QoS-aware Mobile Web Services Discovery Using Utility Functions

By:

Edwin Chan

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2008

Author's Declaration for Electronic Submission of a Thesis

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

ACKNOWLEDGEMENT

I would like to extend my sincerest thanks to my supervisor, Dr. Paul Ward, for providing me with this opportunity. I would like to thank him for all his excellent guidance, precious advice and endless support during my research.

I would also like to thank my examiners Dr. Jay Black and Dr. Kostas Kontogiannis for taking the time to read my thesis. Their feedback was very useful in improving the quality of my thesis.

Finally, I would like to express my sincerest appreciation to my family.

ABSTRACT

Existing QoS-aware Web Services discovery architectures tend to focus solely on fulfilling the requirements of either the client or the provider. However, the interests of the provider and client are not equivalent. The provider's goal is to maximize the profit and consume the least amount of resources. On the other hand, the client's selection is determined by their own requirements which do not always reflect the real resource overheads. This research aims to provide a novel mobile Web Services discovery and selection method based on utility functions to balance the requirements for clients and providers. In the mobile environment, it is critical to conserve resource consumption in addition to fulfilling user requirements, as resources such as wireless network bandwidth and mobile device power are precious. The proposed service selection strategy enables service providers to balance the cost/performance ratios and utilize the network bandwidth more effectively, while the clients can still attain the functional and quality levels specified in the service request.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

API	- Application Programming Interface
CDC	- Connected Device Configuration
CLDC	- Connected Limited Device Configuration
Codec	- Compression/Decompression
CORBA	- Common Request Broker Architecture
CPU	- Central Processing Unit
DCOM	- Distributed Component Object Model
DiffServ	- Differentiated Service
DII	- Dynamic Invocation Interface
FTP	- File Transfer Protocol
GB	- Gigabytes
GHz	- Giga Hertz
HTML	- Hypertext Markup Language
HTTP	- Hypertext Transfer Protocol
IP	- Internet Protocol
J2ME	- Java Platform Micro Edition
JPEG	- Joint Photographic Experts Group
JSR	- Java Specification Request
JXTA	- Juxtapose
MB	- Megabytes
MIDP	- Mobile Information Device Profile
MOS	- Mean Opinion Score
OASIS	- Organization for the Advancement of Structured Information Standards
OS	- Operating System
P2P	- Peer-to-Peer
PDA	- Personal Digital Assistant
RAM	- Random Access Memory
QoS	- Quality-of-Service
RPC	- Remote Procedure Call
SA	- Service Agents
SLP	- Service Location Protocol
SOA	- Service-Oriented Architecture
SOAP	- Simple Object Access Protocol
SQL	- Structured Query Language
TINA	- Telecommunications Information Networking Architecture
tModel	- Technical Model
UA	- User Agents
UDDI	- Universal Description, Discovery and Integration
UDDIe	- Universal Description, Discovery and Integration Extension
URI	- Uniform Resource Identifier

URL	- Uniform Resource Location
VoIP	- Voice over Internet Protocol
W3C	- World Wide Web Consortium
WAP	- Wireless Application Protocol
WML	- Wireless Markup Language
WS	- Web Services
WSDL	- Web Services Description Language
WSLA	- Web Service Level Agreement
WSOL	- Web Service Offerings Language
WS-Policy	- Web Services Policy Framework
XML	- Extensible Markup Language

Chapter 1
Introduction

1.1 Motivation

The growth in handheld-device usage such as mobile phones and personal digital assistants (PDAs), along with the vast improvement in wireless networks over the past few years, has made mobile computing the emerging paradigm of personal computing and communications. As mobile devices are becoming more computationally powerful and wireless networks cover wider geographical areas, the vision of enabling mobile-device users to access their information and services anywhere and anytime has become a reality.

Nevertheless, there are several technological challenges that mobile computing has to face. For example, mobile devices suffer from weak network connectivity when they move out of the range of wireless coverage. Also, the network bandwidth can fluctuate greatly as mobile devices move into another network. Furthermore, mobile devices typically have limited battery power and storage capacity which restricts the number of applications they can run locally [43]. More importantly, the proliferation of the types of mobile devices poses serious interoperability problems because of the diversity of operating systems and the implementation languages of mobile applications. Consequently, developing high-quality mobile applications is a very difficult task [13].

A recent industry trend to facilitate interoperability across heterogeneous technologies and to encourage reuse of existing applications is the deployment of Service-Oriented Architectures (SOAs) [18]. A Service-Oriented Architecture is an architectural style that can be defined as a system in which resources are made available to other participants in the network as independent services. SOA is an ideal paradigm to overcome the interoperability problems in mobile computing [47]. A service in SOA is the implementation of a self-describing, platform-independent functionality. SOA is designed to facilitate sharing of capabilities while minimizing the amount of functionality a single host needs to possess. Such a design is especially effective for mobile devices where storage space on the device is at a premium.

Web Services is presently the most promising technology for realizing SOA [57] after it first emerged a few years ago. Like its predecessors such as the Common Request Broker Architecture (CORBA) and the Distributed Component Object Model (DCOM), Web Services' primary goal is to allow applications to communicate with each other regardless of platform and programming language. By using Web Services, mobile applications can be delivered and executed by the mobile devices only when they are needed. Moreover mobile devices can discover and use available resources from the surrounding devices. This in turn reduces the significance of the constraints in the mobile-computing environment. The convergence of mobile devices and Web Services will create new and compelling possibilities for the mobile telecommunications market. Recognizing this trend, the mobile industry has taken the initiative of implementing Web Services for mobile devices to take advantage of the benefits that Web Services and SOA offer [34].

Web Services discovery is an essential step that the mobile device must perform in order to find Web Services dynamically. Its purpose is to locate a particular Web Service that best matches the user requirements. At present Web Services matching is done statically based on functional requirements, with no discovery at all. However, requirements from a service consumer may be not only functional but also non-functional; i.e., the Web Services quality-of-service (QoS). For example, a service consumer might want to specify attributes such as response time, security level, price, etc., in addition to the desired functional requirements when searching for a service. Moreover, as a consequence of the rapid growth of mobile devices and the abundance of service providers, the consumer now faces the question of how to select the most appropriate service from a variety of available services. In such a scenario, QoS can serve as a key differentiator among different service providers. Furthermore, Web Services discovery mechanisms for the wired environment need to be enhanced for the mobile wireless environment. Challenges unique to the mobile environment, such as rapidly changing connectivity, higher network error rates, along with bandwidth and power constraints, all require special attention. Therefore, there is a need to develop a new mobile Web Services discovery mechanism that has the ability to describe and match QoS offers and demands.

1.2 Industrial Trends

Existing commercial frameworks for mobile Web Services do not address the QoS issue with mobile Web Services discovery, most likely because they are still in their infancy. For example, the Liberty Alliance Web Services Framework, which is a widely accepted standard for identity services in the mobile space, provides mechanisms for identity authentication and Web Services access authorization [4]. Although it is very effective for ensuring authorized access to mobile Web Services, it does not use QoS information for mobile Web Services discovery.

The Java Specification Request (JSR) 232 Mobile Operation Management Specification [37], led by Nokia, Motorola and IBM, is another popular commercial framework for mobile Web Services. The specification defines a standardized component-oriented computing environment for networked services, where software components such as libraries or applications can dynamically use other components. Although the specification makes it easier to realize SOA for the mobile platform, the goal of the specification is not mobile Web Services discovery.

1.3 Research Trends

Service-oriented computing is a promising solution for delivering functionality in mobile networks, especially considering the limited capabilities of mobile devices. This is the primary driver behind the surge in popularity of mobile Web Services. The research effort to date has focused on issues such as optimizing Web Services performance over the wireless network, providing context-aware personalized services, and enhancing the interaction between the user and the mobile-device interface [5]. Nevertheless QoS issues in Web Services are starting to gain importance. An overview of the research efforts on this topic will be presented in Chapter 3.

1.4 Objective

The primary objective of this thesis is to present a QoS-aware mobile Web Services discovery framework. It first investigates how QoS can be used to satisfy the service consumers' requirements. Then it defines QoS parameters that are relevant to mobile Web Services. Finally, it presents the design and implementation of a new QoS-aware Web Services discovery infrastructure that takes into account the unique challenges in the mobile-computing environment, which include minimizing the traffic generated by the discovery process, tolerating intermittent connectivity of devices, and enabling mobile Web Services requesters to differentiate service instances according to the non-functional properties provided. Furthermore, the implementation of this framework makes use of existing tools by expanding and combining them in novel ways, as another key objective for this framework is simplicity of design and deployment.

1.5 Thesis Outline

This thesis proposes a novel architecture for Web Services discovery in the mobile environment. The following is an overview of the remainder of the thesis.

Chapter 2 provides background information on the mobile environment, service discovery, SOA, QoS, and mobile Web Services to provide a better understanding for the remaining chapters. It also lists the benefits of bringing QoS to mobile Web Services.

Chapter 3 investigates the work being done in the field of Web Services discovery for wired as well as wireless environments. It brings out the drawbacks and limitations of existing solutions and also emphasizes areas that this thesis has leveraged.

Chapter 4 presents a new framework for the QoS-aware Web Services discovery in mobile environments that addresses the identified requirements. It provides an overview of the broad functionalities and a brief summary on the implementation.

Chapter 5 provides the details of the prototype implementation of this framework, and explains how to apply the implementation.

Chapter 6 presents the performance evaluation. The simulation and experiments used to test the framework are also described.

Chapter 7 summarizes the contributions of the thesis and enumerates some future directions of research.

Chapter 2
Background

This chapter describes a typical mobile environment and reviews the concepts of Web Services, SOA, Semantic Web, service discovery, and QoS.

2.1 Mobile Environment

Figure 1 [16] depicts a typical mobile computing environment. It illustrates the three main interaction patterns, denoted S1 to S3.

1. The wireless network hosts both the requestor and provider.

2. The requestor is mobile whereas the provider is situated in the wired network. This is probably the most common setting, opening up wired services to mobile users.

3. The requestor is located in the wired network and invokes a service on a mobile device. Examples include push services and tracking applications.

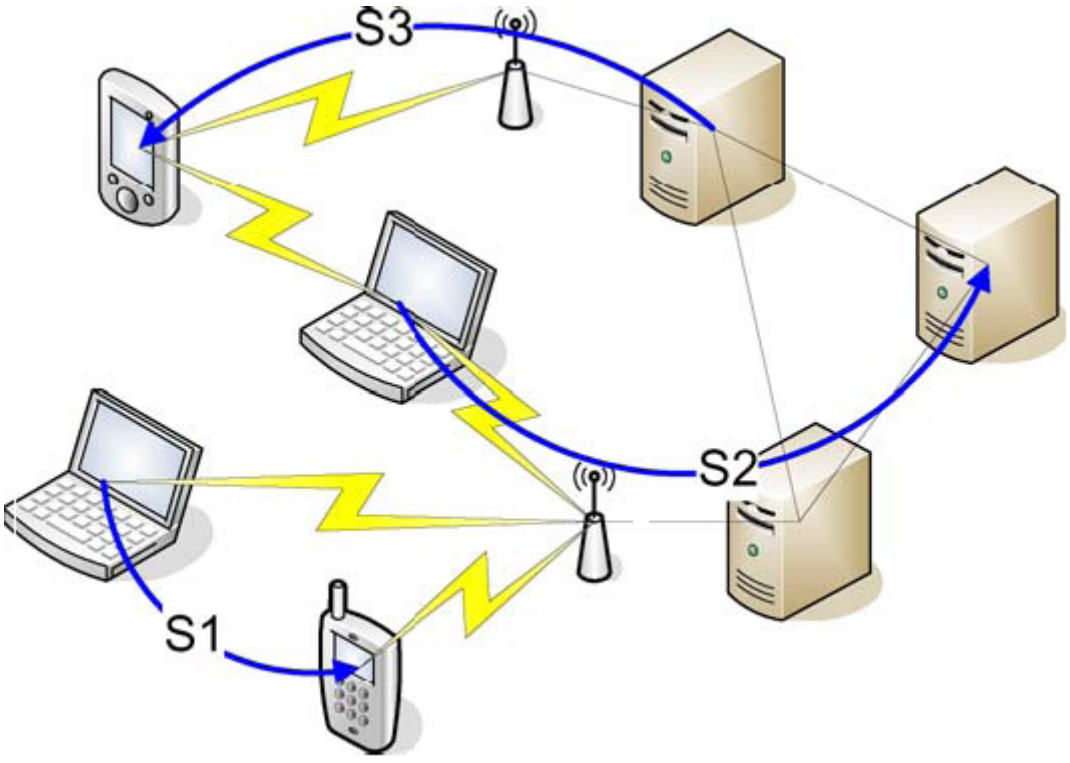


Figure 1: Typical Mobile Environment

Mobile devices come in a wide range of form factors such as PDA or mobile phone. When compared to servers and desktop computers, mobile devices are generally characterized by much lower CPU power, smaller memory and storage capacities. Also, mobile devices generally have to rely on wireless communication for connectivity. Moreover, the user of the device may not always be in the vicinity of a network access point. Therefore, unlike dedicated servers, mobile devices typically have intermittent connectivity to the network, and the services offered on a mobile device may not always be accessible.

The mobile environment is also very heterogeneous. Although the participating mobile devices such as smart phones, PDAs, and laptops tend to share certain functionalities, the mobile applications on the devices are most likely running on different operating systems (e.g., Symbian OS, Windows CE, Palm OS, embedded Linux, etc.) and implemented in different programming languages (e.g., J2ME, .NET Compact Framework, C/C++, etc.) [52]. The heterogeneous nature of the participating devices can often lead to integration problems.

In addition, the mobile device and its environment may not trust each other completely. For example, a shopper may enter a store with a mobile device. The shopper needs to be convinced that he/she is connected to the store directly and not to a man in the middle. On the other hand, the store may permit the mobile device to connect to the 802.11 network in the store, but may restrict the mobile device to accessing only a limited number of network locations. Furthermore, the user of the mobile device may want to ensure the confidentiality of his/her communication traffic from other shoppers at the same store.

2.2 Existing Technologies for Supporting Mobile Internet Access

Previously, many cellular providers relied on the Wireless Application Protocol (WAP) to offer Internet services [35]. A WAP browser is designed to provide all of the basic services of a desktop based web browser but it must operate within the restrictions of a mobile phone. However, WAP technology requires web pages to be written in WML (Wireless Markup Language) instead of HTML. The menu-oriented approach of WML makes it very difficult to provide the rich navigation experience of a desktop web browser. Although surveys suggest people who have tried out WAP services generally are willing to put up with the small inconveniences with browsing the user-unfriendly interface on a small screen, they are still unwilling to adopt WAP because of the dearth of useful content [27].

As mobile devices are becoming more powerful and have more capabilities, WAP technology is gradually being replaced by mobile applications. These new mobile applications overcome the limitations of WAP by offering rich display capabilities and better functionalities for providing mobile services. Examples of mobile services include access to information resources (e.g., searching, language translation, weather forecast, etc.), telemetry (e.g., receiving traffic updates and logistics tracking), and mobile shopping and banking (e.g., booking flights, billing of services, etc.) [58].

Nevertheless mobile applications can vary significantly in terms of the configuration. There are several operating systems and programming languages for implementing mobile applications.

This poses serious interoperability problems and contributes to the high cost of developing mobile applications. Moreover, pre-installing mobile applications on mobile devices is not scalable because of the limited storage space available on mobile devices.

2.3 SOA In Mobile Computing

In software engineering, the effective use of previously written software in building new applications is known as software reuse. Many software designs follow this pattern to ensure minimal interdependence between software components [6]. The various subcomponents are coupled together to ensure integrity and proper functioning of the whole software. In other words, the various subsystems carry out their own specific functions to make a whole system work.

Service-Oriented Architecture (SOA) is an architectural design that helps achieve software reuse. It reorganizes software applications into an interconnected set of services, each accessible through standard interfaces and messaging protocols. An important aspect of SOA is the separation of the service interface (the what) from its implementation (the how). Such services are consumed by clients that are not concerned with how these services will execute their requests. Thus, the SOA architectural-design approach is particularly beneficial when multiple applications running on varied technologies need to communicate with each other.

2.4 Service Discovery

In order to comprehend Web Services discovery, one first needs to understand the concept of service discovery. Service discovery, which is the process of locating services, is an essential requirement for any distributed, open, dynamic environment. It provides a way to reuse components that are available in the vicinity of a device. Figure 2 [2] illustrates the components in a typical service discovery system.

A service is a set of functionalities that can be used by a person, a software program, or another service. A client is an entity that wants to discover and use available services. A directory stores information about the services and is the location where service lookup is performed. Service discovery generally involves:

1) Bootstrapping
2) Service registration
3) Querying for services
4) Service lookup

Bootstrapping specifies how the users and services establish contact with the directory. Service registration stores information about the service in the directory. Querying for services allows the client to search for the service. Service lookup returns the result of the service query to the client.

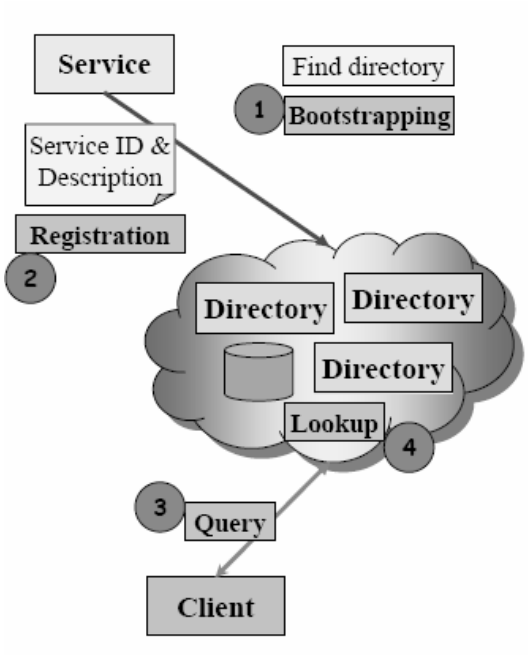


Figure 2: Components in a Typical Service Discovery System

2.5 Game Theory and Utility Functions

It is important to understand some of the concepts in game theory [19] as this thesis uses utility functions.

- *Pareto efficiency* is a measure of efficiency in game theory. An outcome of a game is Pareto efficient if there is no other outcome that makes every player at least as well off and at least one player strictly better off. In another words, a Pareto Optimal outcome cannot be improved upon without hurting at least one player. Note that a Pareto optimal outcome is not necessarily a fair outcome.

- *Stackelberg leadership* model is a strategic game in which the leader firm moves first and then the follower firms move sequentially. The players of this game are a *leader* and a *follower* and they compete on quantity.

- *Nash bargaining game* is a simple two-player game used to model bargaining interactions. In the Nash Bargaining Game two players demand a portion of some good. If the two proposals sum to no more than the total good, then both players get their demand. Otherwise, both get nothing.

- *Utility* is the quantification of a person's preferences with respect to certain objects. A utility function is an abstract, mathematical expression of this measurement. For example, a utility function, written as:

  $U = f(x_1, x_2,...x_n)$,

  means that items x_1, x_2, to x_n all contribute to a person's utility.

2.6 Concepts of Web Services

Web Services is the set of technologies presently being used to implement SOA. The Web Services architecture as defined by the W3C is "a software system whose public interfaces and bindings are defined and described using XML and can be identified with a URI (Uniform Resource Identifier) [54]. These systems may then interact with Web Services in a manner prescribed by its definition, using XML-based messages conveyed by Internet protocols." As described by the definition, the primary goal of Web Services is to enable application-to-application communication over the Internet, by allowing seamless access to other software components through standard Web technologies, regardless of platforms, implementation languages, etc.

Figure 3 shows a typical Web Services development/deployment scenario, which consists of six elements.

1) The UDDI server, which is a central registry providing the yellow-page service.
2) The Web Services provider (WS provider), which hosts the service.
3) The Administrator who deploys and manages the WS provider.
4) The UDDI client, which provides an easy-to-use interface for users to manipulate data in the UDDI server.
5) The Web Services consumer (WS consumer), which invokes the service of the WS provider.
6) The Developer who is responsible for developing the WS consumer.

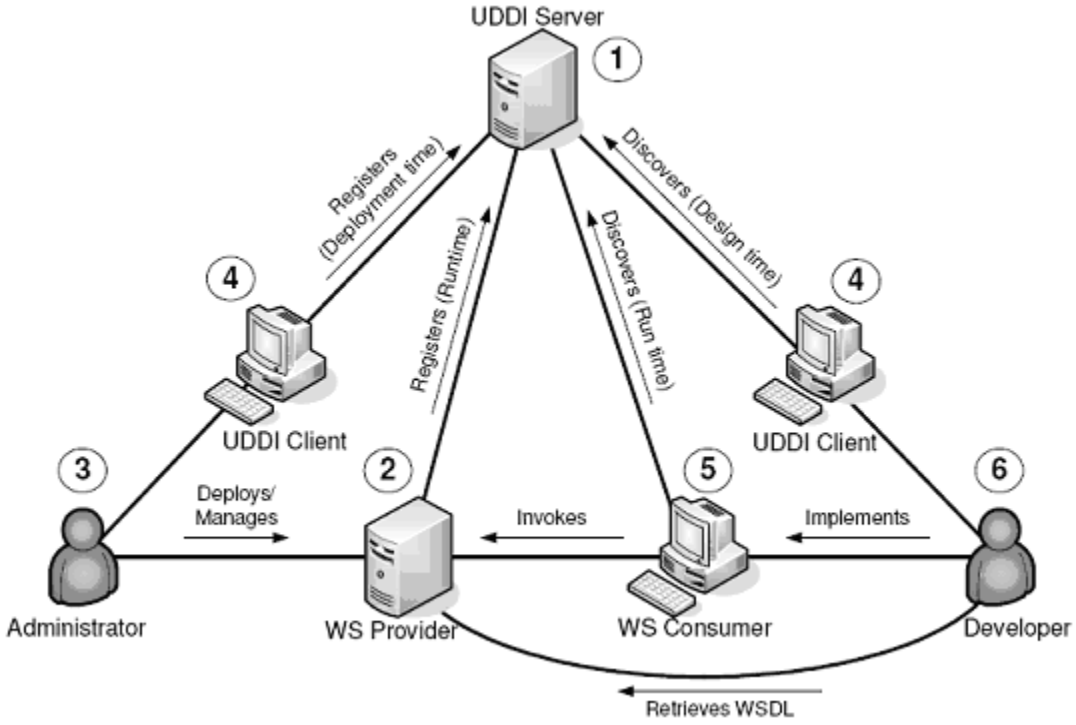


Figure 3: Web Services Development/Deployment Scenario

The elements are related as follows:

• The administrator deploys the WS provider on a physical machine and uses a UDDI client to publish the information about the service to the UDDI server. The WS provider can also update its existing service information in the UDDI server.

