention to compare discovery performance with and without categorization, we design the discovery test with the following topologies:

1. **A. Non-categorization Discovery through relay peer**

   M<-> R

   On a mobile peer M, it measures the time it takes to discover a MSA. All Web Services are attached to advertisements and published into default group NetPeerGroup during the startup time. These advertisements about web services are saved in the local cache of the relay peer of mobile peer.

   B. Categorization Discovery through relay peer

   CM<-> CR

   On a categorized mobile peer CM, it measures the time it takes to discover a web service under the groups structure of categorization. WS are published in groups which are chosen by the service provider according to categorization criterion. During the startup time of JXTA framework the categorization structure is build up. Advertisements containing WS information are created and together with peer group advertisements by categorization are saved in the local cache of the relay peer.

2. **A. Non-categorization Discovery with one relay peer and one RDV peer**

   M <-> R<-> RDV

   On a mobile peer M, it measures the time it takes to find MSA(s) containing requested WS on the local cache of RDV by way of Relay peer R.

   B. Categorization Discovery with one relay peer and one RDV peer

   CM<-> CR<-> CRDV

   On a mobile peer CM, it measures the time it takes to find MSA(s) in the chosen peer group on CRDV by way of CR. Both CR and CRDV are categorized.

3. **A. Non-categorization Discovery with one relay peer and two RDV peers**

   M <-> R<-> RDV1<->RDV2

   On a mobile peer M, it measures the time it takes to find MSA(s) on RDV2 by way of R and RDV1.

   B. Categorization Discovery with one relay peer and two RDV peers

   CM<-> CR<-> CRDV1<-> CRDV2

On a categorized mobile peer CM, it measures the time it takes to find MSA(s) in the chosen peer group on CRDV2 by way of CR and CRDV1. All three peers CR, CRDV1 and CRDV2 are categorized.

4. A. Non- categorization Discovery with one relay peer and three RDV peers

M <-> R<-> RDV1<->RDV2<->RDV3

On a mobile peer M, it measures the time it takes to find MSA(s) on RDV3 by way of R, RDV1 and RDV2.

B. Categorization Discovery with relay peer and three RDV peers

CM<-> CR<-> CRDV1<-> CRDV2<-> CRDV3

On a categorized mobile peer CM, it measures the time it takes to find MSA(s) in the chosen peer group on CRDV3 by way of CR, CRDV1 and CRDV2. All four peers CR, CRDV1, CRDV2 and CRDV3 are categorized.

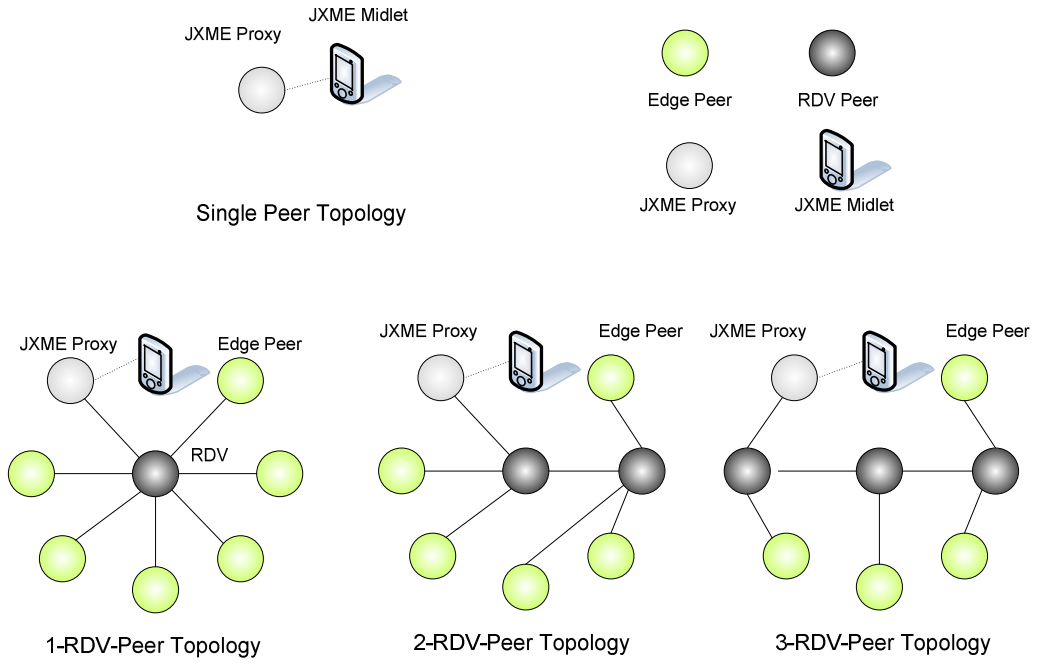


FIGURE 5-3 SCALABILITY TEST TOPOLOGIES

5.4 Performance Results and Analysis

This section presents the performance results drawn from the test. At first hardware, software as well as network environment for testing are introduced. Then we will present and analyze the performance in two sections: pre-discovery stage and discovery stage. Startup latency of JXTA framework is the main benchmark in pre-discovery stage. With the measurement we could have a clear mind about the time cost of building up the categorization structure and the usual time cost to publish WS in form of MSA. Discovery stage performance is of course our main focus in this section and is reviewed in order of four types of topologies planned in our benchmark suit. During the discussion, eye-catching phenomena observed from the tests in the four topologies are picked out and analyzed with attention. Otherwise the similarity and difference of the results from the different topologies are compared and discussed with the intention to get some satisfactory answers to the questions in **Section 5.3.**

5.4.1 Testing Environment

The network environment is campus 100 Mbps LAN at RWTH Aachen, Germany. The hardware environment consists of a Sony Ericsson P990i smart phone and a pool of eight computers. The phone has a memory of 60 MB and 3G technology with data transfer speed up to 384 kbps for Internet. The related configurations of the eight peers are given in **Table 5.1.** Two of them (Jxta 04 and Jxta05), each of which has 2GM RAM, are configurated as RDV peers in some topologies, because more powerful RAM often means better performance for applications and the efficiency is very needed for RDV peers. JXTA-JXSE version 2.4.1 is used to execute the test. Eclipse is the chosen developing framework and the JVM version is required to be above 1.5.0 for Eclipse SDK 3.2.

Name	IP Address	CPU	RAM	OS	Eclipse	JVM
Jxta01	137.226.232.157	PentiumIV,3.2GHz	1GB	Win XP Pro 2002	3.2	1.5.0
Jxta02	137.226.232.168	PentiumIV,3.2GHz	1GB	Win XP Pro 2002	3.2	1.6.0
Jxta03	137.226.232.103	PentiumIV,3.2GHz	1GB	Win XP Pro 2002	3.2	1.6.0
Jxta04	137.226.232.127	PentiumIV,3.2GHz	2GB	Win XP Pro 2002	3.2	1.6.0
Jxta05	137.226.232.153	PentiumIV,3.2GHz	2GB	Win XP Pro 2002	3.2	1.6.0
Jxta06	137.226.232.124	PentiumIV,3.2GHz	1GB	Win XP Pro 2002	3.2	1.6.0
Jxta07	137.226.232.172	PentiumIV,3.2GHz	1GB	Win XP Pro 2002	3.2	1.5.0
Jxta08	137.226.232.101	PentiumIV,3.2GHz	2GB	Win XP Pro 2002	3.2	1.6.0

Table 5-1 Configuration of JXTA test peers

According to our Benchmark Suite the eight computers in the pool are configured in JXTA framework according to four different topologies on the basis of the number of RDV peers (see FIGURE 5.5).

- Single Proxy/Relay Peer, i.e. no RDV peer is available.
- One RDV Peer R1 is available. The Proxy/Relay Peer, as well as all other six Edge Peers, set the exclusive RDV Peer as their seed RDV peer.

