Enabling QoS in Web Service Composition in a P2P Environment

by

Mojdeh Ghodousi, B.Eng.

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of Master of Applied Science in Electrical Engineering

Carleton University, Ottawa, Ontario, Canada
Ottawa-Carleton Institute of Electrical and Computer Engineering
Department of Systems and Computer Engineering

January 18, 2005

The undersigned hereby recommends to
The Faculty of Graduate Studies and Research
acceptance of the thesis,

Enabling QoS in Web Service Composition in a P2P Environment

submitted by

Mojdeh Ghodousi

In partial fulfillment of the requirements
for the degree of Master of Electrical Engineering

Professor Tony White, Thesis Supervisor

Professor Thomas Kunz, Thesis Co-Supervisor

Chairman, Department of Systems and Computer Engineering

Carleton University

January 18, 2005

Abstract

As Web Services technology is growing rapidly, the need to create more complex business services is becoming apparent. Web Service composition research gained momentum as businesses valued the possibility of automatic integration and collaboration. BPEL (Business Process Execution Language) is one of the prominent Web Service composition notations. BPEL facilitates the creation and execution of business processes based on Web Services. This thesis extends the BPEL approach by enabling clients to define their required Quality of Service (QoS), binds selected Web Services at run time and creates BPEL documents dynamically. This thesis introduces a new framework in which Web Service providers can announce the QoS and clients can search and find their required QoS. The framework uses a Peer-to-Peer (P2P) environment; P2P implementations create an overlay network that provides enhanced security. Another advantage is that P2P puts similar services into closed communities so that the search and discovery of services is accelerated.

Acknowledgments

I am sincerely grateful to my supervisor Professor Tony White for his constant support and invaluable guidance throughout the completion of this thesis. Without his encouragement and patience this work would not be possible. I would also like to thank and appreciate my co-supervisor and graduate advisor Professor Thomas Kunz for his help and guidance. I would like to thank my friend, Donna for helping me with formatting this document. I would like to thank my husband for his love, support, and understanding, and for being by my side whenever I needed a review, opinion or comment. I would like to thank my parents and other closest family members, for their love and belief in me.

Table of Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivation

Web Services technology allows interoperability between applications and provides flexibility to integrate businesses. In order to support Business-to-Business or enterprise application integration, Web Services are required to collaborate. Web Service composition focuses on business process creation based on orchestration and choreography of Web Services. The manual composition of Web Service is time consuming and requires a significant amount of detailed coding dealing with heterogeneous services. As the number of services increase in a composite service the orchestration among the services becomes more problematic. The necessity and complexity of the task of service composition has resulted a number of companies and standardization bodies working toward a common language for defining automation of business process executions.

Some have taken the semantic Web approaches such as DAML-S and some have taken the syntactic approach such as BPML, XLANG, WSFL and BPEL (Web Service composition and these languages are described in detail in section 2.7).

However, most of Web Service composition definition languages are heavily based on Web Service definitions that are provided in Web Service Definition Language (WSDL) descriptions.

WSDL describes the location of the service and the operation it provides. These languages expect that the Web Services that take part in the composition to be stationary at the time of the design of the process model.

These definition languages do not allow clients to select a Web Service dynamically based on the specific requirement that they have. In other words, clients cannot specify their functional and QoS requirements for dynamic selection of Web Services at run time.

One motivation for this thesis is to provide users with the opportunity to specify and select the Web Services dynamically based on a set of functional and QoS criteria.

In order to be able to select services at execution time, services are required to be searched and found based on specific criteria. The Universal Description Discovery Integration (UDDI) is a complementary technology to search web services located on different sites on the Internet. Existing UDDI technology uses central server (operators) to store pointers to registered web services. However, UDDI will inherit the drawbacks associated with using a central system such as availability, scalability and performance. The other drawback that relates specifically to the UDDI functionality is that UDDI is not aware if the server providing the services of a registered Web Service is available and online when a client is trying to discover a service.