• The developer performs a lookup operation through a UDDI client (design-time discovery) to find a desirable service from the UDDI server. After finding a suitable service, the interface definition (WSDL document) can be retrieved from the WS provider.

• The developer implements a WS consumer based on the retrieved interface definition. The WS consumer interacts with the WS provider directly to invoke the service.

• If the WS consumer fails to invoke the service of the provider, it tries to obtain updated information about the provider from the UDDI server again (run-time discovery). The WS consumer can arrange the next request according to the newly fetched information.

The Web Services layer itself is based on several Internet protocols, such as WSDL, SOAP and HTTP, as depicted by Figure 4. In term of the Internet model, the Web Services layer is in between the Transport and Application Layers [49].

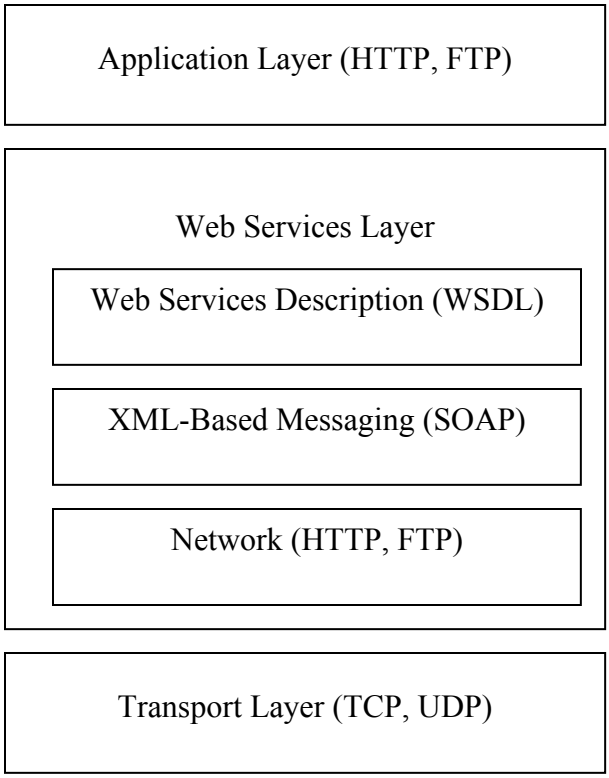


Figure 4: Web Services Protocol Stack

The three most important components that provide the core functionality of Web Services are WSDL, SOAP, and UDDI.

2.6.1 WSDL

WSDL standardizes the service description using XML. A WSDL document describes the following aspects of a service [11]:

- **message**: defines the data types of the input/output messages used by the service
- **port-type**: defines all operations of the service, each of which contains an input and an output message
- **binding**: defines the protocols used to invoke the service
- **service**: defines the name and the port of the service
- **port**: defines the address of the service

After obtaining the WSDL document describing the service, a developer can build a WS consumer immediately because the WSDL document contains sufficient information for service invocation.

2.6.2 SOAP

SOAP is a protocol for exchanging XML-based messages over the computer network. The service consumer sends a SOAP request to the service provider, and the service provider processes the request and returns a SOAP message as the reply to the consumer.

SOAP consists of three parts [38]:

- The SOAP envelope construct defines an overall framework for expressing what is in a message, who should deal with it, and whether it is optional or mandatory.
- The SOAP encoding rules define a serialization mechanism that can be used to exchange instances of an application-defined data-type.
- The SOAP Remote Procedure Call (RPC) representation defines a convention that can be used to represent remote procedure calls and responses.

SOAP does not define any application semantics such as a programming model or implementation-specific semantics; rather it defines a simple mechanism for expressing application semantics by providing a modular packaging model and encoding mechanisms for encoding data within modules. Thus SOAP hides all language and platform-specific information and presents the message in a universal way, so any application can parse the information without any ambiguity.

2.6.3 UDDI

The UDDI registry is the key component connecting the service consumers and the service providers. It can be described as a database for storing information about Web Services. Industrial categorization and technical information about services can be stored and viewed in the UDDI registry either by using a Web interface or an application program interface (API). A service provider makes its services available to public users by publishing information about the service in a UDDI registry. Service requestors then locate the services by searching the UDDI registries.

The information contained in a UDDI registry can be categorized using a metaphor of a telephone directory. The information can be categorized as:

• *White pages*: listing of organizations, contact information and services these organizations provide.

• *Yellow pages*: classifications of both companies and Web Services according to standard or user-defined taxonomies.

• *Green pages*: information on how a given Web Service can be invoked (e.g., WSDL specification).

In a UDDI registry, the data is stored in XML format, and data types are defined using XML Schema. An XML schema is a document that describes the valid format of an XML dataset.

This includes what elements are or are not allowed at any point, the attributes for any element, the number of occurrences of elements, etc.

The UDDI server organizes the data hierarchically, with a unique ID representing the business key, service key, or binding key. The following describes the data types in a UDDI registry [2].

1. Businesses, organizations, and other units are represented by the *businessEntity* data structure
2. Service abstracts, describing the functionalities of services, are represented by the *businessService* data structure.
3. Technical information defining the location of the provider is represented by the *bindingTemplate* data structure.
4. The cooperative relationship between two organizations is represented by the *publisherAssertion* data structure.
5. Descriptive information related to abstract specifications of knowledge of the Web Service, which is represented by the technical model (tModel).

The main content of a tModel consists of a unique key and a pointer to the documentation of the service. The documentation can reside anywhere and is typically written in WSDL. The human service requestor can browse the UDDI registry to gain insight into what a certain service does and which properties and interfaces it provides.

Figure 5 provides an illustration of the tModel.

2.7 Semantic Web and Ontology

In order to understand some of the research works described in Chapter 3, it is important to first understand the concept of Semantic Web. Semantic Web [8] can be seen as an extension of the current Web where information is given well-defined meaning, allowing machines to more easily process Web data. For example, Web Services can be described formally in terms of what effect they have on the world, and the meaning of the data they consume and produce. This will allow machines to discover services automatically and then reason about how they can be composed and invoked.

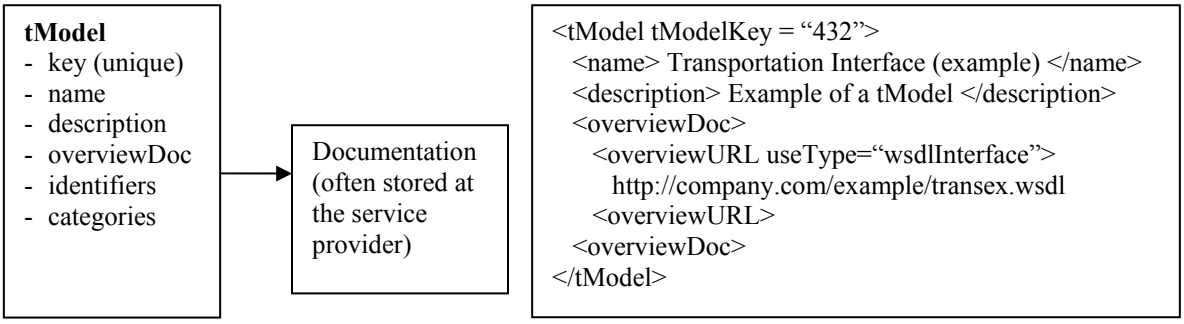


Figure 5: Illustration of the tModel

Ontologies are central to the vision of the Semantic Web [25] . They facilitate knowledge sharing and reuse among both humans and computers. They offer this capability by providing a formal conceptualization of a given domain. They provide a framework to define precisely the concepts we want to reason about, and the relationships which hold between those concepts.

2.8 Mobile Web Services

Mobile Web Services extends the protocols and interfaces outlined by the core Web Services specifications to the mobile environment. Since mobile Web Services emerged just a few years ago, only a subset of the core Web Services specifications are supported. For example, the JSR 172 J2ME specification, which is the latest mobile Web Services specification for Java, has introduced new APIs to provide XML processing and XML-based RPC communication in J2ME. However, it still does not have any APIs to provide capabilities for standard service registration and discovery with Universal Description Discovery and Integration (UDDI) [40].

Mobile Web Services still have a variety of challenges to overcome. The physical and environmental constraints of mobile devices mean mobile Web Services implementations need to be more careful in their usage of computation, memory, and energy resources. Any client-side software that interacts with Web Services on mobile devices should be written in a manner that is tolerant to the above constraints. Any application that uses or composes such Web Services cannot rely on the service being always available and instead needs to operate in an opportunistic manner, leveraging such services when they become available.

2.9 Web Services Discovery

Web Services discovery can be viewed as a specialized form of service discovery. It is defined by the Web Services standard as "the act of locating a machine-process-able description of a Web Service that may have been previously unknown and that meets certain functional criteria" [10]. It refers to the process of obtaining the information needed to use the Web Service, including the protocol and interface information included in the WSDL document, and the XML Schema information in the message content.

Web Services discovery serves three important purposes:

1. To formulate a request that describes the needs of a user.
2. To provide a matching function that pairs requests to services with similar descriptions.
3. To select the service with the best quality among those able to satisfy the user's goal.

However, the accelerated adoption of Web Services can have an adverse effect. As the pool of Web Services grows, a service requestor will require more time to find the desired service. Also, it becomes more difficult for each service provider to differentiate its services from those of other providers. Therefore, in order for Web Services to continue their success, it is crucial to have an effective Web Services discovery mechanism that overcomes the above issue.

2.10 QoS in Mobile Web Services

The current UDDI search mechanism only focuses on search criteria such as business name, business location or category, service type by name, business identifier, or discovery URL [61]. Also, the WSDL interface definition usually only specifies the syntactic signature for a service [36]. This approach suffers from a major shortcoming: it is unable to perform Web Services matching based on non-functional requirements of the requestor; i.e., it is not capable of answering the questions "*Does the Web Service meet the desired performance requirements of the requestor?"* or *"Is the Web Service secure and reliable enough for the transaction?"* Furthermore, with the proliferation of Web Services, it is likely that many Web Services have similar functionalities. It is foreseeable that providers would want to describe both the functional and non-functional aspects of their advertised Web Services in order to distinguish themselves.

The non-functional attributes of Web Services are commonly referred to as "Quality-of-Service," or QoS attributes. QoS is a set of non-functional characteristics that may impact the quality of the Web Services. If the advertised Web Services have certain values of these QoS attributes, then the Web Services are considered to be conforming to a certain QoS level.

The "Quality-of-Service (QoS) requirements for Web Services" document drafted by the W3C working group [55] defines the primary attributes for Web Services QoS, which include performance, reliability, security, availability, etc. Table 1 summarizes the major requirements for supporting QoS in Web Services.

Applying QoS to mobile Web Services discovery has several benefits:

1. When there are multiple Web Services offering the same functionality to the requestor, the associated QoS can be used by the providers to distinguish their services.

2. The requestor can specify non-functional requirements in his/her service query. These requirements include the type of hardware that he/she is using, how he/she desires to access the service (a service being offered in real time, or only between certain time intervals), and the cost he/she is willing to pay for accessing the service. This helps the requestor to select the Web Service with the QoS that best represent his/her desires.

3. The service provider can offer services at different levels. This allows the provider to balance the cost/performance ratios and utilize the network bandwidth more effectively, while also attaining the quality levels expected by the requestor.

The subsequent chapter will elaborate the key issues in mobile Web Services discovery. It will also examine some existing service discovery mechanisms and current research efforts in mobile Web Services discovery with QoS.

QoS Attribute	Description
Performance	Performance is measured in terms of throughput and latency. *Throughput* is the number of Web Services requests served in a given time period. *Latency* is the round-trip time between sending a request and receiving the response.
Reliability	Reliability is the ability for the service provider to ensure the service is delivered with the desired level of quality of service
Security	Security defines whether the Web Services authenticate the parties involved, encrypt messages, and provide access control. The service provider can have different approaches and levels of security.
Cost	Cost measures the monetary cost for requesting the Web Services. It may be charged per request, or could be a flat rate charged for a period of time.
Availability	Availability is the probability that a service is available.

Table 1. Web Services QoS Attributes

Chapter 3
Related Works on Mobile Web Services Discovery with QoS

In this chapter, several service discovery mechanisms for the wireless environments are examined critically, along with the research on QoS-enabled Web Services. The key issues in QoS-related mobile Web Services discovery are also presented.

3.1 Issues in QoS-Related Mobile Web Services Discovery

One of the key issues in mobile Web Services discovery is service matching. Service matching is the process of deciding automatically whether an offered service is able to fulfill a given service request. Furthermore, any service-selection algorithm for the mobile environment should take into account available contextual information such as geographic location, and the device's processing power and battery life.

Another issue in mobile Web Services discovery is resource heterogeneity. Mobile devices with varying resources and capabilities exist in the geographical vicinity of one another. Resources refer to either services or computation power available to devices. The service discovery architecture should ideally be capable of performing its functionality with the most efficient use of the surrounding environment and with minimum network overhead.

Enhancing mobile Web Services discovery with QoS helps refine service matching as it allows non-functional requirements to be added to the service request. QoS-enhanced Web Services discovery also aids in resolving the resource heterogeneity issue because it enables providers to offer different classes of services. It also permits a requestor to select the most appropriate service according to the computational power of the mobile device.

3.2 Types of Service Discovery Architectures

The following sections present the common mobile Web Services discovery mechanisms. As it is impossible to name and consider all the service discovery mechanisms in this thesis, the ones that are most relevant are highlighted.

The architectures for service discovery can be divided primarily into two categories: directory-based or directory-less. The directory-based approach has either a central location or multiple locations on the network that act as the service directory to record the available services as well as methods to use them. In order to use the service, service requestors needs to retrieve information about the service from the directory. On the other hand, in situations where services are available in an inconsistent manner, a mechanism for publishing and discovery without the use of a directory is needed. The directory-less approach uses some form of broadcasting to advertise the services available to the network or for clients to find services.

3.3 Centralized/Semi-Centralized Directory-based Approach

UDDI, Jini, Liberty Identity Web Service Framework Discovery Service and Service Location Protocol are some of the common mobile Web Services discovery infrastructures that follow the directory-based approach.

3.3.1 UDDI

The details of UDDI have been explained in a previous section, but to summarize, it is an XML-based centralized registry that manages information about service providers, service implementations, and service metadata. Service providers describe their services in WSDL and publish this information to a UDDI directory. In order to find a web service with UDDI, the service requestor sends a query containing keywords to the UDDI registry, and from the query results the requestor can select the suitable service.

3.3.2 Jini

The Jini technology infrastructure and programming model are built to enable services to be offered and found in a networking environment. Jini addresses the issues related to mobile and specialized devices by providing the abilities to announce presence on the network and automatic discovery of devices in the neighborhood [48].

There are two methods with which a client can discover the Jini lookup service. In the first method, the client can send out a multicast request of a specified format. All Jini lookup services that receive the request will respond to it, and the client is said to have discovered the lookup services. In the second method, the client attempts to connect to a lookup server having known the existence and location of the lookup server.

The lookup service in Jini consists of a directory of service items, which are made up of three elements: its service interface, a Java object (service proxy) on which calls to use the service can be made, and a set of service attributes that describe the service. In order to be discovered, new services register this information to one or more lookup services. Furthermore, Jini employs the concept of leasing: a service registers itself for a given time period, called a lease. When the lease expires the service is no longer advertised.

Figure 6 outlines the discovery/join/lookup processes using the lookup server in Jini. The service provider locates a lookup server by multicasting a request on the local network for any lookup server to identify itself (discovery). Then, a service object for the service is loaded into the lookup server (join), which contains a Java programming language interface for the service including the methods that users and applications will invoke to execute the service, along with any other descriptive attributes. A client locates an appropriate service by its type, i.e., by its interface written in the Java programming language, along with descriptive attributes that are used in a user interface for the lookup server. Then the service object is loaded into the client to be invoked.

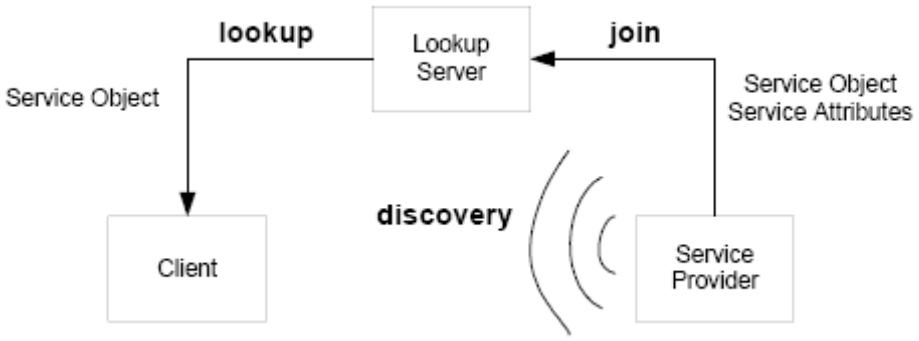


Figure 6: Jini Process of Discovery/Join/Lookup with Lookup Server

3.3.3 Liberty Identity Web Service Framework Discovery Service

As described above, the Liberty Alliance Web Service Framework is an industrial specification for developing mobile Web Services, and the service discovery aspects of the framework are defined by the Liberty Identity Discovery Service Specification [3]. The specification introduces a directory service that provides a security token needed to access the service. The directory service returns the description of an identity-based Web Service, which includes the following attributes:

- *Service Type*: the type of service identified by a URI that can reference an abstract WSDL document;
- *Provider ID*: the service provider's URL from where the metadata can be retrieved;
- *Service instance description*: protocol endpoint and other information needed to access the service;
- *Security mechanism*: the type of security applied;
- *Security credentials*: the credentials needed to invoke the service.

Note that the directory service does more than just providing reference information on how to access the service: it also authorizes and returns information needed to invoke the service, such as security credentials. These characteristics differentiate this approach from UDDI, which focuses solely on providing service and service provider information.

3.3.4 Service Location Protocol

Service Location Protocol (SLP) is a language-independent protocol that establishes a framework for service discovery using three types of agents that operate on behalf of network-based software: (*i*) Service Agents (SA) advertise locations and attributes on behalf of services, (*ii*) Directory Agents (DA) aggregate service information, and (*iii*) User Agents (UA) perform service discovery on behalf of client software.

Active discovery and passive discovery are the two methods of service discovery in SLP. In active discovery, the user agents and service agents multicast their requests to the network. In

passive discovery, directory agents multicast advertisements for their services and continue to do this periodically. SLP has two modes of operation: (*i*) when a directory agent is present, it collects all service information advertised by the service agents, and the user agents unicast their requests to the directory agent, (*ii*) without a directory agent, the user agents will repeatedly multicast their requests. Service agents listen for these multicast requests and make unicast responses to the user agents if they have advertised the requested service.

3.4 Directory-less Approach

Examples of directory-less service discovery architectures include Web Services Dynamic Discovery (WS-Discovery) and Peer-to-Peer.

3.4.1 WS-Discovery

The WS-Discovery specification is a specification that attempts to extend Web Services capabilities into the occasionally connected world of computing and peripheral devices.

The specification defines a multicast discovery protocol to locate services connected to a network. Each requester that is looking for a service propagates its query to the multicast group, and the target services that match return a response directly to the requester. The default attributes for matching are the type of the service and the scope in which the service resides; other attributes, such as the name of the service, can also be used. Each service provider announces itself through the multicast group to expose the services it can provide. By listening to this multicast group, clients can detect newly available target services dynamically without repeated probing.

In order to scale to a large number of endpoints, the specification also defines multicast suppression behavior if a discovery proxy is available in the network. When a discovery proxy detects a probe sent by multicast, the discovery proxy sends an announcement for itself. By listening for these announcements, clients can detect the discovery proxy and switch to use a discovery-proxy-specific protocol. However, if a discovery proxy is unresponsive, clients revert to using the standard multicast protocol. These discovery proxies can communicate with each other in order to extend the discovery scope to other subnets.

3.4.2 Peer-to-Peer

A Peer-to-Peer (P2P) system does not have a fixed infrastructure but instead relies upon the participant nodes of the network. The essence of P2P computing is that nodes in the network directly exploit resources present at other nodes of the network without intervention by any central server. In the context of mobile computing, mobile devices act as the nodes in the P2P paradigm, and each device can provide services to other devices in the distributed environment and can also use remote services.

Schneider [44] discusses the convergence of P2P and Web Services. He compares the common P2P protocols with Web Services in several aspects, namely the conceptual architecture, the wire protocols (the connecting mechanism), security, discovery, reliability, and business standards. In

order to use P2P in Web Services, each service of the Web Services provider must be treated as a service resource of the P2P network. Then, a Web Services consumer can be transformed into a peer querying the service resource in the P2P network (consumer peer), and a Web Services provider into a peer providing the service resource in the P2P network (provider peer). The XML-based SOAP message can be enveloped by a P2P protocol and then transmitted over the P2P network. He concluded that the convergence of P2P and Web Services can potentially increase efficiency and decrease cost.

Nevertheless, there are still several issues with adopting P2P in the mobile environment, since it is composed of both infrastructure-based and ad-hoc networks. For example, each mobile device must be addressable, but in some cases the IP address might be hidden by a firewall. For this purpose a P2P platform such as Juxtapose (JXTA) provides a layering over a communication networks that abstracts from lower level transport protocols. It provides a namespace mechanism that allows for direct P2P communication even if that requires crossing the boundaries of different networks or firewalls. Thus, new nodes may join the network and be integrated on an ad-hoc basis.

A second issue with P2P is that it limits the scope of interactions to only nodes within a local region. In the approach suggested by Gehlen and Pham [23], the problem of limited scope is overcome by selecting the devices which are preferably less dynamic and more powerful to take on the broker role within the environment. The broker enables service exchange between a server and other peers. Services deployed within the framework are described using WSDL and are published to a local service repository component host on a server, and the brokers within the environment synchronize with this server to ensure consistency. Thus, each node can publish its services and clients can consume services provided by the node.

3.4.3 Dynamic Invocation Interface

Currently, there are three methods for accessing and invoking Web Services, which are static stubs, dynamic proxy, and dynamic invocation interface, though J2ME's JSR-172 specification only provides support for static stubs. A static stub is appended to the client at compile time for the WSDL-to-Java mapping tool to generate the required client-side artifacts. As a result, the client can invoke methods of a Web Service directly via a stub. One disadvantage with using static stubs is that they are not portable and thus the generation of a new stub is required even with the slightest change in the Web Services definition. As a consequence, mobile users can only access pre-defined services for their mobile devices.

There are two drawbacks with accessing pre-defined services. First, this approach is unable to satisfy all mobile users, since each individual has different needs. Moreover, it is unable to take advantage of the QoS Schema, since non-functional attributes such as availability and price, are subject to change at runtime. Therefore, a dynamic Web Services selection infrastructure is more appropriate for mobile devices than having pre-installed services. With dynamic service discovery and binding, service providers can add new capabilities at anytime and in turn give mobile users a huge choice of available services at runtime. However, semantic coupling still exists in this approach. For example, a client passes a 10-digit number when he/she invokes a service through the dynamic invocation interface. The server cannot distinguish whether the

client is sending a telephone number or a bank balance without the knowledge being embedded in the code. Thus in this approach semantic coupling still remains even though syntactic coupling is removed.