- Two RDV Peers R1 and R2 are available. The Proxy/Relay Peer sets R1 as its seed RDV, R1 sets R2 as its seed RDV. R1 and R2 each has a few Edge peers connected to them.

- Three RDV Peers R1, R2 and R3 are available. The Proxy/Relay Peer sets R1 as its seed RDV, R1 sets R2 as its seed RDV, R2 sets R3 as its seed RDV. R1, R2 and R3 each has a few Edge peers connected to them.

5.4.2 Pre-discovery stage Performance Results and analysis

According to performance model and benchmark suit we made, the purpose of measuring startup time is to tell the time cost of building up categorization and advertisements to propagate the availability of WS. Non-categorization and no advertisements version takes the shortest time of 21,261 ms. Categorization but no advertisements version takes 407,477 ms, i.e. to build up the peer group structures of categorization alone takes around 386,116 ms. To publish 100 advertisements in NetPeerGroup takes about 22,926 ms, while to publish 100 advertisements in chosen peer groups takes about 39,272 ms.

	Non-categorization	categorization
No Ads published	21,361	407,477
100 Ads published	44,287	446,749

Table 5-2 Startup latency (milliseconds)

The result of categorization latency is not a small number for our application. However, categorization is conducted on the JXTA peers which intend to publish Web Services. Since our main attention is the latency for a mobile peer to find a Web Service, the categorization latency will not lead to a critical concern.

5.4.3 Discovery stage Performance Results and Analysis

Four sets of tests are conducted with four different configurations planned in the Benchmark suite. The respective performance results are reviewed in each configuration and compared with each other.

5.4.3.1 Single Peer Topology – Results and Analysis

In this simplest topology, no RDV peer is available. The single JXTA peer fulfils all the tasks of building up categorization as well as creating advertisements and publishing them, when needed. We start the discussion by comparing the discovery performance of non-categorization and categorization mechanism to have a general view. Then we analyze each performance separately.

Non-Categorization vs. Categorization Discovery

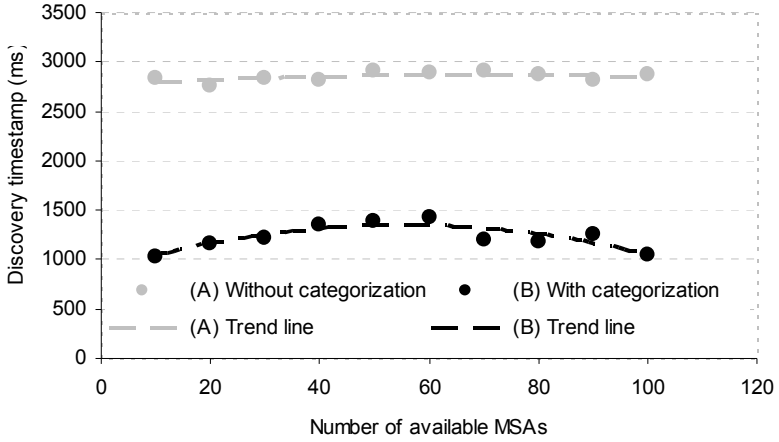


FIGURE 5-4 SINGLE PEER TOPOLOGY – COMPARISON

In the case of non-categorization discovery the test is comparatively easier to conduct. Since the only variable is the number of MSAs to be published, what the single peer needs to fulfil is to create and publish MSAs with number 10, 20… till 100. The discovery latency is measured in each case of different number of MSAs in the local cache of the single peer. In this way we get the points in (A) of **Figure 5.4.**

In categorization discovery the measurement is more complicated. Since four levels of peer groups are built in the category hierarchy and with the concern that the discovery latency may be rather diversified on different levels of category hierarchy, we think it is necessary to measure the performance from each level of hierarchy. With number of MSAs to be published as the variable, discovery latency is measured from groups of each of the four levels. The point in (B) of **Figure 5.4** is the mean of the four results form four groups on different levels with the same number of MSAs. The measurement on four levels of groups will be in detail presented next in **Figure 5.6** . What we would like to discuss now is the different trend line forms and complexity of non-categorization and categorization discovery mechanisms.

With no doubt the results in (B) with categorization shows less latency than (A) without categorization for a mobile peer to find a MSA. The first unanswered question in **section 5.2** has a positive answer in this topology. By comparing the mean of the points in (A) and (B) we could see that the time cost is reduced by about 50% with categorization.

What interest us further is the contrasting form of trend lines (A) and (B). Line (A) shows a very mild but obvious linear trend while line (B) runs in a near logarithm tendency. To have a better view of their respective form, we put them under loop as in **Figure 5.5** and **Figure 5.6** and discuss them one by one next.

Non-Categorization Discovery

The goal of test in the single peer topology is to measure the time it takes for a mobile WS requestor to find a WS in the default JXTA group, NetPeerGroup. As introduced in **Section 4.3.2.2**, an available WS in form of WSDL file is attached to a Module Specification Advertisement of a Peer which joins the JXTA Framework and published to the NetPeerGroup by the Peer. In the Single Peer scenario, the single peer acts as JXME Proxy and at the same time must be a Relay Peer according to the configuration requirement of JXTA-JXSE version 2.4.1. In addition, it must take the task of creating and publishing MSAs into NetPeerGroup serving to be searched later by mobile WS requestor.

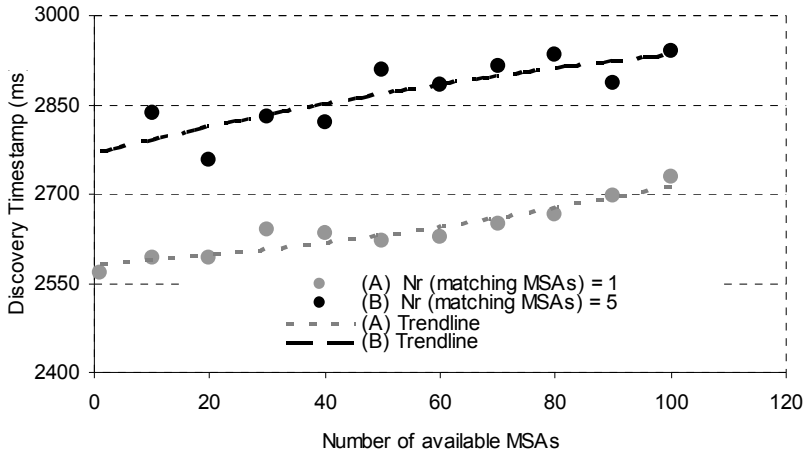


FIGURE 5-5 SINGLE PEER – NON-CATEGORIZATION DISCOVERY

Every time before the single peer creates and publishes MSAs, local cache is cleaned up so that no obsolete advertisement has any effect for the next round operation. MSA(s) is published at the number of 1, 10, and 20 up to 100 with an interval of 10 pieces between 10 and 100. Besides the number of available MSAs, the number of matching MSA(s) could also be a critical factor which affects the discovery time. With this assumption we measure the variable by the number 1 and 5. The number of 1 is taken since it is the least number of matching MSA for a successful discovery process. And the number of 5 matching MSA is an assumed average matching number regarding the total number of available MSAs of 100.

The points on **FIGURE 5-5** shows the time it takes to find 1 and/or 5 matching MSAs with 1 to 100 available MSAs. With the help of their respective polynomial function, we could interpret the trend of discovery time cost in single peer without categorization mechanism. Both lines show a growing trend of time cost with the growing number of available MSAs.