This thesis approaches this issue by using a P2P base discovery and search of the Web Services rather than using a central scheduler. Instead Web Services would advertise their existence when they join a P2P network and are therefore available to provide services.

The other motivation for this thesis is to overcome the security issues that are imposed by using Web Services in a business process execution. Security is an important aspect of business collaborations. Since Web Services message exchanges occur over the Internet using HTTP based protocols, similar security threats can potentially exist. Using a P2P environment creates a virtual overlay network and creates self-organized communities. Several security technologies therefore can be enforced on these communities.

1.2 Problem Statement

This thesis focuses on the following problems:

1.2.1 QoS in Web Service Composition

Clients of Web Service compositions are bound to use the static Web Services that are defined at the time of design in a chosen Web Service Composition Language. They cannot select a specific Web Service dynamically at run time based on their requirement specifications.

1.2.2 Web Service Composition Security

Web Service composition enables businesses to interact and interoperate. Any transaction or message exchanged between businesses could potentially contain confidential information. Service providers are required to take the privacy and integrity of messages seriously.

1.2.3 Web Service Search and Discovery

UDDI is used as a standard technology for dynamic search and discovery of Web Services. However, UDDI uses a central system and is not aware whether a registered Web Service is available at the time of client search.

1.3 Thesis Contributions

We developed an end-to-end solution for executing a Web Service composition. The solution is a framework that integrates best of the breed technologies with newly designed components to achieve the goals of this thesis and to address the problems discussed above in a way that allows service providers to have maximum flexibility with minimum effort.

- Integration of existing and new components to provide an end-to-end solution for QoS enabled web service composition within a framework. The framework is part of a P2P environment where an overlay virtual network is created for enforcing security and second, a dynamic search and discovery for Web Services can be used. Also this framework selects and binds services based on defined functional and QoS criteria of the client at run time.

- Developing an XML based schema for clients to define their functional and QoS criteria and for Web Service providers to present their functional and QoS criteria.

- Designing an enhanced search engine as a service entity in JXTA network to search services advertised and dynamically select and compose the services based on the client's defined criteria. The search engine would be designed to support functionality that the composite search XML document requires.

- Developing a prototype incorporating the above designs and concepts.

Here is a brief introduction to the existing technologies used and the motivation behind them:

Web Service Composition Language: This thesis chooses Business Process Executable Language (BPEL) as the basis of the Web Service composition definition language. BPEL is a dominant Web Service composition language that was originally released by IBM and backed by various vendors, and has been proposed to OASIS as a technical standard in 2003. More description is provided in section 2.7.4.

BPWS4J: The IBM Business Process Execution Language for Web Services Java Run Time (BPWS4J) is a platform that can execute business processes written using BPEL. More details are provided in section 2.7.5.

JXTA: To achieve a P2P environment, JXTA technology is used. JXTA, developed by Sun Microsystems, is an open, generalized P2P platform that supports core functions of a P2P system. JXTA 2, the latest release of JXTA, offers a P2P overlay network that is very scalable. Also JXTA provides a dynamic discovery service where Web Services can be found when available, in the JXTA network, thus overcoming the problems with using UDDI. More details are provided in section 2.5.

1.4 Thesis Organization

The rest of the thesis is organized as follows: Chapter 2 provides an overview of the related technologies to this thesis. This chapter starts by describing Client/Server and P2P technologies, JXTA, and continues with an overview of Web Services composition and BPEL description. Chapter 3 discusses state of the art in QoS enabled Web Services composition technologies and an evaluation of their use. Chapter 4 describes the thesis requirements and the architecture proposed to support a QoS enabled Web Service composition. This chapter provides an architectural view of the whole system. Chapter 5 provides details of implementation of the Enhanced search service and the Web Service Composite Search XML schema. Chapter 6 walks through a prototype developed to support the proposal. Chapter 7 is the complete schema definition for Web Service Composite Search. Chapter 8 provides an analysis of the architecture, the degree of compliance with the requirements, the limitations and potential enhancements and additional research ideas.