Nielsen et al. [42] proposed a solution for providing dynamic service discovery in J2ME. Their solution introduces a dynamic proxy between service providers and mobile clients. This intermediate entity uses the dynamic invocation interface (DII) as its communication mechanism. Similar to the DII concept in CORBA, DII in this framework allows dynamic invocation of Web Services without having to know interface details at compile time by marshalling parameters into a request and invoking methods using runtime type information. Furthermore, the use of the DII communication mechanism avoids the generation of each stub class required for each available service. This intermediate entity possesses a service registry which stores the list of URL addresses of accessible services. This list of available Web Services descriptions is sent to the mobile client. Once the mobile clients have received the description of available services, they send requests for services of their interest to the intermediate entity. The intermediate entity then invokes the appropriate Web Service and returns the result to the user. With this approach, mobile clients can locate new services at runtime without updating their client application, since the intermediate entity invokes the Web Services dynamically; thus, the services offered to the mobile users can be updated during runtime.

3.5 Problems with Existing Service Discovery Architectures

The aforementioned service discovery mechanisms suffer from various shortcomings:

- Jini has issues with interoperability as all components in the discovery system must be tied directly or indirectly to the Java runtime. Furthermore, Jini requires service advertisements to be expressed in the form of Java interface descriptions, which has tight syntactic coupling.

- WS-Discovery does not provide a rich metadata model for information about the Web Services. The metadata model of WS-Discovery by default includes only the Web Services type, scope, and the endpoint reference.

- The mobile environment is vastly different from the P2P environment as mobile devices typically do not act as service providers. The benefits of adopting P2P for mobile Web Services have not been demonstrated.

Most importantly, Web Services QoS information is not being used in any of above service discovery mechanisms.

3.6 Motivation for a New Discovery Mechanism

The growing number of available mobile Web Services is raising new demands in service specification, publication and discovery. Mobile Web Services consumers need mechanisms to discover suitable services. In order to accomplish this, service specifications must provide sufficient information and be published in service registries.

On the other hand, service providers always strive to serve customers with high performance on the one side and to keep their operating expenses low on the other side. In order to save network and server capacity, the provider can either limit the number of customers or expand the capacity. In the first case, client satisfaction can be jeopardized when customers' inquiries are refused. In the second case, capacity can become under-utilized when fewer customers use the service than forecasted. Service providers must find an optimal relation between user satisfaction and system utilization in order to gain the highest possible profit from their business.

The new QoS-aware mobile Web Services discovery framework proposed in this thesis is designed to address the above issues. This framework selects the service that gives the most benefits to the client, while also maximizing the profit of the service provider. Moreover, Web Service QoS can be updated easily by the provider, so that clients can make decisions based on the latest information.

The following are the main requirements targeted by this QoS-aware mobile Web Services discovery framework:

- Allow both clients and providers to specify requests and offers with QoS properties;
- Provide a flexible way for providers to publish and update their service offers;
- Be capable of locating a service based on both functional and QoS properties;
- Provide service selection that is sensitive to attributes such as device power, network bandwidth, provider's cost, etc;
- Be compatible with standard Web Service protocols such as SOAP, WSDL, and UDDI.

3.7 Existing Approaches for Qos-Enabled Web Services

Issues in QoS-enabled Web Services are starting to gain attention from industry and academia. The following section gives an overview of some of the major approaches that have been developed for QoS specification and management for Web Services.

The Web Services provider has to accommodate the diverse characteristics and needs of its consumers. This is further exacerbated as user requirements are becoming more complex with value-added features like encryption or reliability. One way to address this issue is to offer classes of service that differ by Web Services quality.

Web Services quality, or QoS, refers to the observable parameters relating to a non-functional property; for example the response time of a request. The level of quality is an agreed upon constraint over the QoS parameters, potentially dependent on a precondition. In order to offer differentiated service based on QoS, the service provider and its clients need to establish an

ontology that defines the QoS metrics to be measured, expected ranges of values for the measured QoS metrics, conditions to be evaluated, prices for successful execution of operations, description of what happens if the conditions are not met, and the actual values of the measured QoS metrics and evaluated conditions. This stimulates the need to include non-functional attributes in the Web Services description.

While WSDL descriptions are mandatory for using Web Services, they are inadequate since WSDL only provides basic information for Web Services integration, such as the names of the methods, formats of the messages, and the URLs at which the Web Service is hosted along with the binding protocols. Therefore one of the research questions raised by this thesis is how to embed Web Services quality information in the service discovery process. Some existing approaches include Web Services Policy Framework (WS-Policy), Web Service Level Agreement (WSLA), Web Service Offerings Language (WSOL), and Universal Description, Discovery and Integration Extension (UDDIe).

3.7.1 WS-Policy

Efforts are now focusing on improving WSDL by considering aspects that are not directly related to the functional aspects of Web Services. WS-Policy represents one of these efforts and is a candidate to become a future standard for Web Services policy specification. It is a specification that allows service providers to express their Web Services capabilities and for services requestors to specify their requirements.

All Web Services have a minimum set of requirements that must be met in order to be consumed by a client. These are normally documented for the client developer and typically include the kind of security token that must be used, or whether the message needs to be encrypted. WS-Policy provides a means of documenting these minimum requirements and automatically enforces them in WSDL. It comprises a model and syntax for specifying Web Services policies. The Web Services policy is represented by a policy expression in XML format. A policy expression is formed by grouping individual properties of the Web Services QoS characteristics, which are known as *assertions*. For example, an assertion can declare that the message be encrypted. Each set of assertions is termed an *alternative*. A policy is built up using alternatives and nested combinations of the XML tag operators `<wsp:All>` (combining two existing policies to form a new policy), `<wsp:ExactlyOne>` (requires exactly one of the behaviors represented by the assertions), and the attribute *Optional* (whether the behavior is optional during the Web Services interaction). This policy syntax is used to describe the combinations of alternatives that form a set of instructions for matching Web Services.

Figure 7 is an example of a policy expression. This policy expression requires the use of addressing, and also one of transport-or-message-level security for protecting messages.

```
<Policy>
  <All>
    <wsap:UsingAddressing/>
    <ExactlyOne>
      <sp:TransportBinding>...</sp:TransportBinding>
      <sp:AsymmetricBinding>...</sp:AsymmetricBinding >
    </ExactlyOne>
  </All>
</Policy>
```

Figure 7: Example of a Policy Expression

Nevertheless, the syntax of WS-Policy has a major limitation as it can only describe the presence or absence of an attribute. For example, the requestor can specify that exactly 128 bit security is to be used, but it can not specify that at least 128-bit encryption be used. Moreover, WS-Policy encodes the non-functional properties into WSDL. WDSL is designed to hold information about the functional aspects of a Web Service, which are not expected to change often once designed. Conversely, quality attributes of mobile Web Services such as network performance and price are subject to frequent changes under the dynamic nature of the mobile environment. It becomes very awkward to recompile the WSDL service interface each time a non-functional property changes. A more flexible way is to delegate the non-functional description of a service from WSDL to an extra file so that changes to non-functional properties can take place without changing the WSDL. This is the approach taken by WSLA and WSOL.

3.7.2 WSLA

A Service Level Agreement (SLA) specifies an agreement between a service provider and a customer, and specifies the measures to be taken in case of deviation and failure. The Web Service Level Agreement (WSLA) project by IBM aims at defining and monitoring SLAs for Web Services [15]. It provides a language for service requestors and providers to define a wide variety of SLA parameters unambiguously and specify how these parameters are measured.

The WSLA framework consists of a flexible language based on XML and a run-time architecture comprising several SLA monitoring services. The WSLA document defines the agreed-upon QoS performance characteristics and the way to evaluate and measure them. It also describes the metrics and how these can be collected. Furthermore, it contains a section that describes what actions are taken when a violation is detected. Figure 8 illustrates an example of a WSLA service description.

The example below illustrates a service level objective given by a service provider and valid for a full month in the year 2001. It guarantees that the SLA parameter *AvgThroughput* must be greater than 1000 if the SLA parameter *OverUtilization* is less than 0.3; i.e., the service provider must make sure his system is able to handle at least 1000 transactions per second under the condition that his system is operating under normal load conditions for 70% of the time. Upon receipt of an SLA specification, the SLA monitoring services in the WSLA runtime architecture are automatically configured to enforce the SLA.

```
<ServiceLevelObjective name="Conditional SLO For AvgThroughput">
<Validity>
  <Start>2001-11-30T14:00:00.000-05:00</Start>
  <End>2001-12-31T14:00:00.000-05:00</End>
</Validity>
  <Expression>
    <Implies>
      <Expression>
      <Predicate xsi:type="Less">
        <SLAParameter>OverUtilization</SLAParameter>
        <Value>0.3</Value> <!-- 30% -->
      </Predicate>
      </Expression>
      <Expression>
        <Predicate xsi:type="Greater">
        <SLAParameter>AvgThroughput</SLAParameter>
        <Value>1000</Value>
        </Predicate>
      </Expression>
    </Implies>
  </Expression>
</ServiceLevelObjective>
```

Figure 8: Illustration of a Web Service Level Agreement

3.7.3 WSOL

A research group from Carleton University has developed a new XML-based language called Web Service Offerings Language (WSOL) [51]. It complements WSDL by providing various classes of Web Services, which is achieved by having different QoS constraints.

A WSOL file specifies the QoS constraints of the Web Services. A QoS constraint contains the service level objective (i.e. an objective that guarantees the service response time will be less than a certain number of seconds) along with the QoS metrics that are defined in an external ontology. Such an ontology contains precise definitions of how the QoS metrics are measured, and how the QoS metrics and measurement units relate to each other. The WSOL element *QoSmetric* is used for the declaration of a QoS metric in the WSOL file. This element contains the attribute *metricType* to refer to the ontological definition of the QoS metric. It also contains the attribute *measuredBy* to refer to the management party that performs measurements.

The overhead of measurement and calculation of QoS metrics for every operation invocation can be too high for some circumstances. An improvement of WSOL over WSLA is that WSOL provides the XML element *evalPeriod* to enable occasional evaluation of QoS constraints. This means that WSOL constraints checked before and/or after operation invocations can be evaluated occasionally for some randomly chosen invocations. For example, when the evalPeriod attribute is set to 5, this means that the QoS constraint is checked, on average, for one operation invocation out of five. This helps reduce the runtime overhead of monitoring activities, at the cost of quantity and precision of management information.

3.7.4 UDDIe

Another approach developed at Cardiff University, called "UDDIe" [45], extends the functionalities of UDDI to allow the UDDI registry to store QoS properties of Web Services. UDDI has a *businessService* class that represents general information about a single service offering. UDDIe extends this businessService class with a new *propertyBag* class. This propertyBag class contains additional Web Services QoS attributes such as price, network bandwidth, CPU, and memory requirements, which are encoded in the WSDL service interface.

In the UDDIe approach, the requesters first send their requests for services with the desired QoS properties to an intermediary known as the QoS broker. The broker processes the request and submits the service request portion to the UDDIe registry. Based on the stored QoS information, the UDDIe registry sends a reply with the list of services that support this particular query. Finally, the broker selects the most appropriate service by applying a weighted average measured by the closeness to the desired QoS properties and sends the result back to the requester. Moreover, in order to handle Web Services that change properties often, UDDIe supports the notion of a "finite" lease, where the service provider can define an exact period for which the service should be made available for discovery in the registry.

3.7.5 Using tModel in UDDI

The UDDI information model is composed of data structure instances expressed in XML. One of the data structures stored in UDDI is the *tModel*, which is used to describe the technical information about services. A tModel consists of a key, a name, an optional description, and a Uniform Resource Locator (URL) which points to a place where details about the actual concept represented by the tModel can be found. The primary role of a tModel is to represent a technical specification that is used to describe the Web Services. The other role of a tModel is to register categorizations, which provides an extensible mechanism for adding property information to a UDDI registry.

One solution proposed by Blum [9] is that the categorization of tModels in UDDI registries can be used to provide QoS information in *bindingTemplates*. In the proposal, a tModel for quality of service information for the binding template that represents a Web Service deployment is generated. Each QoS metric is represented by a *keyedReference* that is a general-purpose structure for a name-value pair in the generated *tModel*. The name of a QoS attribute is specified by the *keyName*, and its value is specified by the *keyValue*.

When a provider publishes a service in a UDDI registry, a tModel is created to represent the QoS information of the service. It is then registered with the UDDI registry. Each QoS metric is represented by a keyedReference in the generated tModel. The units of QoS attributes are not represented in the tModel. As discussed in the previous chapter, APIs for interacting with the UDDI registry, such as UDDI4J, can be used to facilitate the service publishing and update process.

3.7.6 Summary

The previous sections described different approaches for storing Web Services QoS information. Common characteristics include the use of XML and the conformance with existing Web Services technologies such as WSDL and UDDI. These efforts enable quality metrics of Web Services, and the associated service level objectives, to be described flexibly and meaningfully for the service client.

WS-Policy uses the simplest method for storing Web Services QoS information by embedding non-functional properties in the WSDL description, but it suffers from the fact that the WSDL file must be recompiled to obtain any updates to the non-functional attributes. The WSDL descriptions on the mobile clients generally are compiled statically, thus making WS-Policy unsuitable for mobile Web Services.

The approach adopted by WSLA and WSOL places the non-functional properties in a separate file so that changes can be isolated from WSDL. In both frameworks, the service requester needs to contact either the service provider or a third party in order to query for the latest QoS information. However, this approach places an extra burden on the service providers, and more importantly on the mobile environment, since the service requester might not always be able to connect to the third-party evaluation. These two issues can be resolved by having a discovery agent acting on behalf of the service requestor and keeping a local copy of QoS information for the service. Then the agent periodically contacts the service providers or third parties to update its local QoS repository. Nevertheless, it is unclear how the agent knows when the QoS information should be updated.

The most appropriate method for storing mobile Web Services QoS information is to use the UDDI registry. With this approach, the mobile clients can obtain the latest advertised QoS information directly from the UDDI registry, thus reducing the overhead on the service provider. The drawback with UDDIe is that it does not provide any mechanism for updating the QoS information. Nevertheless, APIs for UDDI are available for facilitating such QoS information updates. Therefore, the method used by this thesis is to adopt Blum's approach and store the QoS attributes that are important in the mobile environment in the tModel, and use the APIs provided by UDDI4J to update QoS information stored in the UDDI registry seamlessly with minimal disturbance to the clients.

3.8 What Should Be In The QoS Model?

In the presence of multiple Web Services with overlapping functionality, QoS attributes can be used to distinguish one service from another. This section presents the QoS model employed by this framework. The model can be used by clients to specify their non-functional requirements and service providers to distinguish their offers.

It is impractical to have a standard QoS model for all Web Services in all domains. The reason is because QoS is a broad concept that encompasses a tremendous number of non-functional properties. Moreover, each domain has specific QoS criteria. For example, important QoS

criteria in the mobile Web Services domain, such as network bandwidth fluctuation and device power consumption, do not have the same impact in the wired Web Services domain. Therefore, the framework includes a new QoS model with both generic and mobile Web Services-specific QoS attributes.

A wide spectrum of metrics which are considered to be important to Web Services QoS has been put forth by the research community with varying interpretations, such as the works by Garcia et al. [22]. Typical Web Services quality attributes are availability, security, performance, trustworthiness and monetary cost. These QoS aspects should be published in the QoS model used by this framework.

Obviously quality attributes related to the mobile environment are missing among the above. Therefore this QoS model also defines new criteria such as provider resource cost, device requirement and device power consumption for enhancing the mobile Web Services selection process. All the QoS metrics in the model are explained further below.

Availability is the quality aspect that represents the percentage of time when the service is ready for immediate use over an observation period. Web Services might be unavailable if the server overloads or malfunctions, or when the service application is unable to connect to other components such as the Database Server. This value is defined by the service provider under the *Service Availability Metrics.*

From the mobile client's perspective, factors such as wireless network performance and message data size can affect availability for a service. Mobile network performance can be measured by network parameters such as latency. Latency refers to the total time taken to complete a service request. Latency is an important quality factor because it affects the client's experience if the service takes too long to complete. A timer can be placed in the mobile device to measure the total time needed for completing a service request over the mobile network. The data is categorized under *Mobile Network Metrics.* On the other hand, the service provider can advertise the message size based on the output parameters and their data types as defined in the WSDL file. The message size can be measured dynamically by the client for calculating service providers' reputation and is categorized under *Message Size Metrics.*

Security is another significant aspect in service selection. The encryption algorithm and cryptographic key size are the two main aspects of security. Selection of the security encryption algorithm is generally performed with syntactic matching, and this is out of the scope of this thesis. On the other hand, cryptographic key size can be considered as a QoS attribute. If the cryptographic key supports a higher number of bits, then the connection is more secure, because each additional bit makes it exponentially harder to decipher the communications [24]. Security levels typical ranges from 80, 112, 128, 192, to 256 bits of security. This attribute is defined by the provider under the *Security Metrics*.

Performance measures the speed in completing a service request. Common performance metrics include service response time and throughput. Server processing time is defined as the time interval between the point when a request arrives at the server process and the point when the

server process sends the response. Throughput is the number of requests completed over a period of time, and lower server throughput often leads to loss of revenue. Bandwidth is the network transmission rate when delivering the service. These data are defined by the provider under the *Performance Metrics*.

Trustworthiness evaluates the degree to which an entity will provide a service as expected. When a service provider does not fulfill the client's requirements as expected, it is considered untrustworthy if it is able to but unwilling to do so. One way to measure a provider's trustworthiness is to evaluate its reputation. As discussed earlier, a reputation system is included in the framework to compute the difference between the suggested value and the actual value delivered for QoS metrics such as availability, server processing time and message size. A normalized value of the difference can be used to compare the reputations of service providers fairly. This value is provided by the reputation system and stored in the *Reputation Metrics.*

Cost is another quality property that influences service selection heavily. Obviously the monetary cost of purchasing the service can affect the client's service selection decision, thus it is included in the *Monetary Cost Metrics* for this category. The resource consumption cost for both the server and the device is another critical cost. The most severe overhead caused by mobile Web Services is to have a response that is too large for the pervasive device to handle. Such a service wastes airtime and service fee. For example, consider a scenario where there are two mobile Web Services: Service 1 offers a video with a resolution of 640x480, whereas Service 2 offers a video with a resolution of 320x240. Higher resolution provides better quality but transmits more data and requires more processing on the client's device. Worse, the device may only have a screen resolution of 320x240, making the cost higher as it must convert the video down to the correct resolution. Further, the server has a higher cost to transmit the larger video. Therefore, the selected service must be within the tolerable processing limits of the device; otherwise server computation, client computation, and network bandwidth are wasted. By including resource consumption cost in the model, service selection then becomes sensitive to mobile environment limitations such as device power and network bandwidth. Moreover, device heterogeneity can be expressed in terms of resource richness: given a service, it is cheaper for a powerful device than a less powerful one.

The server resource consumption cost for providing the Web Service can be measured by attributes such as CPU time and network bandwidth. CPU time is expressed as seconds, and bandwidth is measured by the network input/output activities in bytes per second. The CPU time introduced by a service can be measured by tools such as the Java Virtual Machine Tools Interface. The bandwidth consumption of a service can be obtained by monitoring the number of network bytes transmitted and received during the execution. These metrics are declared by the provider under the *Server Cost Metrics*.

In order to ensure end-to-end service interoperability, the mobile device's capabilities also need to be taken into consideration. A device profile which stores the device's attributes such as CPU power, available memory, supported video resolution, etc., can be sent during the service request to ensure the device has the minimum capability requirements to handle the Web Service response. For instance, a service provider that returns a service invocation message with a video attachment may require the mobile device to have at least 1 MB available memory. These data

are defined by the provider and grouped under *Device Requirement Metrics.* By knowing the client's *Device Requirement Metrics*, the service selection process will be able to perform requirements matching and warn the mobile client if the minimum requirements are not met.

Device Power Consumption is another important metric to consider. Power consumption of mobile Web Service matters when the service is invoked by the user on a streaming-basis. For example, a mobile user might invoke a location tracking service every few seconds to find his/her current location, and the accumulated usage of the service will play a significant role in consuming the power of his/her mobile device. The current energy capacity of portable computers can be obtained via the Advanced Configuration and Power Interface (ACPI) [1], which gives an estimate of the current capacity of the batteries. In practice this will not work because no individual service invocation will make a measurable difference in the energy consumption of the device. While creating the perfect method to measure mobile Web Services power consumption is beyond the scope of this thesis, there are several approaches that might be taken to estimate mobile Web Services power consumption. For example, the study by Oh [39] suggested that processing XML messages is the largest overhead for mobile device, and thus XML message size can be used to estimate the mobile device's power consumption. On the other hand, the study by Batra et al. [7] discovered that although different types of mobile Web Services consume the device battery at different rates, the dominating battery-draining factor of using mobile Web Services is the transmission and processing of data. Therefore they suggested a solution which uses the service response time and response processing time to estimate the units of power consumed by the device for interacting with the service. Service response time is the total time spent starting from request composition, invoking the service, and getting the response back on the device. It includes the time spent on the network and service. Response processing time is the time taken by the mobile device to extract and process relevant information from the response. This attribute does not measure power consumption as an absolute quantity, but quantifies the units in such a way that can be used for comparison. Both values are measured dynamically by the client application and are assumed to have similar power consumption rate. This thesis adopts the method suggested by Batra to estimate mobile Web Service power consumption. Thus the device's power consumption for interacting with the service is defined by:

P = Service response time + Response processing time

A remaining challenge in defining the QoS model is to ensure compatibility of all measurement units between a service client and service endpoint. For example, mobile Web Services do not support Java Collection types, which means the mobile Web Services clients will probably fail to generate stub files from a well-formed WSDL file. In order to address this issue, the framework adopts the proposed rules suggested by S. Fang Rui [20] by using only preferred data types for the measurement units.

All the above QoS metrics along with their measurement units are summarized in Table 2. Except for *Mobile Network Metrics* and *Device Resource Cost Metrics* (*Device Profile Metrics* do not require any computation), all the other metrics are defined at the server, thus saving precious CPU resources of the mobile device.

3.9 How To Ensure Service Quality Fully Complies With The Description?

Another challenge faced by QoS-aware mobile Web Services discovery is that QoS characteristics published by service provider might not be reliable. Service providers may not predict service quality in a neutral manner. Also, service providers tend to overstate the real QoS and do not intend to constantly revise systems to provide recent quality parameters. Consequently, this solution is not effective and trust-aware. How to measure QoS compliance effectively is another question raised by this thesis. The next section examines several research works which vary in terms of the tradeoffs between up-to-date, secure QoS evaluation, and computational overheads. They can be divided into two groups:

1. Solutions that rely on service clients to review service quality.
2. Solutions that rely on a third party to evaluate the Web Services.