Categorization Discovery

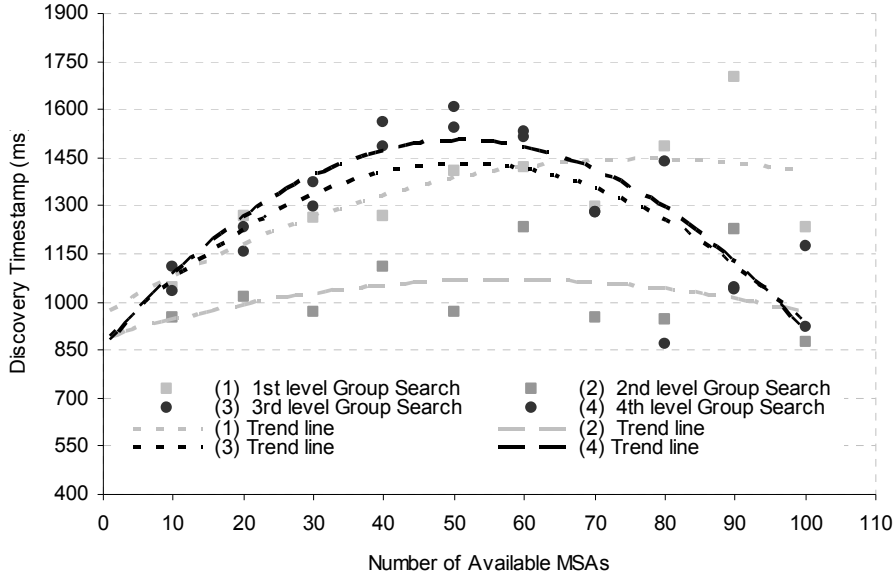


FIGURE 5-6: SINGLE PEER TOPOLOGY – CATEGORIZATION DISCOVERY

Discovery from the peer groups on different levels of categorization hierarchy shows verified performance. The search mechanism according to our implementation starts from the given peer group. The given peer group by mobile WS requestor has to be found at first before the search for MSA starts. The search could have to be conducted in several recursions if the given peer group is on the high level of the hierarchy. This explains on some sense why the latency is high for line (3) and (4). If no MSA is found in the given peer group, recursive depth-first search will be conducted in all the child groups of given peer groups until some MSA is found or till all the leaf groups are visited. This explains partially why the latency of line (1) is not as low as line (2). Line (1) shows the discovery latency by searching WS from the first level of hierarchy, i.e. mWSGroup. If no MSA is found in mWSGroup, then the search is conducted in the worst case in all the child groups, i.e., all the other peer groups in the categorization hierarchy.

5.4.3.2 Topology with One-RDV – Results and Analysis

In the one-RDV Topology, the JXME Proxy/Relay peer, as well as the other six edge peers set the single RDV peer as seed RDV by configurator. The RDV peer publishes MSAs in its local cache with the number between 10 and 100 with an interval of 10. The time is measured with the different number of available MSAs on the mobile peer to find matching MSA(s) from RDV. We start the discussion again with the comparison of two discovery mechanisms and then concentrate on the performance of each mechanism.

Non-Categorization vs. Categorization Discovery

From the trend lines (A) and (B) in **FIGURE 5-7** we could see that the fluctuation of the (A) is milder than that of single peer topology. Besides, the tendency of both categorization and non-categorization remains similar. The performance results of categorization discovery mechanism runs out a relative stable constant trend line. The mean of (B) is 2941 ms, in comparison to 4655 ms of (A) and reduces the time cost by about 37%. Therefore, in one-RDV topology categorization mechanism is still more efficient than non-categorization mechanism.

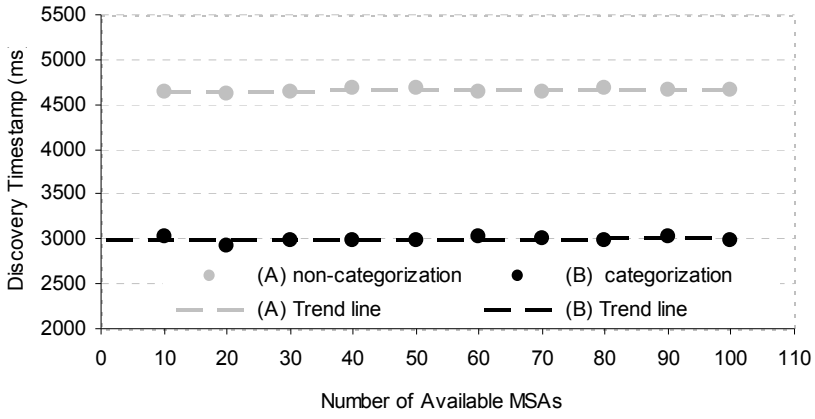


FIGURE 5-7 TOPOLOGY WITH ONE-RDV – COMPARISON

Non-categorization Discovery

In order to observe the developing tendency of discovery performance with a more detailed view, we put the same results in a diagram with larger interval for timestamps on y axis as in **FIGURE 5-8**. The points on Line (A) in **FIGURE 5-8** and **FIGURE 5-7** has exactly the same data resource. In **FIGURE 5-8** we could read more clearly about the trend that the time cost grows steadily with the growth of total available number of MSAs on the RDV peer.

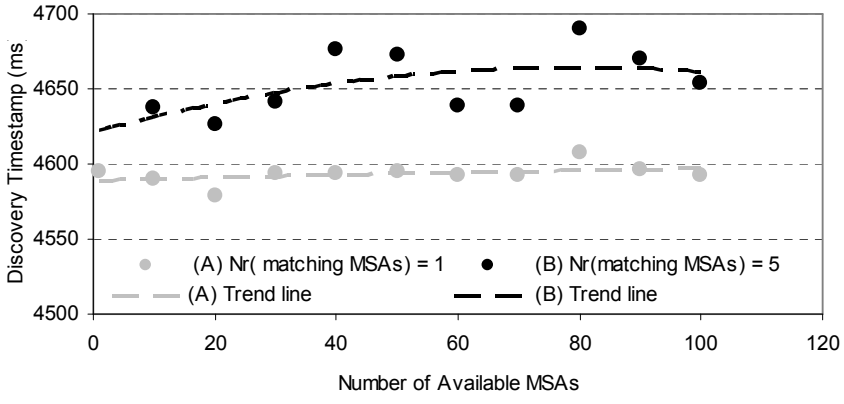


FIGURE 5-8 TOPOLOGY WITH ONE-RDV – NON-CATEGORIZATON DISCOVERY

Categorization Discovery

The main difference of discovery behaviour in 1-RDV topology from single peer topology is that the principal function of remote discovery of advertisement instead of solely searching advertisements in local cache. With the given peer group by mobile WS requestor, the search is conducted **only** in this peer group instead of the exhaustive searching behaviour in all its child groups no matter MSA is found or not. If no matching MSA is found in the local peer group on the JXME proxy/relay peer, the search runs then in the same peer group on the remote RDV peer. Therefore, the recursive search for child peer groups and for advertisements in the groups is not deployed in the test of 1-RDV topology. It explains why the discovery time in (3) and (4) are much less fluctuated than those in **FIGURE 5-6.**