2 Background

This chapter provides a review of the technologies used in this thesis and includes descriptions of distributed computing, P2P and JXTA technologies. An overview of Web Services and Web Service composition definitions is also provided. Sections 2.5 and 2.7.4 provide more details on JXTA and BPEL respectively.

2.1 Distributed Computing Systems

A distributed computing system is a collection of computing nodes that can have different types of hardware architectures and are interconnected by a communication network.

There are various approaches to distributed computing systems, two of the most noteworthy ones are:

- Client/Server Architecture
- Peer to Peer Architecture

2.2 Client/Server Architecture

In a network where a group of nodes is communicating with each other, a more powerful node is assigned as the server and provides services and information to the other nodes considered as clients. Clients send requests to the server and receive results. Figure 1 illustrates a Client/Server architecture view [1].

This architecture has evolved from a 2-tiered architecture to a 3-tier and multi-tier client/servers architecture, where the software is modularized into two or more pieces and usually each module reside on separate hardware. Client/Server architecture provides more scalability and flexibility in software systems.

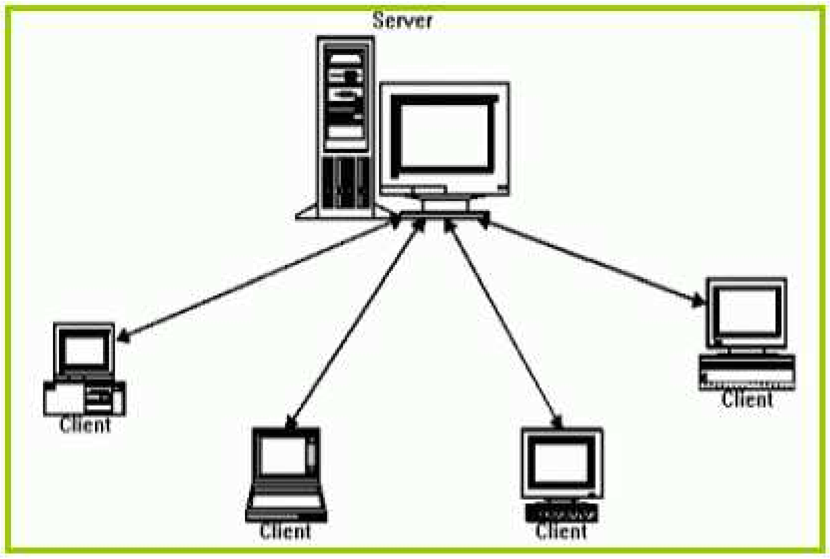


Figure 1: Client/Server Architecture

2.2.1 Pros and Cons of Client/Server architecture

Pros:

- Efficiency: Since data is only transmitted when necessary.
- Security: More level of security can be achieved since there is central point of communication.
- Flexibility: Different hardware can be used for clients; upgrades can be done more easily.

Cons:

- Servers are not aware of the Client’s capabilities and resources.
- Servers usually need to keep large amount of data
- The system depends on the server reliability.
- Network bandwidth between clients and the server affects the performance of the system.

2.3 P2P Architecture

A network architecture may be called a Peer-to-Peer (P-to-P or P2P) network, if the participants share a part of their own hardware resources e.g. processing power, storage capacity, network link capacity and printer. These shared resources are necessary to provide the service and content offered by the network (e.g. file sharing or shared workspaces for collaboration). The participants of such a network are thus resource providers (Service and content) as well as resource requestors [2].

P2P is a newer architecture for distributed computing which overcomes some of the problems associated with the client/server architecture. Figure 2 illustrates P2P architecture in section [1]. The following subsections provide a brief description of various P2P forms: hybrid, pure, virtual.