Category	**Metrics**	**Defined by**	**Value**
Service Availability Metrics	*Availability*	Provider	0
Mobile Network Metrics	*Latency*	Client	Milliseconds
Message Size Metrics	*Message Size*	Provider	Kilobytes
Security Metrics	*Confidentiality, Authentication*	Provider	Boolean
Performance Metrics	*Throughput, Server processing time, Bandwidth*	Provider	# of requests/second, milliseconds, kilobytes/second
Trustworthiness Metrics	*Trustworthiness Reputation*	Reputation system	[0..1]
Monetary Cost Metrics	*Monetary Cost*	Provider	Dollars
Server Resource Cost Metrics	*CPU time, bandwidth cost*	Provider	seconds, # of bytes/second
Device Resource Cost Metrics	*Device Power Consumption*	Client	Units of Power
Device Requirement Metrics	*CPU Power, Memory, Video Resolution*	Provider	Mhz, RAM, Pixels

Table 2. QoS Model

3.9.1 Measuring QoS Compliance By The Clients

In this method, the client keeps track of the provider's previous performance and assesses the compliance with the stated levels of the quality attributes. Common approaches include gathering user ratings to establish a reputation for the provider, and monitoring Web Services QoS data through a user agent.

Reputation is often used to measure QoS compliance and facilitate dynamic Web Services selection, as seen by the works of Lie et al. [31] and Herlocker et al. [26]. Reputation is measured as an average of the user ratings that are given by service clients after each use. The reputation of each service provider is provided to clients to aid service selection. However, in those schemes reputation is heavily influenced by user perception and can be manipulated easily. Kalepu et al. [28] proposed a new framework to measure reputation more accurately by coupling the subjective user perception with the objective view of performance history. It introduces a new QoS metric called "verity" to indicate the trustworthiness of the service provider. "Verity" is the degree of consistency exhibited by the service provider in delivering the quality levels laid out in the service contract. It calculates the variance between the promised and actual value for a QoS metric which is measured by clients over a range of previous transactions. Reputation is then expressed as a weighted sum of user rating and the verity value.

There has also been research on using software agents to automate the QoS data collection process and aid service selection. For instance, Maximilien and Singh [33] proposed a conceptual agent framework for dynamic Web Services selection. The agents collect QoS data dynamically, so that the actual service quality can be determined collaboratively. Moreover, service agents can share their past experience of using the service to help establish a reputation for the provider. The agents act on behalf of the client to evaluate the providers based on the reputation score.

The main disadvantage of having the clients to measure QoS compliance is that a client cannot know the reputation before interacting. Furthermore, this has a limited basis of trust. Moreover, using software agents to measure QoS compliance requires a substantial amount of computation which might not be appropriate for mobile clients, considering they are typically resource constrained.

3.9.2 QoS Compliance Evaluated By Third Party

The third party is typically a specialized unbiased agency that tests published QoS information or stores collective feedback. As discussed earlier, frameworks such as WSLA and WSOL already allow a third party to be specified for measuring QoS compliance. The following describes various kinds of third-party models.

Ran [41] proposed a model in which the traditional service discovery model is extended with a new role called the "Certifier," in addition to the existing three roles of Service Provider, Service Consumer, and UDDI Registry. The Certifier verifies the advertised QoS of the Web Service before its registration. The consumer can also verify the advertised QoS with the Certifier before

binding to the Web Service. Furthermore, the unbiased Certifier can prevent service providers from publishing invalid QoS claims during the registration phase, and help consumers verify the QoS claims to assure satisfactory transactions.

There are three drawbacks with the above model. First, it lacks flexibility as it requires all Web Services providers to advertise their services with the Certifier. It also lacks the ability to meet the dynamics of a market place where the needs of both consumers and providers are changing constantly. For example, it does not have methods for providers to update their QoS dynamically. Furthermore, the certifier is restricted to performing only trivial operations, and not carrying out any transactions.

Sheth et al. [46] proposed a QoS middleware infrastructure which contains a built-in tool to monitor QoS metrics automatically. The tool simply polls all Web Services to collect metrics of their QoS on a timed interval. This solution is expensive and inefficient because of its rather static nature. Moreover the polling interval needs to be tuned carefully. If the polling interval is set too long, the QoS information becomes stale. If the polling interval is set too short, it might incur a high performance overhead.

Wishart et al. [59] designed a method for computing a reputation score based on an aging function. It first calculates the normalized difference of each QoS metric, ΔM_i, using (1).

$$\Delta \mathbf{M}_{\boldsymbol{i}} = \frac{\text{actual value} - \text{predicted value}}{\text{actual value}} \quad (1)$$

This normalized value allows the system to compare (within some limited bound) different metrics fairly. The normalized difference of each QoS metric is used to calculate the average normalized difference A_i using (2).

$$\mathbf{A}_{\boldsymbol{i}} = \frac{1}{n}\sum_{i=1}^{n} | \Delta M_i | \quad (2)$$

where n is the number of QoS metrics measured by the client. After the system has collected sufficient feedback from various service clients, it calculates the reputation score by (3)

$$q_{rep} = \frac{\sum_{i=1}^{f} A_i \lambda^{d_i}}{\sum_{i=1}^{f} \lambda^{d_i}} \quad (3)$$

which determines the average ranking given to the service from the clients. In the equation above, f is the number of feedbacks for the service, λ is the aging factor, $0 < \lambda < 1$. The aging factor λ adjusts the responsiveness of the reputation score to service changes. When λ is set

close to 0, the more recent feedback has greater weight. On the other hand, d_i is the number of hours elapsed between the two times t_c and t_i. t_c is the current time when the reputation score is computed, and t_i is the time when the feedback was measured. The exponent d_i adds more weight to more recent measurements.

The advantages of this approach for finding a service provider's reputation is its high extensibility and customizability, light computation overhead, and the fact that it takes various service clients into consideration. Other issues related to the reputation system, such as security and enforcing honest reporting are beyond the scope of this thesis.

3.9.3 Summary On The Methods For Ensuring QoS Compliance

The third-party approach might not be appropriate for the mobile environment as it poses more restrictions, such as requiring a reliable network connection between the mobile client and the third party. On the other hand, depending solely on user rating is not trustworthy enough because it is subjected to bias by the user.

The easiest approach to ensuring QoS compliance is to record achieved service levels once a transaction has been completed. This gives an insight into the provider's past performance by providing necessary data to assess the compliance levels over a range of past transactions, but consumes precious storage space on mobile clients. A more effective solution is to take a page from Wishart et al. [59]; i.e. for each QoS metric, compute the difference between the suggested value and the actual value.

3.10 How to Take Advantage of the QoS Information?

Many researchers have investigated how Web Services discovery and selection can be enhanced by QoS. Generally, the goal of the existing approaches can be divided into providing better service selection for the user, or improving resource allocation.

3.10.1 Better Service Selection

Luo et al. [32] describe a mathematical QoS vector model based on four QoS attributes: availability, performance, accessibility, and accuracy. The values of these four attributes can be used in a more comprehensive evaluation of QoS. It introduces a weighting factor for each attribute so that consumers can prioritize these attributes according to different requirements. Then it formulates a QoS-Vector which includes the mean and variance of the selected attributes. The "mean" is the average difference between the expected and delivered performance of all services of a service provider. The lower the mean, the less discrepancy between the expectation and actual delivery of the service, and the more satisfying performance the service provider provides. Variance refers to the degree of deviation from the expected performance for a certain service provider. The smaller the variance, the more consistent the service provider is in terms of its performance. The selection of service provider is based on the choice of the best mean and variance among the QoS-Vectors.

Wang et al. [56] disclosed a QoS selection model for Web Services. The user provides requirements, including non-functional, functional, and quality properties, which are formed into a requirement profile; a first filter determines matches of the profile with advertised services, and a second filter considers all quality features to select the service best matching the user's requirements.

Vu et al. [53] have implemented a service discovery solution that enables personalization by using Web Services QoS. They presented a model for the users to describe their QoS selection criteria semantically, taking into account the environmental conditions specified by the providers in their service descriptions via description logics. The QoS-enabled discovery process can be done autonomously by reasoning on the constructed knowledge bases and the various personalized matching criteria and preferences of the users.

Kritikos and Plexousakis [30] devised a semantic QoS-based description and discovery of Web Services to select services with more precision. They proposed an ontology for QoS-based Web Services description called OWL-Q. Both the requester and service provider use three common ontologies: unit, QoS property and QoS value-type. The unit ontology describes the unit used for the measurement prescribed by a QoS metric. The QoS property provides semantic descriptions of domain-independent (e.g., throughput, availability, and response time) and domain-dependent QoS properties (e.g., flexibility of reservation changes in the travel domain). The third ontology is used for simple QoS datatypes (integer, real, string). Their semantic matching algorithm uses a set of complex rules to perform service matching based on the above ontology to rank services that better fulfill the client requirements.

Fedosseev [21] proposed a QoS-aware Web Services composition and selection technique. A Web Services quality model is created based on a set of quality criteria of the Web Services, such as pricing, execution duration, reputation, availability and reliability. It models Web Services composition as different execution plans, and QoS score is used to identify an optimal execution plan. The overall QoS score for each execution plan is calculated by adding each of its individual service component's QoS score. For example, the QoS score for price for an execution plan is the sum of each service's execution price; the QoS score for reputation of a composite service is the average of each service's reputation which is given by the end user. The execution plan which has the highest QoS score is chosen. Constraints can also be defined for each QoS attribute to filter execution plans. However, it does not describe how it prevents the QoS score from becoming skewed by abnormal data values.

3.10.2 Improving Resource Allocation

Another benefit of Web Services QoS is that service providers can offer multiple classes of service. Class of service has been used widely for telecommunications service provisioning. It has been applied extensively in telecommunications technologies such as Differentiated Service (DiffServ) and Telecommunications Information Networking Architecture (TINA). However, the QoS metrics defined by those technologies, such as packet loss rate, delay and jitter, are

typically at the communication level, making it difficult to translate them directly into the mobile Web Services domain.

Nevertheless, the benefits that class of service can bring to mobile Web Services cannot be ignored. When a service provider simultaneously serves a large number of different clients in parallel, it often has to provide various levels of QoS to accommodate different characteristics and needs of its clients. Offering multiple classes of Web Services is a lightweight approach for service providers to address various requirements for different consumers. "Class of service" for Web Services means providers offer services with the same WSDL functional description, but the services vary in terms of QoS provided such as response time and availability. The aforementioned WSOL framework is an example of a research prototype that allows providers to advertise multiple classes of Web Services by specifying different QoS.

Tian et al. [50] implemented a framework that uses classes of services to protect web servers hosting Web Services from overloading. They proposed a simple scheme that allows clients to specify whether they want to receive compressed data when requesting a Web Service. Depending on the current server load, the server compresses only the requests of the clients that required such a service. In their framework, the users decide among three options: do not compress the response, compress the response, or compress the response if possible. If users choose the last option, the server is free to choose what the server considers best. When the last option is chosen and the server demand is low, the server compresses the responses to all clients that have asked for compressed replies and to those clients that have not specified a preference. During high server demand, the server compresses only responses to clients that have asked for compressed data.

Yu and Lin [60] presented a proposal for guaranteeing the level of service quality delivered to different clients. At first all clients send their service request and QoS requirements to a broker, which decides how much resource the provider should assign to the clients to meet their QoS needs. The basic idea of the allocation algorithm is to create a virtual client to reserve some unused system resources. For every incoming client request, if the reserved resource is enough (above the defined threshold), the broker allocates the requested amount of resource to the new client directly. If the reserved resource is not enough, the broker reconfigures the resources allocated among some existing clients with lower service level agreement to let the incoming client receive a satisfactory service quality. Its purpose is to effectively adjust system resources, while ensuring clients with higher service quality requirements will not experience unstable performance.

3.10.3 Better Pricing Strategy

Cao et al. [12] created a QoS model to do some studies on applying game theory for service pricing. They argued the Stackelberg model pricing approach results in unfairness because the service provider can deduct the best profit by charging a high price, leaving the client almost no profit at all, and vice versa. They recommended the Nash bargaining approach where the service provider and clients negotiate for a fair pricing point between their two utility curves. In practice, this bargaining approach is difficult to enforce without a third party regulator, but

nevertheless the study showed how the QoS model can be used for determining optimal service price.

3.11 Concluding Remarks

The above sections examined existing work on mobile Web Services discovery methods and different topics in Web Services QoS, in particular on methods of storing QoS information, mechanisms for ensuring QoS compliance, and frameworks that take advantage of QoS. In general, all common mobile Web Services discovery mechanisms lack adequate QoS support, and this is preventing the adoption of performance-sensitive mobile Web Services. Hence there is a need to design a new QoS-aware Web Services discovery framework for the mobile environment.

As shown by [50], offering classes of service is an effective solution for handling system resource fluctuations. It is foreseeable that classes of service can also help deal with frequent fluctuations of resource availability in the mobile environment, which either happens locally (e.g., battery is running out) or in the surrounding environments (e.g., received signal is getting weaker).

Meanwhile, existing QoS-aware Web Services discovery does not provide enough awareness to balance the requirements of both the clients and providers. Previous work tends to focus solely on fulfilling the requirements of either the client or the provider. However, the interests of the provider and client are different. The provider's goal is to choose the service that maximizes the profit. One possible method is to realize the best QoS for the client at the lowest resource cost. This is particularly important in the mobile environment where wireless network bandwidth is precious. On the other hand, the client's selection is determined by their own requirements which do not always reflect the real resource overheads.

Moreover, many existing QoS selection algorithms apply normalization to scale the quality values for each candidate service to find the service that is closest to the quality requirement. The drawback with this quantitative approach is the rigidness in the requirements. The degree of satisfaction of the client with a particular quality value cannot be measured simply by the difference between the required and actual value. For example, a client might tolerate a lower bandwidth value as long as it is greater than a certain threshold. The QoS selection algorithms using normalization are unable to reflect this kind of flexibility. Furthermore, the QoS score might be skewed by abnormal data values.

The novel Web Services discovery framework presented in this thesis resolves the inflexibility and skewing issues with normalization by using utility functions to evaluate QoS score. Moreover, it takes both the client's and provider's perspectives into consideration during service evaluation. The next chapter presents the design of this framework.

Chapter 4
Design of the QoS-aware Service Discovery Framework

This chapter presents the design of a new QoS-aware mobile Web Services discovery framework that addresses the aforementioned drawbacks. The contributions of this framework are:

1. to give an extensible QoS model for mobile Web Services,
2. to provide a lightweight method for measuring QoS compliance, and
3. to propose a comprehensive utility function for determining a service instance that best satisfies both client and provider needs.

The rest of the chapter is organized as follows. Section 4.1 presents a high-level overview of the framework. Section 4.2 describes the method used for storing QoS information. Section 4.3 examines the service broker. Section 4.4 describes the service discovery request. Section 4.5 details the service selection process that includes a utility function to balance the requirements between the clients and providers.

4.1 High-Level Architecture

The traditional Web Services discovery model consists of three components: service provider, service consumer and UDDI registry. The QoS-aware mobile Web Services discovery framework modifies the above model by introducing two new components: a reputation manager and a Web Services broker, and some enhancements are made to the UDDI registry.

In this framework, the Web Services broker acts as an intermediary between the consumer and the UDDI registry, and is responsible for discovering Web Services that satisfy the consumer's functional, QoS and reputation requirements. The reputation manager measures the Web Services' QoS compliance and provides a reputation score when requested by the broker. The UDDI registry is enhanced so that advertised QoS information is stored in it. A service broker is added for selecting the best service on behalf of the client. Figure 9 illustrates this new model. The white boxes represent the existing components in the typical discovery model, and the shaded boxes are the new components. The details of each new component will be elaborated further in subsequent sections.

4.2 Storing QoS Information in UDDI

The Web Services QoS attributes must be published at a location that is accessible by both client and provider. As discussed earlier, UDDI already supports the publication and discovery of service providers, their Web Services, and the technical interfaces for the clients to bind with the services. By using the solution proposed by Blum, the tModel of UDDI can be extended to include the QoS description. The system assumes the tModel is not cached and it is continuously being updated.

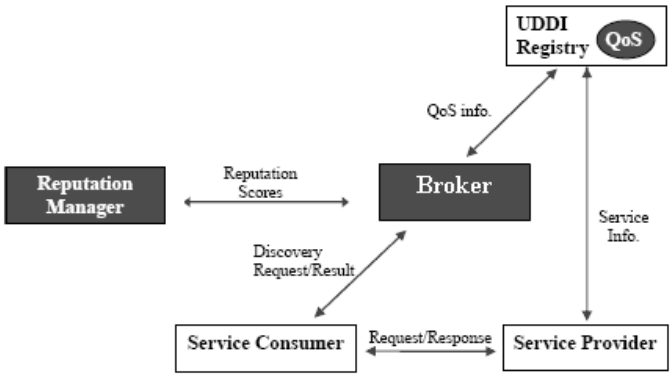


Figure 9: QoS Aware Mobile Web Services Discovery Model

Figure 10 shows an instance of the tModel structure containing QoS information. This example illustrates a tModel used to specify a QoS attribute. The specified attribute is the service throughput. The *overviewURL* element points to a file storing the attribute definition. *CategoryBag* represents the attribute properties. The tModel with the key specifies the metrics. The measure unit is milliseconds, which is specified using a separate tModel.

```
<tModel
  tModelKey="uddi:uddi.org:qos:attribute:throughput">
    <name>uddi-org:qos:attribute:throughput</name>
      <description>Performance attribute specification</description>
        <overviewDoc> http://<URL describing schema of QoS attributes></overviewDoc>
          <categoryBag>
              <keyedReference keyName="QoS attribute specification"
                              keyValue="qosAttributeSpec"
                              tModelKey="uddi:uddi.org:categorization:types"/>
              <keyedReference  keyName="Performance metrics"
                              keyValue="throughput"
                              tModelKey="uddi:uddi.org:qos:metrics:throughput"/>
             <keyedReference keyName="Throughput unit"
                             keyValue="millisecond"
                             tModelKey="uddi:uddi.org:qos:unit:millisecond"/>
          </categoryBag>
</tModel>
```

Figure 10: Code Fragment of QoS tModel

4.3 Service Broker

Borrowing from the idea of using a DII by Nielsen, this framework contains a service broker that acts as an intermediary between the service requestor and the service provider. The broker can be viewed as a remote Web Service for the clients to obtain the access point of the most appropriate service. It interacts with the UDDI server and the service provider over the fixed network and returns the results to the mobile device over the wireless network. Not only does the broker provide dynamic service selection similar to the dynamic proxy entity, it is also responsible for finding Web Services that meet the requestor's requirements. By having a service broker, much of the workload is processed by the broker, thus relieving time-and-processor consuming Web Services tasks from the mobile device. Furthermore, when the network is busy, multiple costly network trips can occur during the interaction with UDDI. This is troublesome in the wireless networks, as the unavailability of the network may hinder the completion of the user request. Since the broker interacts with the UDDI server through the fixed network, the significance of the above issue is much reduced. Finally, the broker can be used to handle the fact that mobile devices non-deterministically lose network connectivity much more than wired applications. It keeps the results of the service invocation and forwards them to the mobile device when the connectivity is re-established.

The steps performed by the service broker during service discovery are:

1. The client first sends a service inquiry that includes functional and quality requirements to the broker. The functional requirements are specified with keywords of service names and descriptions. Section 4.4 will describe the details of how QoS requirements are specified.

2. Upon receiving the inquiry, the broker will then consult one or more UDDI registries. WSDL files for services are then checked and available offers are built.

3. The broker then applies the service selection process on the newly created offer list to find the service that best matches client requirements while also being least costly for providers. The details of the service selection process are elaborated further in a later section.

4. The broker requests the service description (operations provided, parameters, etc.) from the provider, and this information is forwarded via an encoded XML message to the client application that resides at the mobile device. The client processes the information and displays it to the mobile user.

5. The client invokes the service based on the information (Web Service name, selected operation, parameter values, etc.) sent by the broker.

Currently, the framework supports a single instance of a service broker. The advantage is that only one stub class (corresponding to the service broker) is required on the client applications

residing on the mobile devices. Multiple client applications can interact with this service broker running within their network domain.

4.4 Service Discovery Request

The client specifies his/her functional and QoS requirements in the service discovery request. Figure 11 shows the SOAP message for a discovery request. As the figure shows, service name and description are used for describing the client's functional requirements. The *qualityRequirement* category allows a client to specify his/her desired values for the QoS metrics, such as the maximum monetary price, requirements on security, availability and throughput, device hardware profile, etc. The client application should preset some of the QoS values, so that the user does not need to generate all the QoS attributes for each request manually. The relative importance of each QoS attribute is specified in the *weightedFactor* section. Finally, the client can specify the maximum number of services returned in the section *maxNumberService*.

The primary interest in this framework is not in matching functional requirements, but rather in matching the QoS attributes during dynamic service discovery. QoS service matching provides a more feasible alternative compared to semantic service matching for overcoming the limitations of syntactic service matching. It is achieved by differentiating service providers and ensuring the service reaches the client's standards in performance, security and availability.

```
<?xml version="1.0" encoding="UTF-8" ?>
 <envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/"
   <body>
      <find_service generic="1.0" xmlns="urn:uddi-org:api">
            <functionalRequirement>
              Keywords of service name and description
            </functionalRequirement>
            <qualityRequirement>
              <QoS attribute 1>Availability</QoS attribute 1>
              <QoS value 1>0.98</QoS value 1>
              <weightedFactor 1>0.4</weightedFactor 1>
              <QoS attribute 2>Security</QoS attribute 2>
              <QoS value 2>128</QoS value 2>
              <weightedFactor 2>0.6</weightedFactor 2>
              </qualityRequirement>
            <maxNumberService>5</maxNumberService>
       </find_service>
    </body>
 </envelope>
```

Figure 11: SOAP Message Service Discovery Request

4.5 Service Selection Process

Once the broker receives the service query, it applies the service selection method to find the best service, taking account of the client's preferences, overall service resource consumption cost, and service reputation.

The following steps are performed during the service selection process:

1. Find services that meet the customer's functional requirements
2. For each service that meets the customer's functional requirements
    a. Find the service entity representing the service in the UDDI registry with the service key
    b. Find the tModel representing the QoS information for the service
    c. Add the service key to the service candidate list if the service's QoS information in the tModel meets the customer's QoS requirements
3. Rank the services in the candidate list based on their utility score, select and return the specified number of services

4.5.1 Matching Functional Requirements

The UDDI catalogue stores the textual description of each Web Service along with the tModel that provides the service functionality. In order to perform service matching, a service query consisting of keywords is sent to the UDDI registry to find matches in the stored descriptions.