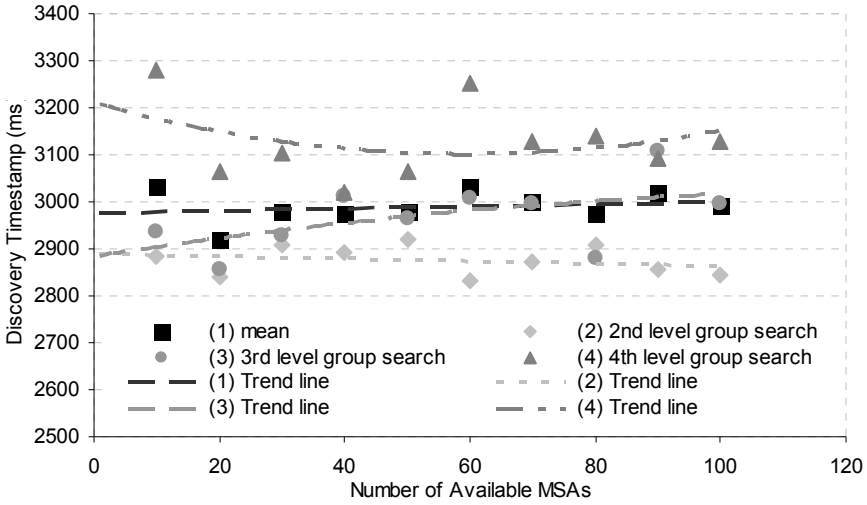


FIGURE 5-9 TOPOLOGY WITH ONE-RDV –CATEGORIZATON DISCOVERY

5.4.3.3 Topology With Two-RDV and Three-RDV – Results and Analysis

By comparing the trend lines of discovery timestamp with categorization in topology with 2-RDV (**FIGURE 5-12)** and in topology with 3-RDV (**FIGURE 5-13)**, we could see that the one or two more RDV on the route of discovery bring no dramatic effect to the discovery time and trend. The results from 2-RDV and 3-RDV topology are very much alike to 1-RDV topology. The non-categorization discovery approach keeps the tendency of mild linear growth, while categorization discovery approach leads to almost constant discovery time hardly affected by the growing number of available MSAs in peer groups.

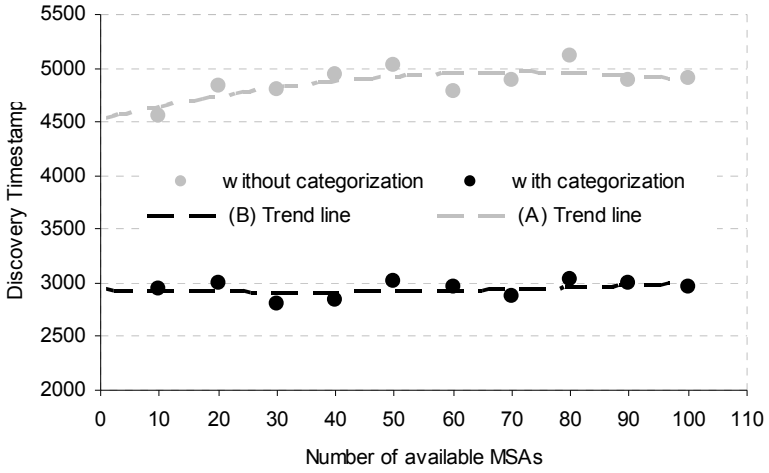


FIGURE 5-10 TOPOLOGY WITH TWO-RDV – COMPARISON

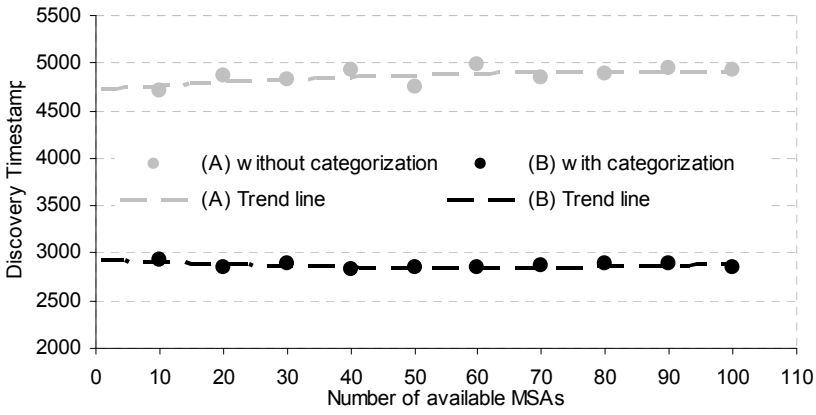


FIGURE 5-11 TOPOLOGY WITH THREE-RDV – COMPARISON

5.4.3.4 Comparison between Difference Topologies

In order to achieve an overview of performance results in different scenarios, we compare the mean discovery time to find Web Services in the cases of four topologies where five matching MSAs are available as search result. The result in **FIGURE 5-12** indicates that the use of categorization in discovery process obviously improved the performance by 40% to 50% in average. With the addition of more RDV peers, the discovery time of non-categorization version grows lightly, which could lead to undesirable scalable property in large-scale network. In contrast the discovery time with categorization mechanism does not grow much, if at all, with the addition of more RDV peers. Although it is just a small-scale test result, we could still tell from different trends of performance that discovery with categorization actually could improve the

performance in mobile Web Service discovery and with categorization the performance is more scalable.

According to the trend analysis from the small-scale test it could be reasonably assumed that the discovery mechanism with categorization will keep the property of scalability even when the scale of the network grows up to some extent. The design of discovery mechanism with categorization then could very probably bring scalable performance even when under the situation of large-scale network.

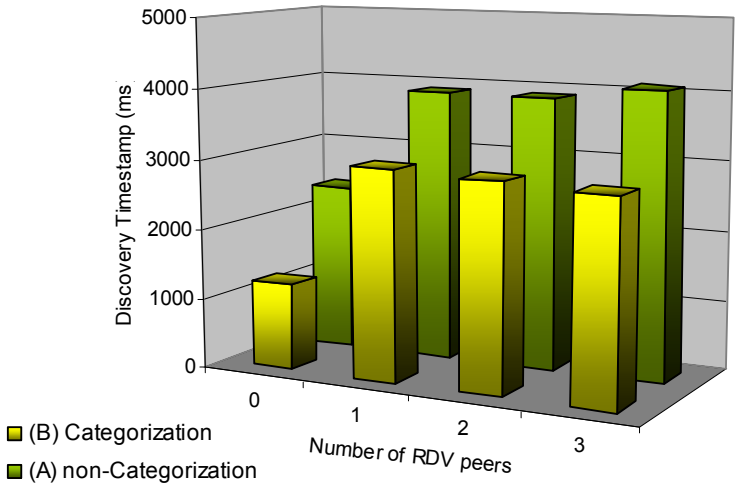


FIGURE 5-12 PERFORMANCE ANALYSIS OF ALL FOUR TOPOLOGIES

5.5 Summary

Scalability is a highly desirable property for network performance. By studying the definition of scalability and related projects about scalability test of JXTA/JXME project, we design a performance model and Benchmark suite for our test. The performance result in different topologies indicate that discovery with categorization mechanism could in fact improve the performance and is more scalable than non-categorization mechanism.

6 Conclusion and Future Work

The main outcome of this thesis is the p2p based mobile Web Service discovery through categorization mechanism. This section summarizes the design, implementation and evaluation of the P2P based mobile Web Service discovery mechanism, reviews encountered issues and discusses perspectives for future research of P2P based mobile Web Service provisioning and consumption.

6.1 Conclusion

Due to limited resource of mobile peers mobile applications usually follow the discover, lease, deploy and discard model when they need some Web Service. To deploy application is the purpose, but to discover applications which meet certain functional criteria is the first and a very crucial stage of the whole process. Web Services Discovery takes place in the first stage of the whole web services process and discovery is the therefore prerequisite for a functional web service process and is the key focus of our project research.

UDDI is normally used for Web Service discovery. But this mechanism on the basis of central registry does not suit the flexible and dynamic behaviour of mobile peers. P2P provides an alternative that does not rely on centralized registries; rather it allows Web services to discover each other dynamically. The P2P based JXTA/JXME is chosen as framework to implement mobile Web Service discovery mechanism.