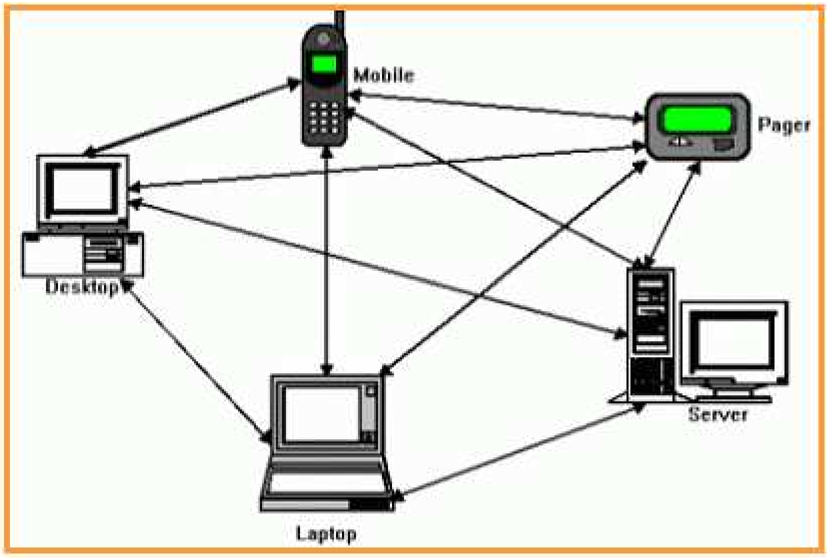


Figure 2: P2P Architecture

2.3.1 Pure P2P

In a pure P2P architecture communication nodes can be both the server and the client and have all the same capabilities. There are certain routing mechanisms that are used for transmission of data among nodes.

2.3.2 Hybrid P2P

In a hybrid environment a central entity still exists but it is different from the client/server architecture in the sense that nodes are sharing resources such as storage, computer power among themselves. Napster (section 2.4.1) is an example of a hybrid P2P.

2.3.3 Virtual P2P Architecture

Currently most of P2P systems are implemented based on a virtual network that connects all the nodes in a non-centralized way with their own routing mechanisms.

2.4 P2P Applications

In the Internet world a set of servers provide the information to the clients through the clients' web browsers. The distinction in P2P computing is the access to decentralized resources available in a network. Each communication node can have the capabilities of both a client and a server. This enhances the Internet network capabilities immensely. The possibility of file sharing among users without interference of servers brings a dramatic scalability to the Internet world. Searching the Internet also takes on a new dimension in a P2P environment. P2P environments/applications provide lower cost and higher availability.

Some of the challenges facing P2P architecture are in addressing schemes, routing and discovery mechanisms. Various applications have been implemented based on P2P architecture, such as Napster, Gnutella, Kazaa and Kazaa lite. Different applications might focus on different characteristics of P2P architecture; in the two following subsections we describe Napster and Gnutella examples in more details.

2.4.1 Napster

Napster shares primarily music files and is an example of hybrid P2P architecture with a centralized file location directory. Users would run desktop software that enables them to share files. This software actually provides a virtual P2P network. Napster is an example of a hybrid P2P architecture. There is a central database, which keeps an index of the files available in the network; when a requester queries for a specific file; the directory server provides the IP address of the user that has that file. The requester then downloads the file from that IP address directly. Although Napster once was a very popular P2P application, it was forced to shut down due to copyright violations. Figure 3 illustrates [3] the Napster architecture overview. In Napster architecture users first connect to a central directory server to find the location of a specific file, then they will directly connect to that location and download the file from that user.