The client specifies his/her functional requirement in the service request with a number of keywords describing the desired functionality. Options are presented to the client for customizing how keyword matching is performed, such as case sensitive/insensitive, use of wildcards and exact match. An array of matching Web Services overview information (keys, names and descriptions) is then returned by the UDDI to the broker as the search result. Afterwards, the broker uses the service key to access the full registered details of the Web Service stored in the tModel.

4.5.2 QoS-Based Service Selection

Syntactic service matching might possibly yield several similar services. Without an intelligent service matching process, the client will be forced to make the selection manually, typically by browsing each service to find which one really suits his/her needs. This is tedious and cumbersome and therefore the framework includes an automated service selection process.

Generally, the purpose of service selection is to match user requirements against advertised capabilities of service providers. In this framework, the service selection not only handles matchmaking based on the service's functional properties but also on the non-functional properties, since qualities of service can influence the service selection decision heavily.

The steps of QoS-based service selection in this framework can be summarized as follows. For each of the Web Services that fulfill the user's functional requirements, the QoS information is retrieved from the tModel registered in UDDI. First the QoS constraints are examined to filter out unwanted services. Then the synthetic QoS score of each remaining service is calculated according to the client's preference and provider's cost. Finally the broker returns a set of suggested services to the client ranked based on overall QoS evaluation. The detailed metric information may also be included in the response.

The first step in service selection is to eliminate services that do not satisfy the QoS requirements. At first the client specifies his/her quality requirements for the expected service, along with its mobile device profile in the *qualityAttribute* section of the service discovery request. After obtaining the QoS information for each candidate Web Service from the UDDI, the broker examines whether the QoS advertised by the provider meets the QoS requirements defined by the client.

The filtering algorithm examines whether the mobile device meets the hardware requirements of the service. Information about the CPU processing power, memory and video resolution of the device are compared with the advertised requirement of the Web Service. If any of the device's attributes is less than the requirement suggested by the provider, then the service is eliminated from the candidate set. Afterwards, the broker employs a set of utility functions to evaluate the overall service utility. The details of the utility functions are explained below.

4.5.3 Utility Function

The QoS model includes metrics from the perspectives of both the users and providers. The main purpose is to balance the interests of users and service providers. This objective is achieved by the utility function explained below.

The utility scores for the client and provider both contribute to the evaluation of a service through the utility function (4).

$$\text{Aggregate Utility} = \text{User Utility} \times w_1 + \text{Provider Utility} \times w_2 \quad (4)$$

where $w_1 + w_2 = 1$. Since both user utility and provider utility fall into [0..1], therefore service utility falls into [0..1].

The utility function introduces two weights w_1 and w_2. The broker can customize these two values to achieve different goals. For example, by setting $w_2 = 1$, the broker will find the service that has the highest provider utility. Conversely by setting $w_1 = 1$, the broker will find the service with the highest user utility. A third variant is finding a tradeoff between user utility and provider utility, which can be realized by giving w_1 and w_2 various values other than 0. The flexibility in expressing the preferences makes the above utility function adaptive to different environments. Moreover, the end users do not need to understand the details involved in optimizing his/her resources and overall system resource consumption.

The values of user and provider utility are calculated by adding all utility scores for the appropriate QoS attributes. Using the above utility function, the services are ranked according to the highest score. The service with the highest utility is ranked first. If there is a tie, the winner is chosen randomly among the services with the highest utility. The subsequent section will explain the details of the utility function for each individual QoS attribute.

4.5.4 Utility Functions for Evaluating QoS Attributes

As discussed earlier, user utility and provider utility are calculated using several utility functions that act as selection criterion. Each utility function represents the degree of satisfaction with a particular QoS attribute, and the utility-function score ranges from 0 to 1. The criteria used to determine user utility include price, availability, reputation, throughput, device power consumption, server processing time, bandwidth, video resolution and security. On the other hand, the criteria used to determine provider utility include price, message size, CPU/memory consumption on the server, and bandwidth.

The utility functions fall into three categories: benefit curve, cost curve, and step function. QoS criteria such as availability, reputation, and throughput are considered to be positive attributes, where a higher value means a higher utility score. They belong to the benefit curve category. Meanwhile, QoS criteria such as price, message size, device power consumption, and CPU/memory consumption on the server are considered to be negative attributes, where higher value means lower utility score. They belong to the cost curve category.

The utility function associated with security belongs to the step function category, where the utility increases as the security level supports a higher number of bits. It is equal to 0 when it does not meet the minimum security requirement, and it steps up by a discrete value as the security level increases, until it reaches the maximum security requirement where the utility value will be equal to 1.

For many QoS criteria such as throughput, bandwidth, server processing time, and video resolution, the method to estimate utility score is unique and should be customizable by the user. The reason is best explained by example.

Consider Figure 12, which shows two examples of utility functions associated with throughput for quite different applications. The utility function for a voice playback using an audio playback codec is described by the $UAudio(x)$ curve. According to the Codec Mean Opinion Score (MOS) [14], which is a quality of speech measurement, on a scale of 1 (bad) to 5 (excellent), a transmission rate of 8 kbit/s gives a MOS score of 3.27, a rate of 16 kbit/s gives a score of 3.61, a rate of 32 kbit/s gives a MOS score of 3.85, and a rate of 64 kbit/s gives a MOS score of 4.1. The MOS score for transmitting voice at less than 8 kb/s is dramatically lower.

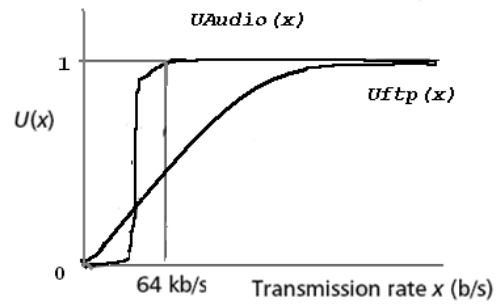


Figure 12: Utility Functions Associated with Throughput for Different Applications

As the graph of UAudio (x) shows, the utility function does not increase much above 64 kb/s, indicating there is little utility to be gained by transmitting faster than 64 kb/s.

With the second example, *Uftp*(*x*), which represents a file transfer application, the utility function reflects a different requirement. Naturally, the user is happier the faster the file is transferred, so the utility function shows how the utility increases as the transmission rate is increased. However, note that the user will be significantly more pleased if the file is transferred within a few seconds, but there is no benefit in transferring faster than the rate at which the receiving computer can process the data. As a result, the utility increases steeply until an acceptable transmission rate is achieved, and continues to increase above this transmission rate, but by a diminishing amount.

As shown by the above example, utility functions for certain QoS criteria need to be flexible in order to meet the user requirements accurately. Moreover, the utility functions are not always linear. Nevertheless, typically users may not be able to express their preferences in terms of complex mathematical functions, therefore the framework first elicits and expresses users' preferences in terms of a set of discrete data points, and then uses ones of the numerous curve-fitting techniques (e.g., regression, interpolation) to determine a function that approximates the data points most accurately.

4.5.5 User Utility

The user utility score helps separate the candidate Web Services into two categories: the ones that satisfy the QoS-based request completely and others that satisfy the request partially. Since different clients may have different QoS preferences, therefore the computation of user utility score must take account of this factor with relative importance. Relative importance is used to describe how critical a resource is to a host, to represent the priority of a QoS attribute to a service client. For example, a less patient client can give a higher relative importance to service processing time than other QoS dimensions to express its preference for fast services. Moreover, if the reputation of the provider, power consumption cost on the device, or the mobile network metrics are available, then they should be taken into account when evaluating user utility.

Finally, based on QoS values and relative importance of the client-specified QoS requirements, the broker computes the overall user utility score of the service to the client using (5).

$$\textbf{User Utility} = \sum_{i=1}^{n}(s_i \times w_i) \qquad (5)$$

where n is the total number of QoS attributes evaluated, s_i is the utility function score for each QoS attribute, and w_i denotes the client's relative importance assigned to the attribute. The sum of the w_i is equal to 1, and the value of each s_i is in [0..1].

Each utility score s_i is multiplied by its corresponding weight w_i to generate the user utility score of the service. If all QoS advertisements of the service match the QoS requirements of the user, then it will have a user utility score of 1. For those services that partially satisfy the user's demand, the user utility score will be between 0 and 1.

4.5.6 Provider Utility

The QoS metrics that contribute to the provider's resource cost and profit, such as message size, CPU/memory consumption on the server, bandwidth and price, are all factored in when calculating the provider utility. Furthermore, as various attributes may bear different importance to the service provider, relative importance is used to characterize the criticality of the various resources. Provider utility thus can be derived from the consumption of each resource and its relative importance by (6).

$$\textbf{Provider utility} = \sum_{i=1}^{m}(d_i \times w_i) \qquad (6)$$

where w_i refers to the relative importance of resource i specified by the provider. The sum of the w_i is equal to 1, and the value of utility score d_i falls into [0..1].

Provider utility is inversely related to the overhead for a service provider to host a service, but this utility is not necessarily published by the service provider, depending on the service provider's strategy (cooperative or not). There are several advantages for the service provider to co-operate. First, the utility function helps the service provider in situations when it is faced with excessive workload. For instance, if the provider is running low on resources and it needs to provide multiple services, then the provider must arbitrate which service is given preference and receives additional resources, and from which service they are taken. In order to be able to arbitrate, the provider can use the utility function to compute the marginal benefits of allocating resources to one service or another.

4.6 Reputation System

Reputation is a general and overall estimate of how reliably a provider services its consumers. Compared to trust, which is the willingness to depend on something or somebody in a given situation with a feeling of relative security, reputation is a public score based on public information while trust is a private score based on both private and public information.

Even if service consumers can obtain QoS advertisements from service providers in a service registry, one cannot be assured that the services found in the discovery process actually perform as advertised. However, with a reputation system, ratings of services can be collected and processed, and reputation scores of services updated. The reputation score can be used as a factor when ranking services. This improves the possibility that the services that best meet user needs are selected, and ensures that the selected services are reliable.

Hence a reputation system is proposed in this service discovery framework. A QoS reputation score is calculated based on feedback by service clients to the reputation system. Since different users have different interpretations of the perceived performance of a certain service, some kind of quantitative data is needed for finding the reputation of the provider. The system collects data measured by the service clients, processes the data, and then updates the reputation score for related service providers. The discovery of services in terms of QoS requires an accurate evaluation of how well a service can fulfill a user's quality requirements. For this estimation, the reputation system exploits data from two information sources:

1. QoS values promised by providers in their service advertisements, and
2. Service users submitting their feedback on the measured QoS of the consumed services.

The reputation score computation is based on the aging function approach suggested by Wishart et al. [59]. For the QoS metrics measurable by the client, such as availability, message size and server processing time, the reputation system computes the difference between the predicted value and the actual value delivered. The predicted value is the value of the QoS metric advertised by the provider, and the actual value is the QoS metric measured by the client during service execution. This requires interfaces used by all service clients to implement some mechanisms to log the actual execution time. Although this approach puts more burden on the service client, it has the following advantages over approaches where the service broker or UDDI registry is required to perform the monitoring: 1) it lowers the overhead of the UDDI registry and simplifies its implementation, 2) data is collected from actual consumption of the service which is recent and objective, 3) it avoids the necessity to install expensive middleware to poll the large number of service providers constantly. Through active monitoring, when a service client gets a worse set of values for the QoS criteria advertised by the service provider, this difference can be logged. The larger the difference, the lower the reputation score will be for that service provider.

The reputation system assumes the service is used on an ongoing basis. Once the system receives the feedback from the client, it is stored in its local database. The feedback consists of a service key, client key, timestamp and the actual values. The service key of the service stored in the UDDI registry is used as the service key, and the IP address of the service client is stored as the client key. Since a service can change its behavior over time, old experiences become irrelevant for the actual reputation evaluation. This calls for a discounting of older experience. Therefore a timestamp is used to determine the aging factor of the particular metric. If the particular measurement is older than a certain threshold, it is discarded.

4.7 Concluding Remarks

In this chapter, a framework is proposed for service selection which includes evaluating each service with a utility function, taking into account service resource consumption cost and client preferences in a comprehensive manner. QoS metric acquisition, QoS evaluation and QoS-based service selection is a complex interactive process involving participation of the service consumer, service provider, service broker, and the UDDI server. The QoS information is stored in the UDDI server where it can be accessed and updated easily. The service broker acts as an intermediary between the client and the provider. It finds services that meet the requirements in the request, ranks the services using their QoS scores and their resources utilization cost on the provider, and returns the candidate service set to the client. The client can select how many candidate services he/she wants to have returned by the broker. The next chapter will elaborate on how to apply the framework.

Chapter 5
Implementation Details

This chapter describes the prototype implementation of the mobile Web Services discovery framework introduced in Chapter 4. It gives an overall solution for discovering mobile Web Services in a QoS-aware manner.

The rest of this chapter is organized as follows. Section 5.1 discusses the implementation of the QoS registry. Then the details of the service broker are described in Section 5.2. Section 5.3 introduces the QoS-editor which allows both service clients and service providers to edit their QoS requirements or offers easily. Sections 5.4 and 5.5 describe the implementation of the QoS-metrics collection mechanisms and the reputation system. Section 5.6 explains how the broker incorporates the utility functions. This chapter finishes with concluding remarks in Section 5.7.

5.1 Implementation of the QoS Registry

In order to demonstrate the proposed QoS model, the framework extends the UDDI registry to store Web Services QoS information. As discussed earlier, existing Java Class libraries such as UDDI4J provides APIs to facilitate service publishing and updates, which allows providers to register and update their Web Services QoS attributes easily. Thus UDDI4J is used by the framework for interacting with the UDDI registry.

The framework is implemented in Apache Axis, an open source, Java-and-XML-based Web Services platform for creating and deploying Web Services. Apache's Web Services Invocation Framework is also used for invoking Web Services, as it contains the APIs for providing binding-independent access to any Web Services. JUDDI, an open source Java implementation of the UDDI specification for Web Services, is used for setting up the UDDI registry on the host machine, which is connected to a MySQL database. The Eclipse 3.1 development platform with the Web Tools Platform plug-in is used as the development environment for building the Web Services and client application.

In this prototype, the service providers are hosted by another local machine for testing purposes. The service provider needs to first describe in a WSDL file the location and operations of the mobile Web Services, along with the details of how to use the service. This WSDL file is then published into the UDDI registry. However, because WSDL only considers functional requirements, the tModel in UDDI is used for storing the QoS information. The steps providers take to publish their Web Services QoS information are described below.

After the provider has obtained the authentication token from the UDDI registry, a tModel is created to represent the Web Services QoS information. The tModel provides a classification of a service's functionality and a canonical description of its interface. Each QoS metric and the location of the WSDL description are stored in a keyedReference of the generated tModel. Figure 13 shows an example.

```
<tModel
 tModelKey="uddi:uddi.org:qos:attribute:serverresponsetime">
  <name>uddi-org:qos:attribute:serverresponseTime</name>
  <description>Server response time attribute specification
  </description>
  <overviewDoc>
    <overviewURL …>
      http://localhost:8080/tmodel/qos/attribute/responsetime.xml
    </overviewURL>
  </overviewDoc>
   <categoryBag>
     <keyedReference keyName="QoS attribute specification"
                     keyValue="qosAttributeSpec"
                     tModelKey="uddi:uddi.org:categorization:types"/>
     <keyedReference keyName="Response time metrics"
                     keyValue="responseTimeMetrics"
                     tModelKey="uddi:uddi.org:qos:metrics:responsetime"/>
      <keyedReference keyName="Response time unit"
                      keyValue="millisecond"
                      tModelKey="uddi:uddi.org:qos:unit:millisecond"/>
   </categoryBag>
</tModel>
```

Figure 13: tModel Example

The provider then creates a bindingtemplate, which points to the tModel that contains the QoS information. Figure 14 shows an example.

```
<name>Service Name</name>
<bindingTemplates>
  <bindingTemplate>
   bindingKey="uddi:mycompany.com:Service:primaryBinding"
   serviceKey="uddi:mycompany.com:Service">
   <accessPoint URLType="http">
     http://location/sample
    </accessPoint>
    <tModelInstanceDetails>
       <tModelInstanceInfo>
         tModelKey="uddi:mycompany.com:Service:PrimaryBinding:QoSInformation">
          <description xml:lang="en">
            This is the reference to the tModel that will have the QoS information.
         </description>
       </tModelInstanceInfo>
    </tModelInstanceDetails>
  </bindingTemplate>
</bindingTemplates>
```

Figure 14: BindingTemplates Example

As discussed earlier, UDDI4J already defines the APIs for communicating with the UDDI registry. The APIs are grouped into inquiry and publish APIs. The inquiry APIs simplify the process for the broker to search for relevant Web Services, and the publish APIs help providers update the QoS information.

The detailed steps for publishing the service are shown below. The steps assume the provider has already registered at the UDDI registry and has a user id and a password. This is also the first time the provider publishes its services in the registry, and thus it needs to create and save a business entity.

1. Get an authorization token by passing the user id and password registered at the UDDI registry.
2. Create a business entity to represent the provider.
3. For each service to be published:
    a. Create a tModel to represent the QoS information for the service, and save it in the UDDI;
    b. Create a bindingTemplate containing a reference to the tModel*;*
    c. Create a service entity to represent the service that the provider is publishing;
    d. Set the reference to the bindingTemplate in the service entity;
    e. Add the service entity to the business entity;
4. Save the business entity in the UDDI registry; receive a business key and a list of service keys assigned by the UDDI registry.

When the provider needs to update the QoS information, it retrieves the registered tModel from the UDDI registry, updates its content and saves it with the same tModel key. The detailed steps of the process for updating the services are shown below. The steps assume the provider has already registered at the UDDI registry, has a user id and a password, and the service has been published in the UDDI registry.

1. Get an authorization token by passing the user id and password registered at the UDDI registry.
2. Find the business entity representing the provider with the business key.
3. For each service to be updated:
    a. Find the service entity representing the service that is to be updated with the service key;
    b. Find and update the tModel representing the QoS information for the service;
    c. Save this tModel in the UDDI registry with the same tModel key.

5.2 Implementation of the Service Broker

The service broker is an integral part of the framework. It acts as a mediator among the service providers and mobile devices, and is responsible for the information flow between both components. It is implemented as a Web Service entity that resides on a web server connected to a MySQL database. Since there is only a single service broker Web Service instance, client

applications residing on the mobile devices only need to generate a single stub class to interact with the service broker. The remote service broker object must be instantiated in the constructor of the client application, and it loads a configuration file residing on the client that specifies the URL of the service broker.

The following code sample shows how the service broker is declared in the client application:

```
// UDDI registry key of the tModel the service implements
private string MyUDDITModelKey = “uuid: ... “;
// WS service broker
private WSBroker MyBroker;
// best offer currently available
private ServiceOffer CurrentOffer = null;

// create offer broker
this.MyBroker = new WSBroker ();
```

The following XML file shows the configuration file loaded by the remote service broker object. It contains information about the location of the service broker:

```
<?XML version=“1.0” encoding=“utf-8” ?>
<config>
<BrokerServiceUrl>http://192.168.2.1/WSServiceBroker.asmx</BrokerServiceUrl>
</config>
```

When a service broker starts up, it first processes a configuration file that holds information on the location of the UDDI registry to be used. The following XML file shows the configuration file loaded by the service broker at startup:

```
<?XML version=“1.0” encoding=“utf-8” ?>
<config>
<UDDIRegistryUrl>http://localhost:8080/UDDI/inquiryapi</UDDIRegistryUrl>
</config>
```

The service broker also contains a local cache which is used as a user profile database. The local cache uses client identification as the key for storing the user’s default QoS weights. Prior to returning the service query results to the client, the results are stored in the cache by the broker, so that if network connectivity to the client is lost, the broker can forward the result at a later time. The following code sample shows how the service broker caches the service query result:

```
public boolean returnResult(SOAPobj obj)
{
   boolean status = false;
   if(obj != null)
   {
      synchronized(cache)
      {
         Vector params = new Vector();
         params.addElement(new Parameter(“SOAPobj”,
            SOAPobj.class, obj, null));
         cache.addObject(SOAPobj.getRequestId(), SOAPobj);
```

```
            Boolean bool = (Boolean) soap.invoke("soapObj", params);
            if (bool == null)
            {
               status = false;
            }
            else
            {
               status = bool.booleanValue();
            }
         }
      }
      return (status);
}
```

All communications between components in the framework use SOAP messages for easy extensibility and adoption. The messages types are either related to service inquiry or service reputation updates. A requester can find the related services by sending inquiry messages to the service broker, as the framework adopts a 'pull' approach for service discovery. The service reputation update messages are sent in batch by the reputation system to keep the network overhead reasonable. After the reputation update messages are received by the broker, they are stored in a local database for processing at a later stage.

The following lists the steps the service broker performs when the client sends a service inquiry request.

1. The client first sends the service inquiry to the service broker. If the inquiry does not contain the required QoS weight information, then the default QoS weights are used. Otherwise if the user identification is presented, then the weights are extracted from the user profile database.
   Figure 15 shows a service discovery request example using SOAP, where the required service is related to stock quote with a desired QoS attribute availability at least of 0.9.