On the basis of the past study of mobile Web Service discovery, this diploma thesis intends to design and implement a more efficient discovery mechanism by borrowing some good characteristics of UDDI such as categorization. With the reference of some popular industry categorization standards we build up a categorization hierarchy. It starts with the root group mobile Web Service Group and consists of peer groups of further three levels hierarchically. The group structure which we build is a first draft to realize the idea of categorization. Neither have we the ambition to build up a complete categorization hierarchy nor do we intend to include all possible existing mobile Web Services into this hierarchy because those are not our intension.

In JXTA/JXME Framework, this structure is initiated by a shell command “category” self-defined and implemented by us. If the user wants to deploy the categorization mechanism, all s/he needs to do is type the command on the JXTA shell. Web Service provider could choose suitable peer groups to publish the advertisement about Web Service in one or more peer groups. When a mobile Web Service requestor searches for a Web Service, s/he could choose a suitable peer group in addition to keyword for search. Categorization in form of peer groups in JXTA is supposed to reduce the searching scope efficiently.

In order to test the scalability of the categorization mechanism, we take related research as reference and make a practical performance model and benchmark suite with limited resource available to us. Since the large scale test on multi-site distributed tested is not realizable, we take eight peers in computer pool of the institute, which could be configured for our need, as test sample and conduct the scalability test in small scale. The results from the test denote that the categorization mechanism works more efficiently than non-categorization mechanism. The time cost could be reduced by 40 to 50 percent by deploying categorization mechanism for discovery. With

different topologies of the configuration for the test peers we intend to know how the growing number of RDV peer affects performance. With up to three RDV peers the discovery performance shows similar developing trends and is not affected much. We interpret the result as the property of being scalable for the categorization mechanism in the initial small scale test. It could further be inferred that with high probability that the scalable property of the designed mechanism in this thesis will be kept in even larger scale network.

6.2 Future Work

The main focus of this thesis is to improve performance efficiency of in the stage of mobile Web Service discovery. The goal is partially reached by borrowing the idea of categorization from UDDI. It is planned in the projects to compare the performance of UDDI approach and the P2P approach in which both have categorization paradigm. However, invocation of the found Web Service is still to be implemented in order to do the comparison between UDDI and the P2P approach realized in this thesis. The function of CategoryPack, which contains categorization details according to widely acknowledged industry standards, is not fully used in this thesis. It could be further deployed in future research.

Since the test is based on a small scale sample of advertisements and peers, the present searching mechanism is not much overburdened. But in large scale network and large amount of advertisements the searching mechanism is not sufficient. The performance could be improved by using advanced searching mechanism such as advanced features of Lucene. In addition, the search mechanism could be extended by context awareness. The search request from the client could be stored in a middleware device. When a user makes a new request those stored information could help improve the search results.

Proxyless version is preferred to proxy-based version in respect to mobile peer. With the ongoing of JXTA-JXME project it is expected that the stable proxyless version is developed soon. In JXTA-SOAP[JSOA07] project it is found that kXML should be more appropriate to invocate Web Services for mobile peers with JXME.

Appendix A. Category in UDDI

A-1 Simple Categorization

```
<businessEntity businessKey="uddi:my_company.example">
    …
    <categoryBag>
        <keyedReference
            tModelKey="uddi:uddi.org:ubr:categorization:unspsc"
            keyName="UNSPSC:Medical Equipment and Accessories and Supplies"
            keyValue="42.00.00.00.00"/>
        <keyedReference
            tModelKey="uddi:uddi.org:ubr:categorization:unspsc"
            keyName="UNSPSC:Drugs and Pharmaceutical Products"
            keyValue="51.00.00.00.00"/>
        <keyedReference
            tModelKey="uddi:uddi.org:ubr:categorization:iso3166"
            keyName="GEO:Germany"
            keyValue="DE"/>
        <keyedReference
            tModelKey="uddi:uddi.org:ubr:categorization:iso3166"
            keyName="GEO:France"
            keyValue="FR"/>
        <keyedReference
            tModelKey="uddi:uddi.org:ubr:categorization:iso3166"
            keyName="GEO:United States"
            keyValue="US"/>
    </categoryBag>
</businessEntity>
```

A-2 Group Categorization

```
<businessEntity businessKey="uddi:my_company.example">
    …
    <categoryBag>
        <keyedReferenceGroup
            tModelKey=
                "uddi:uddi.org:ubr:categorizationgroup:unspsc_geo3166">
            <keyedReference
                tModelKey="uddi:uddi.org:ubr:categorization:unspsc"
                keyName="UNSPSC:Medical Equipment and Accessories and Supplies"
                keyValue="42.00.00.00.00"/>
            <keyedReference
                tModelKey="uddi:uddi.org:ubr:categorization:iso3166"
                keyName="GEO:Germany"
                keyValue="DE"/>
        </keyedReferenceGroup>
        <keyedReferenceGroup
            tModelKey=
                "uddi:uddi.org:ubr:categorizationgroup:unspsc_geo3166">
            <keyedReference
                tModelKey="uddi:uddi.org:ubr:categorization:unspsc"
                keyName="UNSPSC:Medical Equipment and Accessories and Supplies"
                keyValue="42.00.00.00.00"/>
            <keyedReference
                tModelKey="uddi:uddi.org:ubr:categorization:iso3166"
                keyName="GEO:France"
                keyValue="FR"/>
        </keyedReferenceGroup>
        <keyedReferenceGroup
            tModelKey=
                "uddi:uddi.org:ubr:categorization:unspsc_geo3166">
            <keyedReference
                tModelKey="uddi:uddi.org:ubr:categorization:unspsc"
                keyName="UNSPSC:Drugs and Pharmaceutical Products"
                keyValue="51.00.00.00.00"/>
            <keyedReference
                tModelKey="uddi:uddi.org:ubr:categorization:iso3166"
                keyName="GEO:United States"
                keyValue="US"/>
        </keyedReferenceGroup>
        …
    </categoryBag>
</businessEntity>
```

Appendix B. Modules in JXTA

B-1 Declare existence of a Web Service by MCA

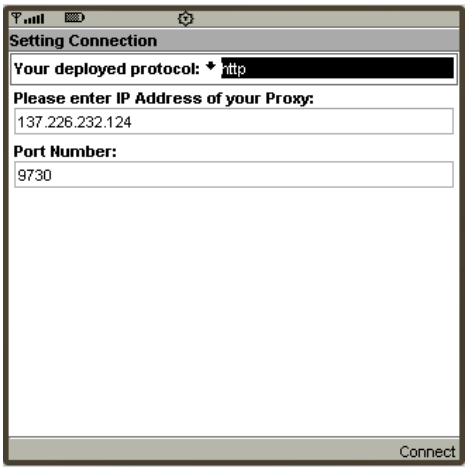
```
<?xml version=”1.0” encoding=”UTF-8”?>
<!DOCTYPE jxta:MCA>
<jxta:MCA xmlns:jxta=”http://jxta.org”>
        <MCID>
                urn:jxta:uuid-E28CE82813DC49C8B53EE5B679FF900505
        </MCID>
        <Name>
                exchange
        </Name>
        <Desc>
                currency exchange service
        </Desc>
</jxta:MCA>
```

B-2 Describe features of a Web Service by MSA

```
<?xml version=”1.0” encoding=”UTF-8”?>
<!DOCTYPE jxta:MSA>
<jxta:MSA xmlns:jxta=”http://jxta.org”>
    <MSID>
    urn:jxta:uuid-E28CE82813DC49C8B53EE5B679FF90054646F2ACD0A64D8E9A19A405B32
BD88506
    </MSID>
    <Name>
        currencyExchange
    </Name>
    <Desc>
        foreign currency exchange service
    </Desc>
    <Crtr>
        Hongyan Zhu
    </Crtr>
    <SURI>
        http://www.hongyanzhu.org
    </SURI>
    <Vers>
        WSDL Version 1.2
    </Vers>
    <jxta :PipeAdvertisement xmlns :jxta= »http ://jxta.org »>
        <Id>
    urn :jxta :uuid-59616261646162614A787461503250336398086O3ECD42B5B3F3F1D8D
12BC49F03
        </Id>
        <Type>
            JxtaUnicast
        </Type>
        <Name>
            Hongyan Unicast Pipe Advertisement
        </Name>
    </jxta:PipeAdvertisement>
    <Parm>
        <WSDL>
            …

        </WSDL>
    </Parm>
</jxta:MSA>
```