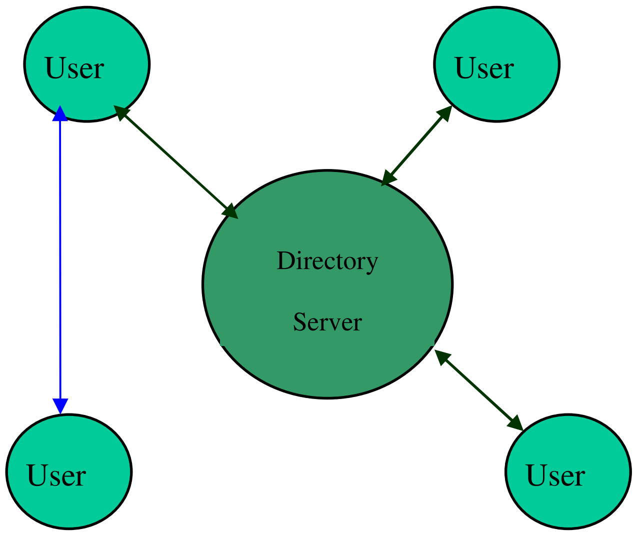


Figure 3: Napster Architecture

2.4.2 Gnutella

Gnutella is also a popular P2P system for sharing files. Unlike Napster, there is no central database involved in this architecture, but users still need to run Gnutella software in order to become part of the P2P network. Figure 4 illustrates the Gnutella architecture [4], where peers exchange files directly without a central directory service.

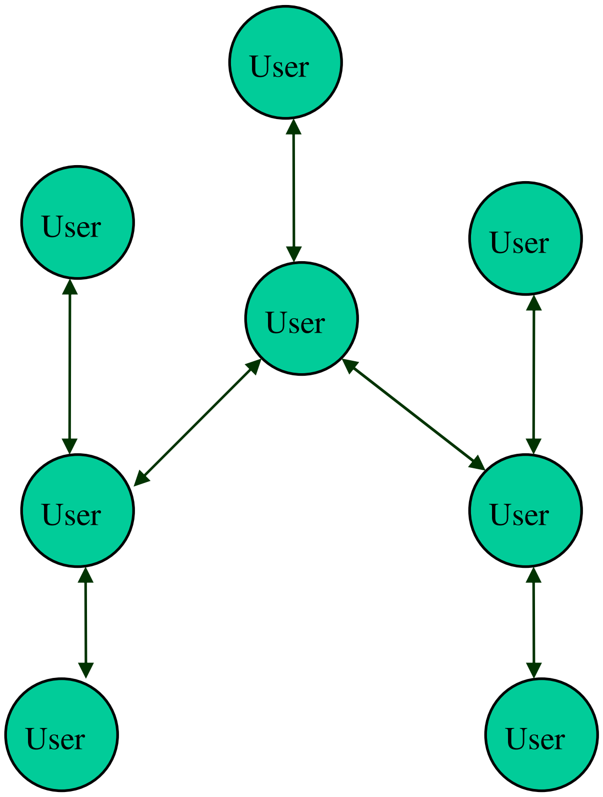


Figure 4: Gnutella Architecture

2.5 JXTA

As part of the framework proposal for this thesis, JXTA has been selected as the technology for providing a P2P platform. The JXTA project is one of the leading, open-source P2P platforms available. Sun's Project JXTA is a set of open, generalized peer-to-peer protocols that allow any devices on a network to connect, communicate, and collaborate with any other device on a network independent of their hardware type. The goal of project JXTA is to provide a platform that supports functionality essential for P2P computing. The term JXTA is taken from the word Juxtapose meaning side by side.

2.5.1 JXTA Benefits

- JXTA peers create a virtual network where any peer can interact with other peers and resources directly, even when some of the peers and resources are behind firewalls or are on different network transports **[5]**.
- JXTA creates an overlay network that limits the propagation of queries to a subset of peers rather than all peers; thus increasing scalability and performance.
- JXTA offers the peer group concept, which defines a scope that creates a virtual boundary of peers where an authentication policy can be enforced upon them for security.
- JXTA is platform independent and all the protocols are based on XML messaging.
- JXTA includes a set of core services such as peer pipes for communications between peers and peer discovery service.
- JXTA provides advertisements, which are XML documents that describe peers, peer groups, pipes and services in the network.