2. The broker then checks its local cache to see whether the cache can provide the result. If the cache can provide the result, it sends this result back to the requester and terminates the process. Otherwise, it sends the query to the local UDDI registry to perform functional requirements matching. UDDI4J APIs find_business() and find_Qualifiers() in UDDI4J are used for searching the UDDI registry.

```
<?xml version="1.0" encoding="UTF-8" ?>
<envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
<body>
  <name>Stock quote</name>
   <qualityInformation>
     <availability> 0.9 </availability>
   </qualityInformation>
</body>
</envelope>
```

Figure 15: SOAP Request for Service Discovery

3. The local UDDI Registry processes the query and returns the list of candidate services to the service broker. Then the service broker obtains further information about each candidate service by first retrieving the bindingTemplate and then the WSDL. The following code displays how steps 2 and 3 are performed.

```
// Get a list of all businesses matching the search criteria
BusinessList businessList =
       proxy.find_business(names, null, null, null, null, null,10);

Vector businessInfoVector  =
       businessList.getBusinessInfos().getBusinessInfoVector();
..
..

// After obtaining the business service, retrieve the binding template
Vector bindingTemplateVector =
        businessService.getBindingTemplates().getBindingTemplateVector();
..
..
//Obtain WSDL from the binding template
Vector tmodelInstanceInfoVector =
bindingTemplate.getTModelInstanceDetails().getTModelInstanceInfoVector();

for(int i=0; i<tmodelInstanceInfoVector.size(); i++) {
    TModelInstanceInfo instanceInfo =
            (TModelInstanceInfo)tmodelInstanceInfoVector.elementAt(i);
    ..
    ..
    wsdlImplURI = wsdlImpl.getOverviewURLString();
}
..
..
// get the definition object got the WSDL implementation
try {
    ..
    ..
    implDef = reader.readWSDL(implURI);
 } catch() {
    ..
 }
```

4. The service broker filters the candidate services based on the client's QoS requirements and the device requirement metrics. Afterwards it applies the utility-function-based QoS ranking method, and sorts the resulting Web Services according to the utility score. Our current implementation of the broker uses a static known set of QoS attributes. In another words, the set of QoS parameters is fixed and does not vary by service. While it is possible to extend the set of attributes for specific services, for example, a mapping service may have a "scale" parameter, the current set is reasonable as it covers the major QoS attributes for mobile Web Services. The following code shows the high-level algorithm.

```
Public vector filterServices
(int fMatches, Vector implDefList, QoSAttributes qosRequirements,
```

```
 devProfile deviceProfile, int maxNumServices)
{
  for (s1 = 0; s1 <= fMatches; s1++) {
    QoSAttributes advertised = implDefList[s1];
     if (qosMatchAdvert (advertised, qosRequirements) &&
         meetsHardwareRequirement(deviceProfile, advertised))
       qmatches.add(s1);
  }

  // qosSort() sorts the matching services with utility score
  Vector matches = qosSort(qmatches, qosRequirements);
  for (m1 = 0; m1 <= maxNumServices; m1++)
    selection.add(m1)
  return selection
}

Public boolean qosMatchAdvert
(QoSAttributes advertised, QoSAttributes requirement)
{
  // Compare each QoS attribute
  if (requirement.availability > advertised.availability) {
    isMatch = false
    break;
  }
  if (requirement.servicePrice > advertised.servicePrice) {
    isMatch = false
    break;
  }
  ..
  return true;
}

Public boolean meetsHardwareRequirement
(devProfile deviceProfile, devProfile advertised) {
  if (advertised.CPU > deviceProfile.CPU)
    return false;
  if (advertised.RAM > deviceProfile.RAM)
    return false;
  if (advertised.Resolution > deviceProfile.Resolution)
    return false;
  return true;
}
```

5.3 QoS-Editor

The QoS-editor is a client-side application that allows the clients to easily edit their QoS requirements. As mentioned earlier, the set of QoS attributes is fixed. Using the QoS editor, the client can define:

- The desired QoS values such as processing time, request per second, availability, reliability, reputation, price for the service usage the client is willing to pay, etc.

- The threshold for some QoS values such as video resolution and required network bandwidth.

The QoS-editor is developed using the Java 2 Micro Edition (J2ME) wireless toolkit, along with Java Server Pages (JSP), and a third-party XML parser. The J2ME platform is selected because it is designed for non-conventional consumer devices, which are characterized by some typical features, such as mobility, limited memory and processing power, small display areas, and limitations and variety with respect to input and output methods.

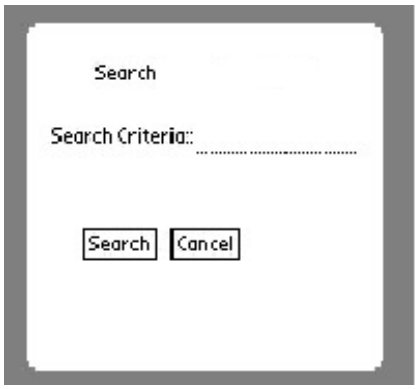
J2ME has two configurations, Connected Limited Device Configuration (CLDC) and Connected Device Configuration (CDC) [17]. CLDC has been designed for severely resource-constrained devices such as today's cell phones and PDAs. CLDC can be used for writing applications for small devices, but it gives very limited functionality. For instance, there is no easy way to provide a graphical user interface. For this reason, J2ME provides the Mobile Information Device Profile (MIDP) profile. MIDP sticks to the CLDC approach of minimizing resource usage but provides additional functionalities to generate attractive user interfaces. The class packages included by MIDP are used extensively for developing the QoS-editor.

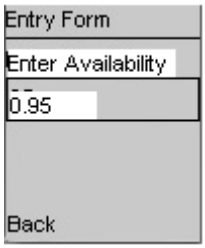
During the development of the QoS-editor, a few design issues were encountered. For instance, since one of the computing tasks that cannot be transferred to the server is the graphical user interface, the QoS-editor has to be designed in such a way that the basic functionality is achieved without putting much load on the resources of the device. Therefore the QoS-editor does not have any images. Moreover, since the memory budget available on the targeted devices is small, the size of the QoS-editor must be restricted to be within this budget so that the application runs comfortably without affecting performance. Figure 16 shows some screenshots of the QoS-editor for defining QoS properties.

The client application corresponds to the Java class *MainClass*, which simply defines the methods required for managing the application life cycle. Event handling consists of mainly checking the item selected by the user.

The application also contains the class *DataParser* which includes the package org.kxml.parser. The package contains an external XML parser for parsing XML received from the server. It makes a connection to the specified URL that refers to an XML file. It contains methods for different functions such as reading the data, adding records to data, etc., and the class uses Vectors and Hashtables to store parsed data.

The class *ServerSearch* is called when the client wants to search for available services. In addition to defining the event handling and displaying the user-interface elements, it also defines the *parseData*() method that holds the callbacks generated when XML parsing is completed.





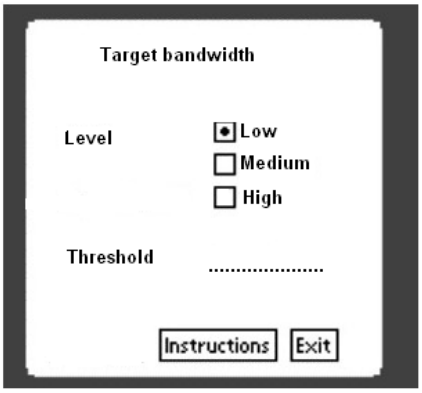


Figure 16: Some Screenshots of the Qos-Editor

5.4 Metric Collection

For QoS metrics that require active monitoring, such as latency, message size and device power consumption, the clients measure these values and send this information to the reputation system.

Once the message is received by the client, its size can be measured easily. In order to measure latency, the client application uses a *latency_timer*, while *ttl_timer* is a predefined counter used to set the timeout period. If the message response is received before ttl_timer expires, then the latency time is equal to the measured time; otherwise it is equal to the value of ttl_timer.

For all latency measurements the total latency to invoke one reference method is divided into t_{SOAP} and t_{call}. The time t_{SOAP} represents the time used for parsing SOAP messages, and t_{call} is the roundtrip time for the message transmission. More power is consumed by the device when it requires more time to transmit and parse the message, so these two time values are used to estimate device's power consumption using (7).

$$P = t_{SOAP} + t_{call} \quad (7)$$

The client then publishes the latency and device's power consumption information to the reputation system via a SOAP message. The implementation details of the reputation system are explained in the next section.

5.5 Reputation System

The reputation system collects feedback from clients and provides reputation scores to the service broker for ranking services that meet a customer's QoS requirements. Once the feedback is received, the system processes the data then updates the reputation score for the related service providers. Similar to the service broker component, a remote object representing the reputation system is created on the client side to ease communication.

The reputation system is divided into two main components, *reputation manager* and *connection manager*. The connection manager receives feedback from clients and updates the database, storing the feedback. The reputation manager computes the reputation score for each service, and decides whether to ask the connection manager to update the service's reputation score.

The connection manager is responsible for receiving feedback from the service clients. It assumes the clients will provide feedback after each interaction with the service. It is also in charge of integrating and maintaining users' feedback information. At the beginning, the client sends a service request to invoke a particular service. The request might fail because the service is off-line, or the request might be rejected due to high system workload or a system fault. In each case, the client evaluates the availability, message size and latency of the service, and sends the feedback to the reputation system. The value for measuring availability is either 0 or 1, where 0 indicates that the service is unavailable or inaccessible and 1 indicates that the service is available or accessible. If the service is accessible, the feedback will also include the message size and the transmission roundtrip time measured by the client. The feedback is sent as a SOAP message to the reputation system.

Every time the connection manager receives feedback from one of its clients, it updates the reputation system's local database. Each feedback entry in the database consists of service key, client key, availability, message size, latency and a timestamp. The service key in the UDDI registry of the service is used as the service key for the reputation system. The IP address of the service consumer is used as the client key. The reason for using the IP address rather than the mobile device ID is because attributes such as availability and server processing time are more related to the network location than the mobile device. The timestamp is used to determine the

aging factor of a particular service feedback. Table 3 gives an example of how the client feedback is stored by the reputation system.

After storing the feedback, the connection manager asks the reputation manager to compute the difference between the measured value and the advertised value for availability, message size and server processing time. The measured value of availability is calculated by dividing the number of successful accesses by the number of attempts. If the reputation score for a service has changed, then the reputation manager informs the connection manager to update the reputation score in the UDDI registry. Updates performed by the connection manager occur in batches with the intent of reducing the number of transmissions. The reputation manager then calculates the average normalized difference for each measurement. The method to compute the average normalized difference is based on Equations (1) and (2) in Chapter 3.

Client feedback should become less relevant as it ages. In order to accommodate this behavior, the reputation system purges client feedback stored in its database if their timestamp value is older than a certain period (the system default is set to 3 days). Moreover it employs a damping function to model the reduction of feedback influence over time based on Equation (3) in Chapter 3. This ensures that a given reputation score will converge to very small positive value as time passes, and reputation values provided recently are more important than those that were provided a long time ago. Furthermore, an aging factor is included to help adjust the influence of feedback.

Service key	Client key	Availability	Server Processing Time (ms)	Message Size (kb)	Timestamp
74154900-f0b0-11d5-bca4-002035229c64	69.36.87.10	1	5.43	13	2007-06-20 09:20:22
9021cb6e-e8c9-4fe3-9ea8-3c99b1fa8bf3	24.23.36.12	1	8.29	21	2007-06-20 09:25:02
9021cb6e-e8c9-4fe3-9ea8-3c99b1fa8bf3	6.16.87.10	1	6.79	21	2007-06-20 10:10:56
b6cb1cf0-3aaf-11d5-80dc-002035229c64	30.15.6.210	0			2007-06-20 10:15:01

Table 3. Example of the Client Feedback Table

Figure 17 shows the sequence diagram of the reputation system. On starting up, the client sends the feedback to the connection manager. The connection manager adds a new entry to the database and exchanges acknowledgements with the client. It also notifies the reputation manager to compute the reputation score based on the new data. After the reputation score is computed, the reputation manager updates the service's reputation attribute stored in the UDDI. The reputation manager will also periodically purge database entries that are older than a certain period.

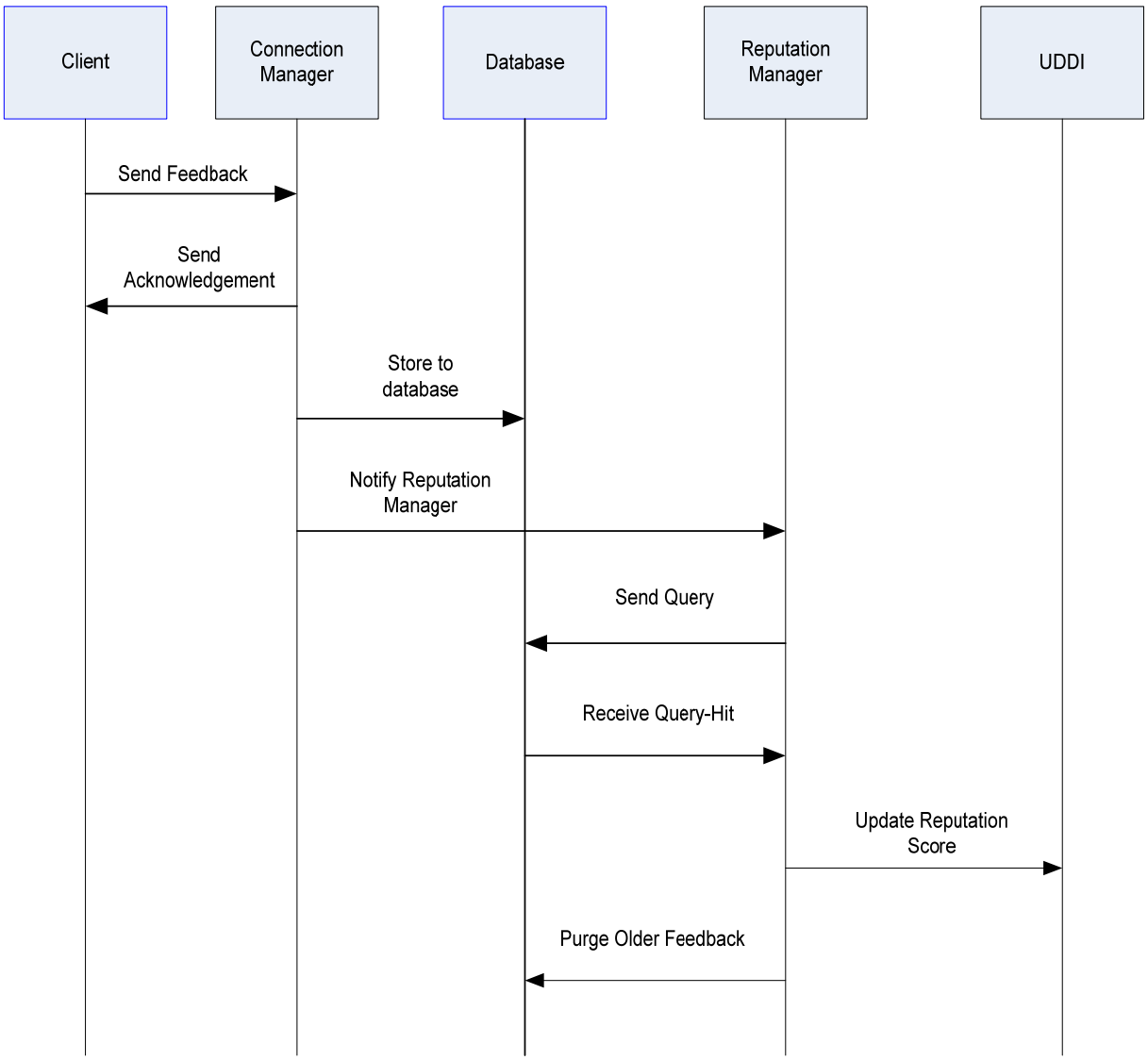


Figure 17: Sequence Diagram of Reputation System

5.6 Implementation of the Utility Functions

After collecting the advertised QoS information for each candidate Web Service, the broker ranks them by applying the utility function evaluation technique. The implementation of this technique is based on a set of requirements defined by the clients and providers, expressed through utility functions, along with the normalized resource costs and the weight assigned to each criterion. Keep in mind that this selection technique does not intend to address how to achieve optimality for both the service provider and service clients. Rather, it focuses on how to select a Web Service that best reaches the equilibrium that is beneficial for both parties.

Utility functions can be difficult to formulate because it is hard to make the requirements explicit, and often clients themselves do not understand their intended use of the Web Services. The general approach adopted by this system for building these utility functions is to choose salient points and then interpolate between them. As recommended by Keeton and Wilkes [29], well-designed graphical tools can guide users in enunciating requirements, as well as providing scenario analyses to confirm whether the utility functions are valid. The following highlights the utility function curves for some important QoS attributes.

The *availability* of the service is the percentage of requests that are satisfactorily fulfilled by the service provider. This is an important quality factor from the user's perspective. For example, for a mobile Web Service that provides stock quotes, a trader will want the service to have high availability, because he/she needs access to the real-time information on a streaming basis. In this framework, the client needs to specify the minimum availability threshold and a second availability above which the client does not care. If the client specifies that availability below 98% is unacceptable, then 0.98 is chosen as one salient point of the availability utility function. If the client does not care whether the availability is above 99%, then a second salient point is yielded at 0.99. The resulting availability utility function curve is produced by interpolating and is shown in Figure 18.

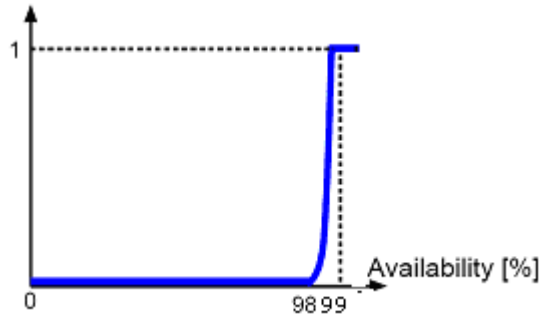


Figure 18: Availability Utility Function Curve

The following explains the utility curve for the *security* attribute. As mentioned in Chapter 3.8, the matching of security encryption algorithm is out of the scope of this thesis, and this attribute only considers the cryptographic key size. If the client considers any security level under 64-bit to be useless, and the ideal security level to be 128-bit, then when the security level provided is less than 64-bit, the utility score is zero because it is considered to be unacceptable if the Web Services do not provide the minimum security level specified by the client. When the security level provided is equal to 64-bit, the utility score increases to a midrange value between 0 and 1. When the security level provided is equal to the client's ideal requirement, in this case 128-bit, the utility score increases to a midrange value between 0.5 and 1. Finally, when the security level provided exceeds the client's ideal requirement, the utility score is equal to 1. Figure 19 shows a step function which represents the security utility curve.

In order to determine the salient points for *latency*, it is important to understand that mobile Web Services response times can affect user experience. In this framework, the client can specify the mobile Web Services response time threshold. When the service fails to respond within this threshold, the client will become frustrated and therefore has a utility score of 0. The client can also specify the response time that gives clients the feeling of instantaneous response. Any response time below this period should have a utility score of 1. The framework interpolates between these two points to obtain the latency utility function curve. For example, if the client specifies 8-11 seconds as the response time threshold, and 0-100 milliseconds as the time that gives him/her the instantaneous response, then the latency utility curve shown in Figure 20 is produced.

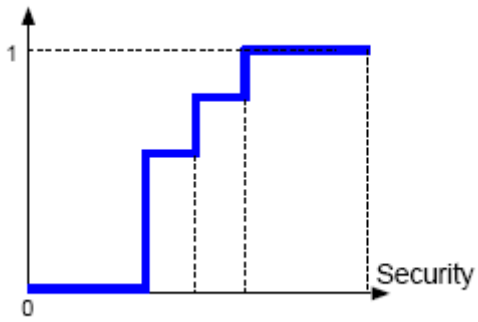


Figure 19: Security Utility Function Curve

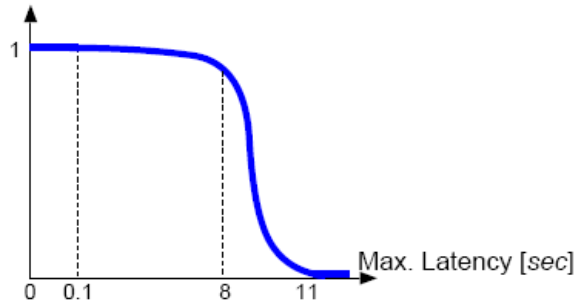


Figure 20: Latency Utility Function Curve

For attributes such as throughput and bandwidth, the framework allows the client to decide the rate that he/she believes is sufficient or useless for the application. Similarly, the provider can decide the bandwidth and throughput that is too costly or very inexpensive to provide. Using these two salient points, the framework applies a curve-fitting technique selected by the user (e.g., linear, polynomial, etc.) to generate the utility function curve. For example, if a client thinks a throughput rate of 1,000 requests/seconds or below is useless and any throughput of 2,000 or more requests/seconds is good enough, then the utility function curve shown in Figure 21 is generated.

Price is the monetary cost the client is willing to pay for the service. The client specifies the maximum price he/she is willing to pay for the service, and the price where the user does not care paying. Using these two salient points and polynomial interpolation, the resulting curve shown in Figure 22 is produced.

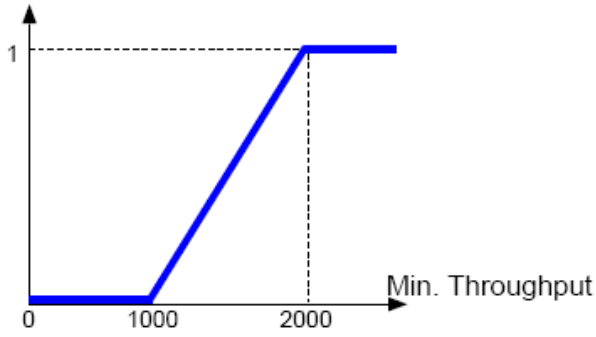


Figure 21: Throughput Utility Function Curve

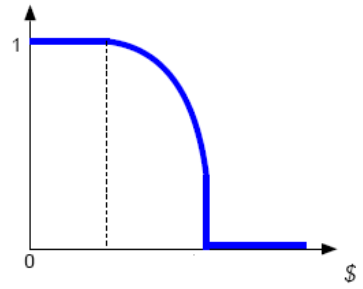


Figure 22: Price Utility Function Curve

After the broker obtains the utility score for each attribute, it computes the score for user utility and provider utility. The QoS attributes such as monetary cost, availability, throughput, reputation, bandwidth, video resolution, security, device power consumption and server processing time, along with the specified weighted factors are included for determining user utility. The QoS attributes message size, CPU/memory consumption on the server and bandwidth are used in determining provider utility. Once the user utility and provider utility for the service have been obtained, the overall service score can be calculated by the aggregate utility equation as mentioned earlier.

Afterwards the broker will rank the Web Services based on the utility scores. If there are several services with maximal score, one of them is selected randomly. If no service satisfies the user requirements, an execution exception will be raised and the broker will propose that the user relax the requirements.

5.7 Concluding Remarks

This chapter has presented the implementation of a QoS-aware mobile Web Services Discovery framework, integrating different components including the service broker with QoS-aware service selection, a reputation system, UDDI, and the QoS-editor. The framework allows providers to register their services along with their QoS properties, and also enables clients to discover services based on both functional and non-functional requirements. Using this framework, the client is able to select a service that best matches its needs among the service instances that satisfy its requirements, and balances the resource consumption cost of the service providers.

The prototype has been deployed and the overhead seems to be reasonable for enhancing service discovery with QoS awareness in the mobile computing environments. In summary, the framework demonstrates an effective overall solution to QoS-aware service discovery in mobile computing environments. The next chapter validates the framework experimentally.

Chapter 6
Evaluation of the Framework

6.1 Evaluation Approach

The main goal of this research is to improve mobile Web Services discovery with a new approach. This chapter presents an analytical study and empirical results of the QoS-aware mobile Web Services discovery framework, using the prototype described in the previous chapter. These analytical and experimental comparisons validate the proposed service-selection mechanism.