Appendix C. Screenshots of mobile Web Service requestor



Screenshot C-1: setting connection

Screenshot C-2: confirming connection

Screenshot C-3: root peer group - mWSGroup

Screenshot C-1: setting connection

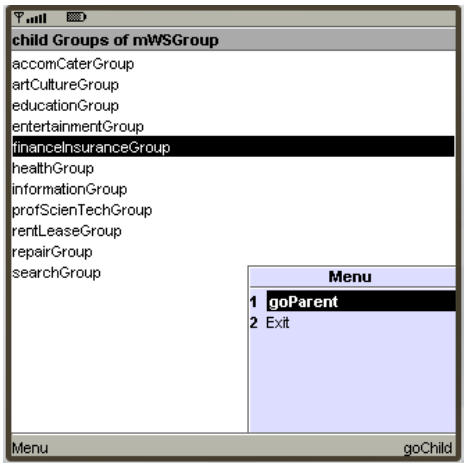
- Provide protocol zu deploy
- Provide IP Address of JXME proxy
- Provide Port Number of JXME proxy

Screenshot C-2: confirming connection

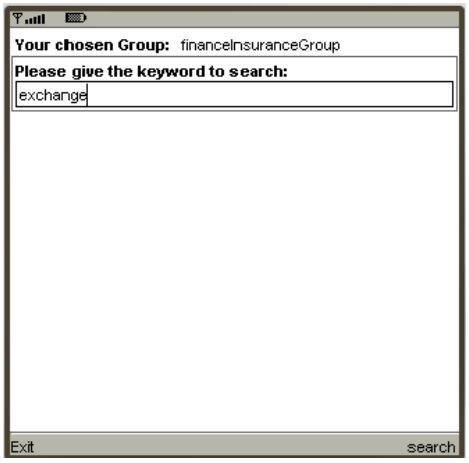
- Connect with JXME proxy
- Confirm connection after successful connection

Screenshot C-3: root peer group

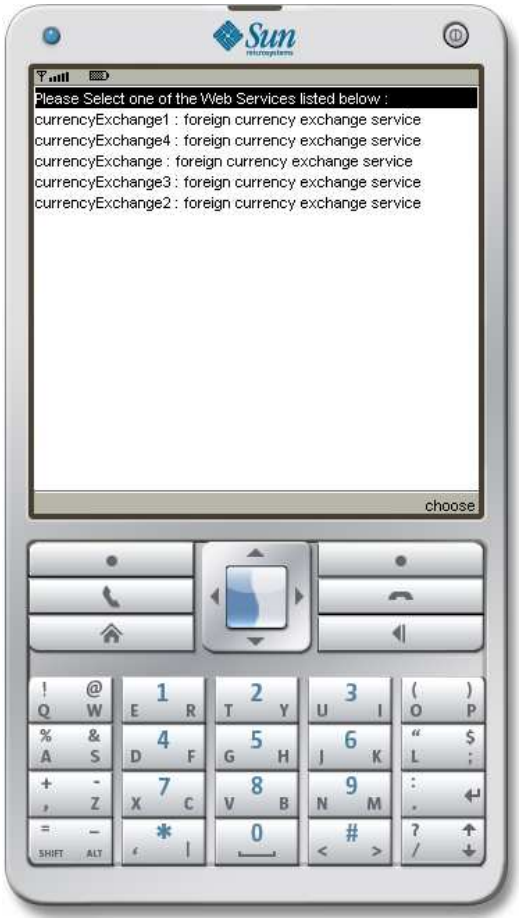
- Go to root peer group – mWSGroup
- Choose child group by goChild command



Screenshot C-4: selecting peer group



Screenshot C-5: providing a keyword to search



Screenshot C-6: getting search result(s)

Screenshot C-4: selecting peer group

- Select parent group by goParent command
- Select child group by goChild command

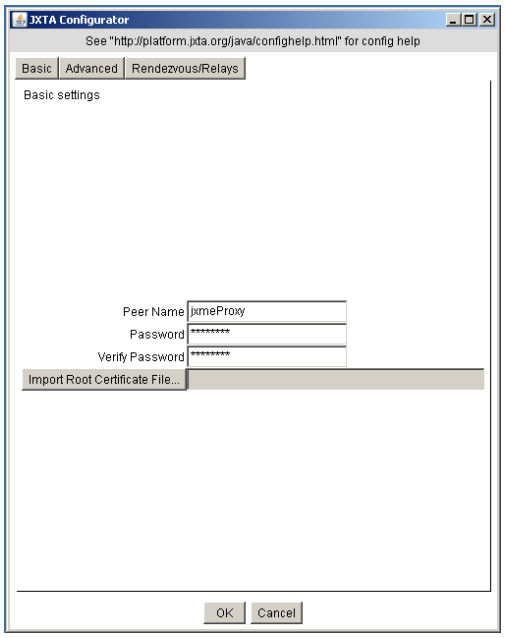
Screenshot C-5: providing a keyword

- Provide a keyword to search
- Construct search criterion composed of keyword and selected peer group
- Send Search message to JXME proxy by search command

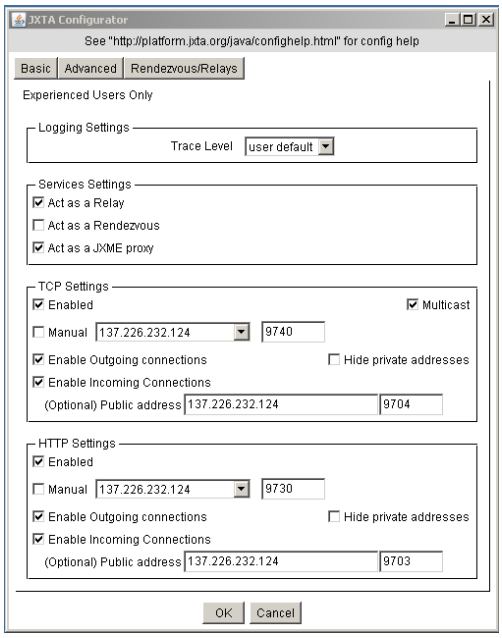
Screenshot C-6: getting search result(s)

- Get one or more search results returned from JXME proxy
- Select a Web Service to deploy by choose command

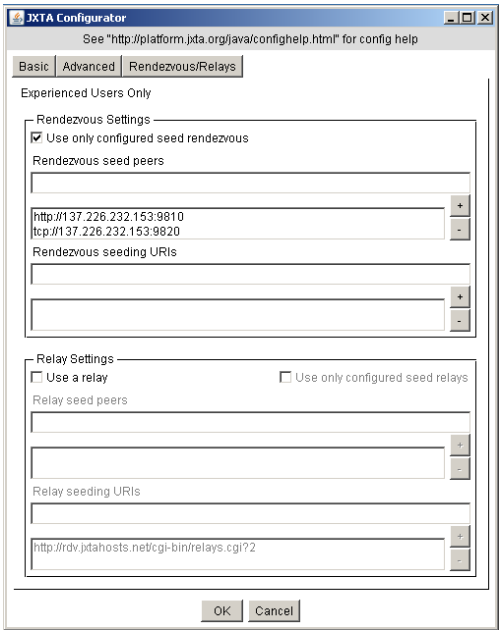
Appendix D. Screenshots of JXTA configurator



Screenshot D-1: Basic Settings



Screenshot D-2: Advanced Settings



Screenshot D-3: RDV/Relay Settings

Screenshot D-1: Basic Settings

- Set peer name (compulsory)
- Set password (compulsory)

Screenshot D-2: Advanced Settings

- Set services (selective)
- Set address and port number for TCP if needed
- Set address and port number for HTTP if needed

Screenshot D-3: RDV/Relay Settings

- Set RDV(s) (selective)
- Set Relay(s) (selective)

LIST OF REFERENCES

[ABGP01] R.AGRAWAL, R.BAYARDO, D.GRUHL, S.PAPADIMITRIOU: VINCI: A SERVICE-ORIENTED ARCHITECTURE FOR RAPID DEVELOPMENT OF WEB APPLICATIONS, IN PROCEEDINGS OF THE 10TH INTERNATIONAL CONFERENCE ON WORLD WIDE WEB (HONG KONG, HONG KONG, MAY 01 – 05, 2001), WWW '01, ACM PRESS, NEW YORK, NY, 355-365

[ACDJ06] G.ANTONIU, L.CUDENNEC, M.DUIGOU, M.JAN: PERFORMANCE SCALABILITY OF THE JXTA P2P FRAMEWORK, PARALLEL AND DISTRIBUTED PROCESSING SYMPOSIUM, 2007. IPDPS 2007. IEEE INTERNATIONAL

[APAC05] THE APACHE SOFTWARE FOUNDATION: AXIS PROJECT, AVAILABLE AT: HTTP://WS.APACHE.ORG/AXIS/

[BACH00] P.BAILEY, N.CRASWELL AND D.HAWKING: DARK MATTER ON THE WEB. IN POSTER PROCEEDINGS, 9TH WORLD-WIDE WEB CONFERENCE, 2000

[BHMN04] BOOTH, H.HAAS, F.MCCABE, E.NEWCOMER, W3C WORKING GROUP NOTE, FEB. 2004, AVAILABLE AT: HTTP://WWW.W3.ORG/TR/WS-ARCH/

[BMCM04] BROSN, T.MAITRAT, A.COLHOUN, B. MACARDLE, AVAILABLE AT: HTTP://NTRG.CS.TCD.IE/UNDERGRAD/4BA2.02-03/INTRO.HTML

[BPEL05] WHAT IS BPEL? AVAILABLE AT: HTTP://SEARCHWEBSERVICES.TECHTARGET.COM/SDEFINITION/0,290660,SID26_GCI845110,00.HTML

[CALS99] UPNP WHITEPAPER : UPNP,JINI AND SALUTATION – A LOOK AT SOME POPULAR COORDINATION FRAMEWORKS FOR FUTURE NETWORKED DEVICES. AVAILABLE AT: HTTP://WWW.CALSOFTLABS.COM/WHITEPAPERS/UPNP-DEVICES.HTML

[CCDD05] F. CAPPELLO, E. CARON, M. DAYDE, F. DESPREZ ET AL: GRID'5000: A LARGE SCALE AND HIGHLY RECONFIGURABLE GRID EXPERIMENTAL TESTBED, GRID COMPUTING WORKSHOP, 2005

[CZM05] P. CONTRERAS, D. ZERVAS, F. MURTAGH: WEB SERVICES IN NATURAL LANGUAGE: TOWARDS AN INTEGRATION OF WEB SERVICE AND SEMANTIC WEB STANDARDS IN THE JXTA PEER TO PEER ENVIRONMENT, MAY 2005, AVAILABLE AT: HTTP://THAMES.CS.RHUL.AC.UK/~WSTALK/PAPERS/RHULPAPERS/ORCHESTRATION.PDF

[CERA02] E.CERAMI: WEB SERVICES ESSENTIALS. AVAILABLE AT : HTTP://WWW.DEVELOPER.COM/SERVICES/ARTICLE.PHP/10928_1602051_1

[DEJO04] M. DE JODE: PROGRAMMING JAVA 2 MICRO EDITION ON SYMBIAN OS, A DEVELOPER'S GUIDE TO MIDP 2.0, SYMBIAN PRESS, 2004

[DOYP02] D. M. DOOLIN, O. YEUNG, K.S. PABLA: JXTA P2P PLATFORM BENCHMARK PLAN. AVAILABLE AT: HTTPS://JXTA-BENCHMARKING.DEV.JAVA.NET/BENCHPLAN.HTML

[DUTR06] S. DUSTDAR, M. TREIBER: INTEGRATION OF TRANSIENT WEB SERVICES INTO A VIRTUAL PEER-TO-PEER WEB SERVICE REGISTRY, DISTRIB PARALLEL DATABASES, 2006

[ELEN03] ELENIUS: SERVICE DISCOVERY IN PEER TO PEER NETWORKS, SEPT. 2003, AVAILABLE AT: HTTP://WWW.IDA.LIU.SE/~DAELE/EXJOBB/

[GRAD05] J. GRADECKI: MASTERING JXTA: BUILDING JXTA PEER-TO-PEER APPLICATIONS, 2005

[GROS04] D. GROSS: BUY CELL – HOW MANY MOBILE PHONES DOES THE WORLD NEED? AVAILABLE AT: HTTP://WWW.SLATE.COM/ID/2101625

[GKLS00] S. GRAUPNER, W. KIM, D. LENKOV, A. SAHAI: E-SPEARK – AN ENABLING INFRASTRUCTUREFOR WEB-BASED E-SERVICES, HEWLETT-PACKARD CORPORATION E-SPEAK OPERATION, 2000

[HADE03] E. HALEPOVIC, R. DETERS: THE COST OF USING JXTA, PROCEEDINGS OF THE THIRD INTERNATIONAL CONFERENCE ON PEER-TO-PEER COMPUTING, 2003

[HAJA03] S. HAJAMOHIDEEN: A MODEL FOR WEB SERVICE AND INVOCATION IN JXTA, 2003 AVAILABE AT: HTTP://WWW.TI5.TU-HARBURG.DE/PUBLICATION/2003/THESIS/HAJA03/HAJA03.PDF

[HALL06] M. HALLING-BROWN: COMPUTATIONAL TECHNIQUES FOR THE PREDICTION MINOR HISTOCOMPATABILITY AND T CELL ANTIGENS, AVAILABLE AT: HTTP://IGRID-EXT.CRYST.BBK.AC.UK/WWW/PHD_THESIS.HTM

[HYAR04] E. HARJULA, M. YLIANTTILA, J. ALA-KURKKA, J. RIEKKI, PLUG-AND-PLAY APPLICATION PLATFORM: TOWARDS MOBILE PEER-TO-PEER, OKT. 2004, AVAILABLE AT: HTTP://WWW.MEDIATEAM.OULU.FI/PUBLICATIONS/PDF/570.PDF

[JANU02] K. JANUSZEWSKI: THE IMPORTANCE OF METADA: REIFICATION, CATEGORIZATION, AND UDDI, SEP. 2002, AVAILABLE AT: HTTP://MSDN2.MICROSOFT.COM/EN-US/LIBRARY/MS953942.ASPX#IMPMETADATA_TOPIC6

[JSOA07] JXTA-SOAP PROJECT, AVAILABLE AT HTTPS://SOAP.DEV.JAVA.NET/

[JUWR99] M. B.JURIC, T.WELZER, T.ROZMAN, ET AL, "PERFORMANCE ASSESSMENT FRAMEWORK FOR DISTRIBUTED OBJECT ARCHITECTURES," *ADVANCED IN DATABASES AND INFORMATION SYSTEMS*, VOL. 1691, PP. 349-366, 1999