The analytical study compares the proposed QoS-aware utility-function-based service-selection mechanism with two other alternatives to help understand the benefits and limitations of adopting this new approach. The evaluation results demonstrate the improvement in discovery results of the new QoS approach compared to the syntactic approach. Several usage scenarios illustrate the functionality of the prototype, and performance overhead measurements are also presented.

The evaluation attempts to answer the following questions:

- What are the benefits for the client and the provider when they adopt this new strategy over the traditional service-selection approach?

- As more Web Service QoS information is included, what is the impact on latency?

6.2 Analytical Study

In order to estimate the complexity and assess the benefits and limitations of the proposed service-selection mechanism, it is compared analytically with two alternatives, static service invocation and dynamic random service selection. In static service invocation, the client is tied to a single provider for each Web Service, and the broker is not involved. In dynamic random service selection, the client sends its functional requirements to a directory agent, which searches from a list of available service providers to find matching services. The agent then randomly selects a provider and returns the provider's information to the client, which sends a service request to the provider based on this information.

Service-discovery latency measures the time between when the client searches for the service and receives the reply. For service discovery without QoS, the time only involves network transmission, as services are selected randomly and reputation is not checked. In contrast, the latency with QoS-aware service discovery is larger because of the time spent in service selection. On the other hand, the proposed QoS-aware service discovery method can increase aggregate

utility and individual utility compared to service discovery without QoS. This section studies the benefits and costs of the proposed QoS-aware service discovery method.

The system environment is modeled as a set of clients that randomly request for service, which can be characterized with the following assumptions:

- Each client is independent of all other clients.
- The total number of clients that may request a service is fixed.
- The provider and the broker are modeled as M/M/1 queues. This assumption is chosen to simplify the analysis.
- The requests passed from the broker are serviced by the provider on a first-come-first-serve basis.
- Dynamic random service-selection picks the provider uniformly. This assumption is chosen to help simplify the modeling equations for dynamic random service-selection.
- The transmission time for a client to send a request to a provider is the same for all providers.
- The transmission time for a provider to return a service result to a client is the same for all clients. The above two assumptions are chosen to help make it easier to derive the modeling equations.

A model of the system is shown in Figure 23. Table 4 explains the notations used in Figure 23 and the modeling equations.

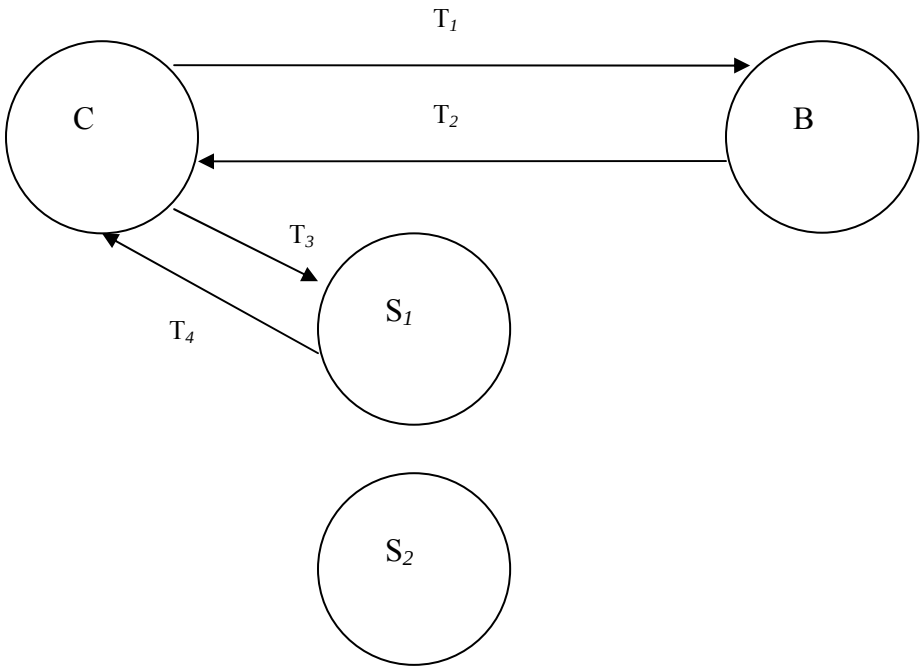


Figure 23: Mobile Web Services Discovery Model

Notation	Meaning
$C_1, C_2, ..., C_n$	Each client
B	Service broker
S_1 $S_1, S_2, ..., S_n$	Each service provider
T_1	Transmission time for the client to send a service request to the broker
T_2	Transmission time for the broker to return the service query result to the client
T_3	Transmission time for the client to request service from the provider
T_4	Transmission time for a provider to return the service result to client
T_b	Average processing time of the broker for each client's request
μ_b	Mean service rate of the broker for each client's request
μ_{S_1}	Mean service rate of provider 1
T_{c_3}	Computation time for the client to prepare the service request to the provider
T_{c_4}	Computation time for the client to process the service response from the provider
$U_{c_{(s1)}}$	Utility score of client c for using the service from provider 1
$U_{S_{1(c)}}$	Utility score of service provider 1 for having client c. The model assumes $U_{S_{1(c)}}$ is the same for all clients.
λ_b	The arrival rate of the admitted requests to the broker
λ_n	The arrival rate of the admitted requests to each service provider S_n
p_n	The probability of request assignment to the service provider S_n, and $\sum_{j=1}^{n} p_j = 1$

Table 4. Notation Used in Analytic Comparisons

6.2.1 Overall Utility

The new QoS-based service selection strategy should help a client find mobile Web Services that better suit their non-functional requirements and also reduce the provider's resource consumption. This analytical study will justify this claim.

In order to compare the utility achieved by the proposed mechanism with that of static service invocation and dynamic random service selection, imagine a scenario where a service has 1 QoS parameter, "bandwidth", and 2 attributes, "high" and "low." As stated earlier, the total number of clients requesting the service is assumed to be fixed; i.e., there are N clients that want to use this service. In this scenario, there are X clients that want high bandwidth, and N - X clients that want low bandwidth. Service provider 1 provides the high bandwidth service, and service provider 2 provides the low bandwidth service. It is also assumed that the utility score of the client when using the desired service is equal to 1, and when using the non-desired service the utility score is assumed to be equal to $(1 - d)$, $0 < d < 1$.

In static service invocation, all the clients are tied to the same service. Therefore, if they are all tied to the service that provides high bandwidth, then the utility score becomes:

$$\begin{aligned}
\text{Utility} &= \sum_{c=1}^{X}(1+Us_1(c)) + \sum_{c=1}^{N-X}(1-d+Us_1(c)) \\
&= X + \sum_{c=1}^{X}Us_1(c) + (N-X)(1-d) + \sum_{c=1}^{N-X}Us_1(c) \\
&= N - d(N-X) + \sum_{c=1}^{N}Us_1(c) \\
&= N(1+Us_1(c)) - d(N-X)
\end{aligned}$$

Therefore decreasing the number of clients that want high bandwidth service, or increasing the difference in utility d, will reduce the overall utility score.

In dynamic random service selection, the probability of getting the desired service is 0.5, and the probability of getting the non-desired is (1-0.5) = 0.5. In this case, the utility score for dynamic random service selection is:

$$\begin{aligned}
\text{Utility} &= \sum_{c=1}^{X/2}(1+Us_1(c)) + \sum_{c=1}^{(N-X)/2}(1-d+Us_1(c)) + \sum_{c=1}^{(N-X)/2}(1+Us_2(c)) + \sum_{c=1}^{X/2}(1-d+Us_2(c)) \\
&= \sum_{c=1}^{X/2}(2+Us_1(c)+Us_2(c)) + \sum_{c=1}^{(N-X)/2}(2+Us_1(c)+Us_2(c)) - \sum_{c=1}^{N/2}d \\
&= N + \sum_{c=1}^{N/2}Us_1(c) + \sum_{c=1}^{N/2}Us_2(c) - \sum_{c=1}^{N/2}d \\
&= N + \frac{N}{2}(Us_1(c)+Us_2(c)-d) \\
&= \frac{N}{2}(Us_1(c)+Us_2(c)+2-d)
\end{aligned}$$

Therefore the overall utility score is proportional to the two providers' utility scores and inversely proportional to the difference in utility *d*.

Meanwhile, the utility score using the proposed mechanism is:

$$\begin{aligned}\text{Utility} &= \sum_{c=1}^{X}(1+Us_1(c)) + \sum_{c=1}^{(N-X)}(1+Us_2(c)) \\ &= N + \sum_{c=1}^{X} Us_1(c) + \sum_{c=1}^{(N-X)} Us_2(c) \\ &= N + XUs_1(c) + (N-X)Us_2(c)\end{aligned}$$

The gain in utility using the proposed mechanism over static service invocation is:

$$\text{Gain in Utility} = (N-X)(Us_2(c) - Us_1(c) + d)$$

The term $(Us_2(c) - Us_1(c) + d)$ can become negative if the difference between the utility of service provider 1 and service provider 2 for having the client is greater than the difference in client utility for having the preferred service over the non-preferred service. In another words, if the difference in client utility between the two services is insignificant, and service provider 2 has a much lower utility for having a client than service provider 1, then it is better to tie all service invocations to service provider 1.

The gain in utility using the proposed mechanism over dynamic selection is:

$$\begin{aligned}\text{Gain in Utility} &= (X - \frac{N}{2})(Us_1(c) - Us_2(c)) + \frac{N}{2}d \\ &= \frac{N}{2}(Us_2(c) - Us_1(c) + d) - X(Us_2(c) - Us_1(c)) \\ &= (\frac{N}{2} - X)(Us_2(c) - Us_1(c)) + \frac{N}{2}d\end{aligned}$$

The above shows the gain in utility consists of the provider's utility gain/loss and the client's utility gain. If $Us_2(c)$ is less than $Us_1(c)$, and less than half of the clients want high bandwidth service, then the provider's utility is a loss. Similarly, if more than half of the clients want high bandwidth service, but $Us_2(c)$ is greater than $Us_1(c)$, then the provider's utility is also a loss.

Consider the case when the number of matching service instances is equal to *M*. In dynamic random service selection, the probability of getting the desired service is $\frac{1}{m}$, and the probability of landing a non-ideal service is $\frac{m-1}{m}$. Assuming for all clients, the client's utility score for having his/her ideal service is equal to 1, the client's utility score for having the least-preferred

service is equal to 0, and the client's utility score for all other services follows uniform random distribution. Therefore on average, the client's utility score will be equal to 0.5. With *N* clients and 1 QoS parameter with *M* different attributes, the overall utility score in dynamic random service selection is:

$$\text{Utility} = \left(\frac{N}{M}\right)\sum_{i=1}^{M} Us_i(c) + 0.5N$$

On the other hand, for the proposed mechanism, assuming *N/M* clients want to use the service from provider *p*, for $1 \le p \le M$, then the utility score is:

$$\text{Utility} = \sum_{c=1}^{N/M}(1 + Us_1(c)) + \sum_{c=1}^{N/M}(1 + Us_2(c)) + \ldots + \sum_{c=1}^{N/M}(1 + Us_m(c))$$
$$= \left(\frac{N}{M}\right)\sum_{i=1}^{M} Us_i(c) + N$$

The gain in utility using the proposed mechanism is 0.5N. This shows that in the case of M matching services under the previous assumptions, the gain in utility with the proposed mechanism is proportional to the number of clients N. Note that if the distribution of the preferred provider is not identical, as is more likely, the gain/loss will depend on the relative differences in provider's utility.

6.2.2 Overall Cost

The intent of this section is to introduce some intuition on the cost, and not a full detailed queuing theory analysis on the cost, which is beyond of scope of this thesis. Also, the network transmission latency is ignored in the overall cost.

In the static service invocation model, all clients' requests are directed to a service provider. The arrival rate of admitted request to a service provider is equal to λ_s, and the mean service rate is equal to μ_s. The total time spent in the service provider is:

$$\text{Time spent in service provider} = \frac{\frac{1}{\mu_s}}{1 - \frac{\lambda_s}{\mu_s}}$$
$$= \frac{1}{\mu_s - \lambda_s}$$

In dynamic random service-selection, the client's requests are distributed evenly to all the service providers. The arrival rate of admitted requests to each service provider *n* is equal to λ_n, and the total cost is:

$$\text{Total time in dynamic service-selection} = \frac{1}{\text{n}} \sum_{i=1}^{n} \frac{\frac{1}{\mu_{s_i}}}{1 - \frac{\lambda_s}{n\mu_{s_i}}}$$

$$= \frac{1}{\text{n}} \sum_{i=1}^{n} \frac{1}{\mu_{s_i} - \frac{\lambda_s}{n}}$$

Note that the arrival rate of admitted requests to each provider in dynamic random service-selection is lower than static service invocation because the incoming requests are distributed among the providers.

The total time needed for performing service discovery in the proposed QoS-aware utility-function-based service selection is divided into two portions, the total time spent between the clients and the broker, and the total time spent between the clients and the providers. In order to keep things simple, only the case with two providers is considered. Assuming the fraction of clients selecting provider 1 is *K*, the fraction of clients selecting provider 2 is (1 - *K*). Also assuming μ_{s_2} is equal to some ratio *R* of μ_{s_1}, then the total cost of the proposed mechanism is:

$$\text{Total time in proposed mechanism} = \frac{1}{\mu_b - \lambda_b} + \frac{K}{\mu_{s_1} - K\lambda_1} + \frac{(1-K)}{\text{R}\mu_{s_1} - (1-K)\lambda_1}$$

Compared to the other two methods, the proposed QoS-aware service selection mechanism adds an overhead of $\frac{1}{\mu_b - \lambda_b}$ as the clients need to first contact a broker before sending a service request to a provider. Also, as noted before, the value of λ_1 will depend on how many requests get sent to the provider, which will depend on the service selection strategy. Moreover, for the proposed selection mechanism, a larger value of *K* will increase the value of λ_1, but how the two variables correlate depends upon the implementation of the provider, and determining this correlation factor is beyond the scope of this thesis.

6.3 Experimental Study

As described in the previous chapter, the framework prototype is implemented as follow: the service broker, reputation system, and service provider are implemented with Apache Axis, MySQL database, and the Java 1.5 platform. The service client is implemented with the Java 2

Wireless Toolkit. The service broker and reputation system are running on a Pentium IV CPU 2.67 GHz, 1MB Cache, 1.00 GB RAM, Windows XP. The service provider is running on a Pentium IV 2 GHz, 512 kb Cache, 1.00 GB RAM, Windows XP. A Sony Ericsson z550a cell phone is used for the client mobile application.

6.3.1 Test Scenario 1

Test scenario 1 has five different mobile Web Services for converting the client-requested JPEG images to a different quality. Each service returns a JPEG image of a different quality. Lower quality leads to smaller image size, thus less network traffic and less power consumption, but at a cost of worst perceived quality. The purpose of this experiment is to analyze the utility-function approach for evaluating the overall service utility.

6.3.2 Test Scenario 2

Another test scenario is needed to demonstrate how this service discovery framework is beneficial to the service provider. The purpose of this scenario is to compare the server throughput performance of the service provider using the QoS-utility approach vs. random service-selection. The experiment consists of a service that has 1 QoS parameter "bandwidth" with 2 attributes "high" and "low". There are 2 service providers, with service provider 1 running on a Pentium IV 2 GHz, 512 kb Cache, 1.00 GB RAM, Windows XP, and service provider 2 running on a Pentium IV 1.3 GHz, 256 kb Cache, 512 MB RAM, Windows XP. Different client request rates are simulated for each run, ranging from 5-70 requests/second, in steps of 5 requests/second. Then it measures the average total throughput of the 2 service providers. The same runs are repeated for random service-selection. Figure 24 illustrates the setup for test scenario 2.

6.3.3 Test Scenario 3

In this scenario, the performance of the prototype is evaluated in terms of the overhead of service discovery latency, the time between when the client searches for the service and receives the reply. The latency is decomposed into three parts:

- The time spent in transmitting the SOAP messages in the network.
- The time spent by the reputation system for checking and computing the reputation of service providers.
- The time spent by the service broker for choosing the best among service offers.

The latency and message size are measured for a number of service offers and QoS dimensions.

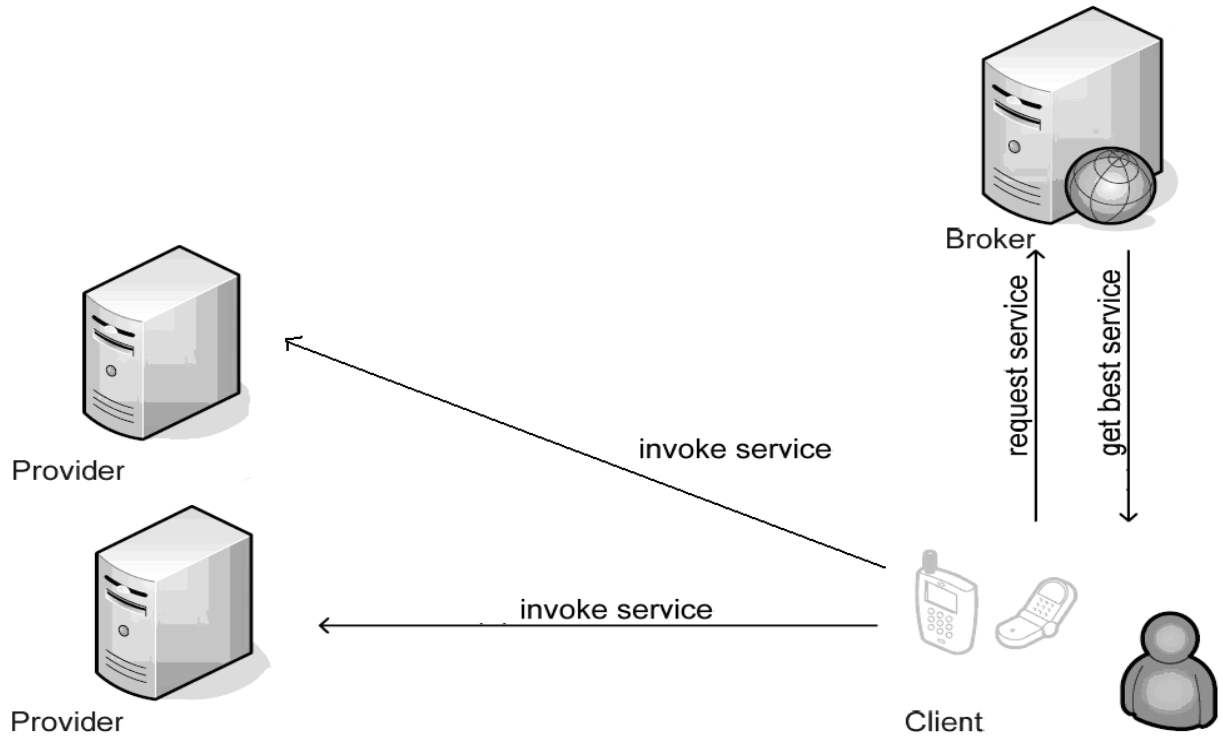


Figure 24. Test Scenario 2 Setup

6.4 Experimental Results

In this section, the experimental results for the different test scenarios are presented. The purpose is to evaluate the prototype and to study whether the actual results deviate from the expected observations.

6.4.1 Test Scenario 1 Results

Test scenario 1 uses five different mobile Web Services for converting the client-requested JPEG images to different quality. Each service returns a JPEG image with different quality. All mobile Web Services are hosted by the same service provider. The client specifies their QoS requirements in the service request. Table 5 shows the typical image size associated with the JPEG quality factor for the mobile Web Services. Regarding the resource consumption cost for the service provider, Table 6 gives the average CPU time and bandwidth cost for each service.

Assume a service request specifies a QoS requirement of availability >= 0.95, server processing time of >= 8 seconds to be unbearable, the ideal processing time to be <= 3, and image size to be greater than > 5000 bytes. The relative importance weightings are 0.3, 0.3 and 0.4 respectively. Other dimensions are not considered to be relevant by the client. Table 7 shows the QoS properties advertised by the provider. The CPU time and server processing time are actual measured values, while availability and bandwidth are arbitrary values.

Service	*1*	*2*	*3*	*4*	*5*
Image Size (bytes)	468	1264	5654	8942	10564

Table 5. Image Size and its JPEG Quality Factor

Service	*1*	*2*	*3*	*4*	*5*
CPU time (seconds)	0.015	0.035	0.064	0.15	0.576
Bandwidth (bytes/second)	56	1649	3455	9756	16497

Table 6. Resource Consumption Properties of the Services

Service	*1*	*2*	*3*	*4*	*5*
Availability	0.95	0.94	0.98	0.99	0.99
Server processing time (seconds)	1.203	1.455	1.896	1.983	9.654

Table 7. Qos Properties Advertised by Providers

Table 8 shows the utility values of each service, and Table 9 shows the overall utility. Assuming all the providers set the relative importance of the QoS attributes CPU time and bandwidth to be 0.5, the relative importance of availability, server processing time and message sizes are 0.3, 0.3 and 0.4 respectively, and the weighting factors for calculating the overall utility are set to 0.5. If the service is selected randomly, then there is a possibility that the service selected does not fulfill the client's non-functional requirement, as is the case with services 1, 2 and 5. On the other hand, if service selection is based on the best QoS from the client's perspective, then service 4 will be selected. However, service 4 has a relatively higher resource consumption cost for the provider; thus it is less beneficial to the provider compared to other services. Using the utility-function selection method, the overall benefit of each service can be evaluated. Since the utility score for the QoS attribute is equal to zero, the method is able to filter out services that do not meet client's requirement effectively. Service 3 is selected because not only does it provide a decent benefit to the user, it also has a comparatively low resource cost for both provider and

clients. This example demonstrates how the utility-function selection approach can select service with the highest overall QoS benefit relative to cost.

Service	*Availability utility*	*Server processing time utility*	*Message size utility*	*CPU time utility*	*Bandwidth utility*
S_1	0.8	0.999	0	1	1
S_2	0	0.976	0	1	1
S_3	0.95	0.956	0.95	0.92	0.91
S_4	0.99	0.92	0.98	0.87	0.8
S_5	0.99	0	0.99	0.82	0.75

Table 8. Utility Values for Each Qos Attribute

Service	*Client utility*	*Provider utility*	*Overall utility*
S_1	Does not meet client's requirement	0.9995	Does not meet client's requirement
S_2	Does not meet client's requirement	0.988	Does not meet client's requirement
S_3	0.9288	0.938	0.9334
S_4	0.905	0.895	0.9
S_5	Does not meet client's requirement	Does not meet provider's requirement	Does not meet client's requirement

Table 9. Overall Utility Value of Each Service

6.4.2 Test Scenario 2

This experiment compares the server throughput of using the proposed mechanism vs. random service-selection. Server throughput is defined as the number of completed responses sent by the server hosting the mobile Web Services within a time interval. It does not include the time needed by the server to publish the QoS information, or the time for the service broker to select the service instance.