[JXME05] JXTA-JXME PROJECT. AVAILABLE AT: HTTPS://JXTA-JXME.DEV.JAVA.NET/

[JXTA02] K. WILSON: JXTA, 2004, NEW RIDER'S PUBLISHING, AVAILABLE AT HTTP://WWW.BRENDONWILSON.COM/PROJECTS/JXTA-BOOK/

[JXTA04A] JXTA TECHNOLOGY: CREATING CONNECTED COMMUNITIES, 2004, AVAILABLE AT HTTP://WWW.JXTA.ORG/DOCS/JXTA-EXEC-BRIEF.PDF

[JXTA04B] WHAT IS JXTA? AVAILABLE AT: HTTP://SEARCHSOA.TECHTARGET.COM/SDEFINITION/0,,SID26_GCI778096,00.HTML

[JXTA05] JXTA V2.3X : JAVA PROGRAMMER'S GUIDE, APR. 2005, AVAILABLE AT HTTP://WWW.JXTA.ORG/DOCS/JXTAPROGGUIDE_V2.3.PDF

[JXTA07] JXTA V 2.5 PROGRAMMER'S GUIDE, SEPT, 2007, AVAILABLE AT HTTP://WWW.LULU.COM/INCLUDES/DOWNLOAD.PHP?FCID=1197206&FMID=104608

[LAGI99] S.LAWRENCE AND C. L.GILES. ACCESSIBILITY OF INFORMATION ON THE WEB. NATURE, 400: 107-109, JULY 1999

[LHDV03] C.LEE, A. HELAL, N.DESAI, V.VERMA ET AL: KONARK: A SYSTEM AND PROTOCOLS FOR DEVICE INDEPENDENT, PEER-TO-PEER DISCOVERY AND DELIVERY OF MOBILE SERVICES, IEEEE TRANSACTIONS ON SYSTEMS, MAN AND CYBERNETICS, VOL. 33, NO. 6, NOV, 2003

[MAKR02] S.MARTI, V.KRISHNAN: CARMAN: A DYNAMIC SERVICE DISCOVERY ARCHITECTURE, MOBILE AND MEDIA SYSTEMS LABORATORY, HP LAB PALO ALTO, SEPT. 2002

[MID03] MIDLET BASICS, AVAILABLE AT: HTTP://WWW6.SOFTWARE.IBM.COM/DEVELOPERWORKS/EDUCATION

[MANN04] R. MANNING: JXTA TECHNOLOGY FOR XML MESSAGING, OASIS SYMPOSIUM NEW ORLEANS, LA, APRIL 2004

[OASI06] OASIS REFERENCE MODEL DEFINITION FOR SOA 1.0, AVAILABLE AT: HTTP://WWW.OASIS-OPEN.ORG/COMMITTEES/DOWNLOAD.PHP/19679/SOA-RM-CS.PDF

[ORT05] E.ORT: SERVICE-ORIENTED ARCHITECTURE AND WEB SERVICES: CONCEPTS, TECHNOLOGIES, AND TOOLS, APRIL 2005, AVAILABLE AT: HTTP://JAVA.SUN.COM/DEVELOPER/TECHNICALARTICLES/WEBSERVICES/SOA2/SOA2.PDF

[RIOR06] L. RIORDAN: SERVICES REGISTRY, LAST EDITED ON DECEMBER 2006, AVAILABLE AT: HTTP://WWW.GOVDEX.GOV.AU/CONFLUENCE/DISPLAY/GOVDEXREFERENCE/SERVICES+REGISTRY

[RYAN06] M. RYAN: WHAT IS A SOA SERVICE? NOV 2006, AVAILABLE AT: HTTP://WWW.DIGERATEUR.COM/ARTICLES/WHATISASERVICE.JSP

[SCAL04] WHAT IS SCALABILITY? 2004, AVAILABLE AT: HTTP://WWW.WHATIS.COM/SCALABILITY.HTM

[SRJP06A] S. SRIRAMA, M. JARKE, W. PRINZ: MOBILE WEB SERVICE PROVISIONING, INTERNATIONAL CONFERENCE ON INTERNET AND WEB APPLICATION AND SERVICES (ICIW06), IEEE COMPUTER SOCIETY, FEB. 2006

[SRJP06B] S. SRIRAMA, M. JARKE, W. PRINZ: MOBILE HOST: A FEASIBILITY ANALYSIS OF MOBILE WEB SERVICE PROVISIONING, 4TH INTERNATIONAL WORKSHOP ON UBIQUITOUS MOBILE INFORMATION AND COLLABORATION SYSTEMS, UMICS 2006

[SSJP06] S. SRIRAMA, M. JARKE, W. PRINZ: A MEDIATION FRAMEWORK FOR MOBILE WEB SERVICE PROVISIONING, 10TH IEEE INTERNATIONAL ENTERPRISE DISTRIBUTED OBJECT COMPUTING CONFERENCE WORKSHOPS (EDOCW'06), 2006

[SRIR04] S. SRIRAMA: CONCEPT, IMPLEMENTATION AND PERFORMANCE TESTING OF A MOBILE WEB SERVICE PROVIDER FOR SMART PHONES, MASTER THESIS, JULY, 2004, RWTH AACHEN

[SRIR06] S. SRIRAMA: PUBLISHING AND DISCOVERY OF MOBILE WEB SERVICES IN PEER-TO-PEER NETWORKS, INTERNATIONAL WORKSHOP ON MOBILE SERVICES AND PERSONALIZED ENVIRONMENTS (MSPE'06), NOV. 2006

[TENH06] R. TEN-HOVE: WHAT IS ENTERPRISE SERVICE BUS? APR 2006, AVAILABLE AT: HTTP://BLOGS.SUN.COM/RTENHOVE/ENTRY/WHAT_IS_ENTERPRISE_SERVICE_BUS

[TOPR06] A.TOPRAK: MOBILE WEB SERVICE DISCOVERY IN JXTA/JXME, MASTER THESIS, DECEMBER 2006, RWTH AACHEN

[RIOR06] N,RIORDAN: SERVICES REGISTRY, LAST EDITED ON DECEMBER 2006, AVAILABLE AT: HTTP://WWW.GOVDEX.GOV.AU/CONFLUENCE/DISPLAY/GOVDEXREFERENCE/SERVICES+REGISTRY

[UDDI04] UDDI VERSION 3.0.2, OKT, 2004, AVAILABLE AT : HTTP://UDDI.ORG/PUBS/UDDI-V3.0.2-20041019.HTM

[SOUR04] D. SOURCE: INTRODUCTION TO SOAP 1.1, MAY 2004, AVAILABLE AT: HTTP://WWW.DEVELOPER.COM/SERVICES/ARTICLE.PHP/10928_1602051_1

[WALS02] L.WALSH: UDDI, SOAP, AND WSDL: THE WEB SERVICES SPECIFICATION REFERENCE BOOK, PRENTICE HALL PROFESSIONAL TECHNICAL REFERENCE, 2002

[WEHR06] K. WEHRLE: "MASSIVELY DISTRIBUTED SYSTEMS I" LECTURE NOTES, IN RWTH AACHEN, 2006

[WIKI07] WHAT IS PEER-TO-PEER? 2007, AVAILABLE AT: HTTP://EN.WIKIPEDIA.ORG/WIKI/PEER-TO-PEER

[YUAG03] Y.YUAN, W. ARBAUGH: A SECURE SERVICE DISCOVERY PROTOCOL FOR MANET, THE 14TH IEEE INTERNATIONAL SYMPOSIUM ON PERSONAL, INDOOR AND MOBILE RADIO COMMUNICATION PROCEEDINGS, 2003