As described earlier, the experiment consists of a service that has 1 QoS parameter "bandwidth" with 2 attributes "high" and "low". There are 2 service providers, and the machine hosting the high bandwidth service is more powerful than the machine hosting the low bandwidth service. It simulates clients' service requests at a rate of 5, 10, 15, and up to 70 requests per second. The QoS attribute value specified in the request is randomly selected. Then it measures the average total throughput of the 2 service providers, where the provider is chosen using the proposed QoS-aware strategy. Afterwards, the run is repeated for the random service-selection strategy. The average total throughput at each request rate is calculated based on 5 simulated runs. The mechanism for collecting this value represents a small computation overhead and is used by both runs; thus it should not affect the actual throughput value. Figure 25 shows the measurements from the experiment. The x-axis represents the request rates ranging from 5 to 70 requests per second. The y-axis shows the resulting total throughput of the servers hosting the mobile Web Services. Table 10 shows the range of the throughput values measured for each service selection strategy.

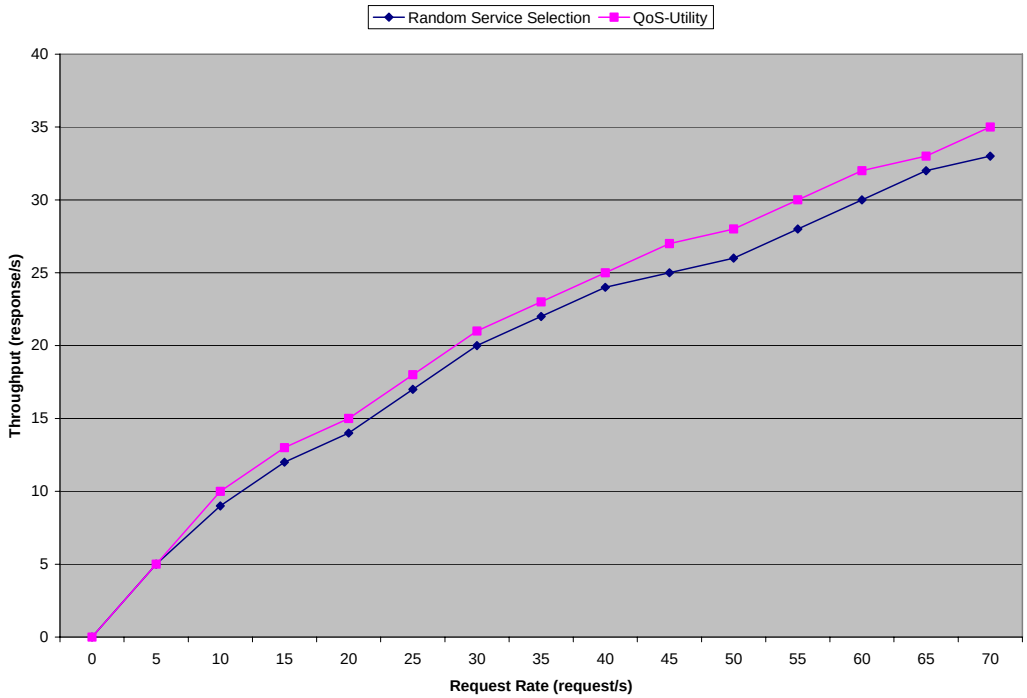


Figure 25: Comparison of Server Throughput

Request Rate	Min Throughput (QoS)	Max Throughput (QoS)	Average Throughput (QoS)	Min Throughput (random)	Max Throughput (random)	Average Throughput (random)
5	5	5	5	5	5	5
10	10	10	10	9	10	9
15	12	15	13	10	15	12
20	14	19	15	14	17	14
25	17	21	18	16	21	17
30	20	25	21	18	25	20
35	22	30	23	22	27	22
40	23	31	25	23	29	24
45	26	32	27	24	30	25
50	27	33	28	25	31	26
55	27	33	30	26	33	28
60	31	37	32	28	34	30
65	32	37	33	32	36	32
75	34	39	35	33	37	33

Table 10. Max/Min Measured Throughput Value

At low request rates, the throughput is equal to the request rate. Then when the request rate increases to 15 requests/s, the throughput starts to level off, and this trend continues as the request rate increases. At high request rates, the throughput becomes approximately half of the request rate. This pattern can be observed for both service selection strategies. If this trend stays consistent, then the server hosting the provider might become overloaded, and in this case it is assumed that the excess requests are buffered in a queue with infinite size. Some possible reasons are that the average throughput is skewed by the lower measured values, or the mean service rate might be reduced at high request rates. Nevertheless the exact reason for this behaviour is not known and should be investigated in the future. The experiment also shows that the QoS-utility-function service selection strategy offers a better server throughput than the random service-selection approach, especially at high request rates. One possible reason is that during the QoS-utility-function trial, the more powerful provider was selected more than half of the time, resulting in a higher overall average throughput.

6.4.3 Test Scenario 3

Table 11 and Table 12 show the performance measurement of the time spent in the service broker and reputation system for varying number of service offers and the number of QoS attributes specified for evaluation. All measurements are in milliseconds. The average, range and standard deviation are calculated from 5 simulation runs. All values are rounded to 2 decimal places; all numbers are in milliseconds.

Avg = Average
R = Range between the highest and lowest measurement
SD = Standard Deviation

Number of Service Instances	*Number of QoS attributes*								
	4			*8*			*12*		
	Avg	*R*	*SD*	*Avg*	*R*	*SD*	*Avg*	*R*	*SD*
4	0.18	0.06	0.03	0.36	0.12	0.07	0.61	0.23	0.12
8	0.22	0.10	0.06	0.44	0.18	0.11	0.98	0.23	0.12
12	0.53	0.29	0.17	0.97	0.31	0.22	1.81	0.42	0.19

Table 11. Time Spent in Service Broker for Service Selection

Number of Service Instances	*Number of QoS attributes*								
	4			*8*			*12*		
	Avg	*R*	*SD*	*Avg*	*R*	*SD*	*Avg*	*R*	*SD*
4	0.08	0.02	0.02	0.10	0.04	0.02	0.11	0.03	0.01
8	0.17	0.04	0.03	0.20	0.05	0.02	0.22	0.06	0.03
12	0.24	0.02	0.01	0.28	0.05	0.03	0.31	0.03	0.01

Table 12. Time Spent in Reputation System

As seen from the measurements, the time spent in the service broker for the utility function computation increases more rapidly with the number of QoS attributes. Also a few of the measurements do not fall within 2 standard deviations of the mean More simulation runs are needed in the future to thoroughly investigate the reason behind this behavior.

Having QoS-aware mobile Web Services discovery can lead to a higher overhead caused by larger message size and additional time for parsing the SOAP message. This overhead is measured by comparing the average latency for a service request with and without QoS description to the same service provider. Table 13 shows the result based on 5 simulation runs.

Number of QoS attributes	*Latency*		
	Avg	*R*	*SD*
No QoS description	60	4	3
4 QoS attribute descriptions	84	7	5
8 QoS attribute descriptions	97	12	9
12 QoS attribute descriptions	117	23	11

Table 13. Average Latency

As seen from the measurements, the amount of increase ranges from 24 ms for 4 QoS attributes to 57 ms for 12 QoS attributes. It can be observed that with more QoS attributes, the latency increases linearly because of growth in message size and more QoS information to parse.

6.5 Summary of the Evaluation

Three different sets of experiments are presented in this chapter to evaluate the prototype quantitatively. The first experiment demonstrates how QoS-utility-based service selection satisfies the client's requirements better than a typical service selection approach. It shows how this new approach helps resource-constrained mobile devices by ensuring the device meets the processing power and display resolution requirements of the delivered Web Services. It also demonstrates how this new approach allows clients and providers to specify their QoS requirements. The experiments have shown how the providers can reduce their resource consumption while still satisfying the clients' needs with this approach.

The second experiment demonstrates the advantages of the QoS-utility-based service-selection approach for Web Services providers. Service providers can achieve better throughput compared with the conventional service-selection approach. This is because the service-selection criteria can be based on several factors and can be defined by service providers themselves. In addition, this new approach allows QoS metrics to be modified dynamically at runtime so that service providers can prevent their servers from overloading in order to service clients efficiently.

The third experiment demonstrates the performance of the service-broker prototype. It first measures the performance of the utility-function service-selection algorithm. It shows that the computation cost of the algorithm grows with the number of QoS attributes specified. Then the latency of this service selection process is measured. Testing with different numbers of candidate services and QoS attributes, it is observed that the performance overhead of parsing and computation performed by the service broker is in the milliseconds range.

Chapter 7
Conclusions and Future Work

7.1 Summary

As mobile computing becomes increasingly widespread, mobile technologies are revolutionizing the way people interact in daily life, work and business. Nevertheless, challenges such as the heterogeneous nature and bandwidth limitation in the wireless environment, along with the limited capabilities and short battery life of mobile devices still need to be overcome.

Web Services QoS schemes are used in wired networks but QoS schemes for wired networks cannot be applied directly to wireless mobile networks. Furthermore, the resource constraints faced by mobile clients and the wireless environment add more new challenges for providing QoS. In order to support QoS for mobile Web Services, information such as bandwidth, latency, device power consumption, etc. should be made available and taken into consideration.

This thesis examines some major problems in mobile Web Services discovery with QoS. Key challenges identified in this area include how to publish and update Web Services QoS information, how to ensure the reputation of service providers, and, more importantly, how to match and rank QoS requirements for both clients and providers. A utility-function based service-selection algorithm is proposed as a solution to address the service-ranking problem, and the validity of this approach is tested with different experiments. The proposed service selection approach evaluates both mobile-client benefits and resource-consumption cost for the provider to derive an overall benefit score of a candidate service. A prototype has been built and the experiments demonstrate the usefulness of the approach in terms of how it provides more benefits to both the clients and providers.

7.2 Thesis Contributions

The following highlights the contributions made by the thesis:

- Publishing and updating mobile Web Services QoS information can be easily performed by the provider. This is achieved by using tModels, a feature in UDDI registries, along with the existing APIs for interacting with UDDI, to store the advertised QoS information of mobile Web Services. When a provider publishes a Web Service, it creates and registers a tModel within a UDDI registry, which represents the QoS information of the Web Service, and is referenced in a binding template that represents the Web Service deployment.

- A mobile application which allows clients to specify the QoS requirements on their mobile devices through an easy-to-use graphical interface.

- In order to evaluate the trustworthiness of the service provider, a reputation-management system is implemented. It collects the provider's past performance as measured by the client to help find service providers that consistently deliver stable QoS performance.

- A novel service-selection strategy which evaluates QoS attributes of each service with utility functions. These utility functions measure the degree of satisfaction with the QoS offered. Service selection chooses the best instance in terms of the overall utility score for both client and provider. The preferences for the client and the provider are taken into account in a comprehensive manner by using weighting factors.

7.3 Future Work

This thesis has demonstrated the advantages of this framework for mobile devices such as cell phones. The ultimate goal is to provide such wireless devices with a flexible and adaptive platform for offering services. Therefore a potential future research project is to study this framework using other kinds of wireless devices such as wireless sensor nodes. Moreover, to fully exploit the potential of the proposed framework, it might be necessary to incorporate semantic modeling of QoS categories. Finally, the utility-function service-selection method can be used to help service providers determine their service price. Future research can explore how to define a pricing strategy for service providers using this service-selection method.

Bibliography

[1] ACPI, "Advanced configuration and power interface specification," Available at HTTP: http://www.acpi.info/

[2] R. Ahmed, R. Boutaba, F. Cuervo, Y. Iraqi, D.T. Li, and J. Ziembicki, "Service discovery protocol, A comparative study," in *Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management Application Session*, Nice, France, 2005, pp. 397-410.

[3] Liberty Alliance, "Liberty Alliance ID-WSF 2.0 Specifications including Errata v1.0 Updates," Available at HTTP: http://www.projectliberty.org/resource_center/specifications/liberty_alliance_id_wsf_2_0_specifications_including_errata_v1_0_updates

[4] Liberty Alliance, "Liberty Alliance Project," Available at HTTP: http://www.projectliberty.org/liberty/strategic_initiatives/telecommunications

[5] Dan Applequist and Stephane Boyera, "Enabling the mobile web," presented at *WWW2005 Conference*, Chiba, Japan, 2005.

[6] Len Bass, Paul Clements, and Rick Kazman, *Software Architecture in Practice*, 2nd ed. Boston, USA: Pearson Education, 2003.

[7] Vishal Batra and Nipun Batra, "Improving web service QoS for wireless pervasive devices," in *Proceedings of the IEEE International Conference on Web Services*, Orlando, Florida, 2005, pp. 130-137.

[8] Tim Berners-Lee, J Hendler, and O Lassila, "The semantic web," *Scientific American*, pp. 16-25, May 2001.

[9] A. Blum, "UDDI as an Extended Web Services Registry: Versioning, quality of service, and more," Available at HTTP: http://www.syscon.com/story/?storyid=45102&DE=1

[10] D. Booth, H. Haas, F. McCabe, E. Newcomer, Champion M., C. Ferris, and D. Orchard, "Web Services Architecture," Available at HTTP: http://www.w3.org/TR/ws-arch/

[11] David Booth and Canyang Kevin Liu, "Web Services Description Language (WSDL) Version 2.0 Part 0: Primer," Available at HTTP: http://www.w3.org/TR/2006/CR-wsdl20-primer-20060327/

[12] Xie-Ren Cao, Hong-Xia Shen, Rodolfo Milito, and Patrica Wirth, "Internet pricing with a game theoretical approach: concepts and examples," *IEEE/ACM Transactions on networking*, vol. 10, pp. 208-216 April 2002.

[13] A. Chatterjee, G. Kuravakal, T. Dias, V. Poddar, and S. Padmanabhuni, "Adding reliability to occasionally connected computing in mobile devices," *SOA Web Service Journal*, vol. 6, pp. 26-31, April 2006.

[14] Cisco, "Understanding codecs: complexity, hardware support, MOS, and negotiation," Available at HTTP: http://www.cisco.com/warp/public/788/voip/codec_complexity.html#mos

[15] A. Dan, D. Davis, R. Kearney, R. King, A. Keller, D. Kuebler, H. Ludwig, M. Polan, M. Spreitzer, and A. Youssef, "Web services on demand: WSLA-driven automated management," *IBM Systems Journal, Special Issue on Utility Computing,* vol. 43, pp. 136-158, March 2004.

[16] Christoph Dorn and Schahram Dustdar, "Sharing hierarchical context for mobile web services," *Distributed and Parallel Databases,* vol. 21, pp. 85-111, Feb 2007.

[17] Dreamtech, *Wireless Programming with J2ME: Cracking the Code*. San Francisco, CA John Wiley & Sons, 2002.

[18] A. Duke, J. Davies, and M. Richardson, "Enabling a scalable architecture with semantic web services," *BT Technology Journal,* vol. 23, pp. 191-201, July 2005.

[19] R. Duncan Luce and Howard Raiffa, *Games and Decisions: Introduction and Critical Survey*. Mineola, NY: Dover Publications, 1989.

[20] S. Fang Rui, "Designing mobile Web services," Available at HTTP: http://www.128.ibm.com/developerworks/wireless/library/wiwebsvc/

[21] Pavel Fedosseev, "Composition of web services and QoS aspects," presented at *Data Communication and Distributed Systems Seminar, University of Technology of Aachen*, Germany, 2004.

[22] Diego Garcia and Maria Beatriz Felgar de Toledo, "A web service architecture providing QoS management," in *Proceedings of the Fourth Latin American Web Congress*, Washington, USA, 2006, pp. 189-198.

[23] G. Gehlen and L. Pham, "Mobile web services for peer-to-peer applications," in *Proceedings of the Consumer Communications and Networking Conference*, 2005, pp. 427-433.

[24] Dieter Gollmann, *Computer Security*, 1 ed.: John Wiley & Sons, 1999.

[25] T. R. Gruber, ""A translation approach to portable ontologies"," *Knowledge Acquisition,* vol. 5, pp. 199-220, 1993.

[26] J Herlocker, L. Konstan, and J. Riedl, "Explaining collaborative filtering recommendations.," in *Proceedings of ACM Conference on Computer Supported Cooperative Work*, 2000, pp. 241-250.

[27] Ilpo Koskinen, Esko Kurvinen, and Petteri Repo, "WAP as Situated Action: Explaining the Failure of Mobile Technology." Master's Thesis, University of Helsinki, 2004.

[28] S. Kalepu, S. Krishnaswamy, and S. Loke, "Verity: A QoS Metric for Selecting Web Services and Providers," in *Proceedings of the 4th International Conference on Web Information Systems Engineering*, 2003, pp. 131- 139.

[29] K. Keeton and J. Wilkes, "Automating data dependability," presented at *SIGOPS European Workshop (2002)*, Saint-Emilion, France, 2002.

[30] Kyriakos Kritikos and Dimitris Plexousakis, "Semantic QoS metric matching," in *Proceedings of the European Conference on Web Services*, 2006, pp. 265-274

[31] Y. Liu, A. Ngu, and L. Yeng, "QoS computation and policing in dynamic web service selection," in *Proceedings of World Web Web 2004*, 2004, pp. 200-208.

[32] Zongwei Luo, Kun Qian, Dongjun Cai, and Jenny S. Li, "QoS driven web services assessment and selection," *International Journal of Services Operations and Informatics,* vol. 1, pp. 78-93, 2006.

[33] E.M. Maximilien and M.P. Singh, "Conceptual model of web services reputation," *ACM SIGMOD Record,* vol. 31, pp. 36-41, 2002.

[34] Microsoft and Vodafone, White Paper, "Mobile web services: convergence of PC and mobile applications and services," Available at HTTP: http://whitepapers.zdnet.co.uk/0,1000000651,260085852p-39000438q,00.htm

[35] Roy Mitchell, "Web Services on Mobile Devices," Available at HTTP: http://itmanagement.earthweb.com/entdev/article.php/3612721

[36] Arun Nagarajan and Anbazhagan Mani, IBM Developer Works, "Understanding quality of service for Web Services," Available at HTTP: http://www-128.ibm.com/developerworks/library/ws-quality.html

[37] Oskar Oala, "Service Oriented Architecture in Mobile Devices: Protocols and Tools." Master's Thesis, Helsinki University of Technology, November 2005.

[38] OASIS, "What is SOAP," Available at HTTP: http://www.xml.org/xml/resources_focus_soap.shtml

[39] Sangyoon Oh, "Web Service Architecture For Mobile Computing." PhD Thesis, Indiana University, 2006.

[40] Enrique Ortiz, Sun Developer Network, "Understanding the Web Services Subset API for Java ME," Available at HTTP: http://developers.sun.com/techtopics/mobility/midp/articles/webservices/

[41] S. Ran, "A model for web services discovery with QoS," *SIGEcom Exchanges,* vol. 4, pp. 1-10, 2004.

[42] E. Sanchez-Nielsen, Martin-Ruiz S., and Rodriquez-Pedrianes J., "An open and dynamical service oriented architecture for supporting mobile services," in *Proceedings of the 6th international conference on Web engineering*, Palo Alto, USA, 2006, pp. 121-128.

[43] M. Satyanarayanan, "Fundamental challenges in mobile computing," in *Proceedings of the Fifteenth ACM Symposium on Principles of Distributed Computing*, Philadelphia, PA, 1996, pp. 1-7.

[44] J Schneider, "Convergence of peer and web services," presented at *O'Reilly's Open Source Convention*, San Francisco, CA, 2002.

[45] A. ShaikhAli, O. F. Rana, R. Al-Ali, and D W. Walker, " UDDIe: an extended registry for web services," in *Proceedings of Applications and the Internet Workshops,* Orlando, Florida, January 2003, pp. 85--89.

[46] Amit Sheth, Jorge Cardoso, John Miller, and Krys Kochut, "QoS for service-oriented middleware," in *Proceedings of the 6th World Multiconference on Systemics, Cybernetics and Informatics*, 2002, pp. 528-534.

[47] Zoran Stojanovic and Ajantha Dahanayake, *Service-Oriented Software System Engineering: Challenges and Practices*. Hershey, PA: Idea Group Publishing, 2005.

[48] Sun Microsystems Incorporation, "Jini technology architectural overview. Technical report," Available at HTTP: http://www.sun.com/software/jini/whitepapers/architecture.pdf.

[49] M. Tian, T. Voigt, T. Naumowicz, H. Ritter, and J. Schiller, "Performance impact of web services on internet servers," in *Proceedings of Parallel and Distributed Computing and Systems*, California, USA, 2003, pp. 392-402.

[50] M. Tian, A. Gramm, H. Ritter, J. Schiller, and T. Voigt, "QoS-aware cross-layer communication for mobile web services with the WS-QoS framework," presented at *Mobile Computing and Media Communication in the Internet*, Berlin, Germany, 2004.

[51] V. Tosic, B. Pagurek, K. Patel, B. Esfandiari, and W. Ma, "Management applications of the Web Service Offerings Language (WSOL)," *Information Systems,* vol. 30, pp. 565-586, 2005.

[52] S.J. Vaughan-Nichols, "OSes battle in the smart-phone market," *IEEE Computer,* pp. 10-12, June 2003.

[53] Le-Hung Vu, Fabio Porto, Manfred Hauswirth, and Karl Aberer, "An extensible and personalized approach to QoS-enabled semantic web service discovery," École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland, Technical Report 2006.

[54] W3C, "Web Services Activity," Available at HTTP: http://www.w3.org/2002/ws/

[55] W3C, "QoS for Web Services: Requirements and Possible Approaches," in *World Wide Consortium (W3C) working group note specification*, 2003.

[56] Xia Wang, Tomas Vitvar, Mick Kerrigan, and Ioan Toma, "A QoS-Aware Selection Model for Semantic Web Services," presented at *ICSOC 2006*, Chicago, USA, 2006.

[57] S. Weerawarana, F. Curbera, F. Leymann, T. Storey, and D.F. Ferguson, *Web Services Platform Architecture*. New York, NY: Prentice Hall, 2005.

[58] Katherine S. Willis, "Gap: mobile applications and wayfinding," presented at *Workshop for User Experience Design for Pervasive Computing, Pervasive 2005*, Munich, Germany, 2005.

[59] R. Wishart, R. Robinson, J. Indulska, and A. Josang, "SuperstringRep: reputation-enhanced service discovery," in *Proceedings of the Twenty-eighth Australian conference on Computer Science*, 2005, pp. 49-57.

[60] Tao Yu and Kwei Jay Lin, "The design of QoS broker algorithms for QoS-capable web services," in *Proceedings of 2004 IEEE International Conference on e-Technology, e-Commerce and e-Service*, 2004, pp. 17-24.

[61] Liang-Jie Zhang and Qun Zhou, IBM developer works, "Aggregate UDDI searches with Business Explorer for Web Services," Available at HTTP: http://www-128.ibm.com/developerworks/webservices/library/ws-be4ws/