This thesis will benefit from all of the above JXTA features and specifically from the overlay network creation and peer groups, where closed communities will be created for a set of Web Services for enhanced security, scalability and performance. The services will be part of a closed community where security technologies such as authentication can be enforced. The JXTA advertisements, a mechanism for Web Services to announce their services and

QoS information in the form of XML documents to the overlay network, will also be used extensively as part of this proposal.

2.5.2 *JXTA Architecture*

Figure 5 illustrates a logical view of JXTA project:

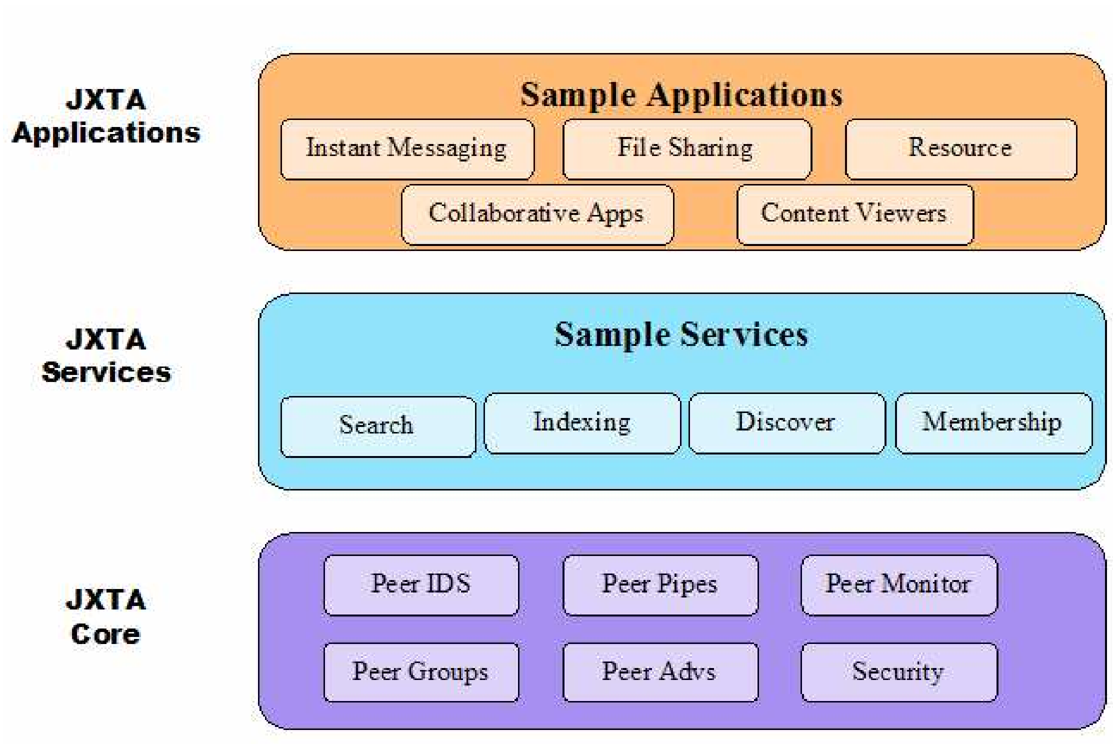


Figure 5: JXTA Architecture View

JXTA is divided in three layers:

Platform or Core layer: The platform layer provides the basic and essential functionalities of a P2P environment, such as peers, peer groups, peer IDs, discovery and basic security.

Services layer: JXTA services are on top of core layer and provide additional functionality that can be beneficial in P2P environment such as searching, indexing, authentication and PKI (*Public Key Infrastructure*) membership.

Application layer: The applications are written by using features in core and services layers.

2.5.3 JXTA Concepts

2.5.3.1 Peers

Peers can be any networked device such as sensors, phones, PDAs, PCs, servers or super computers as long as they implement one or more of the JXTA protocols. A peer ID uniquely defines each peer. Peers publish their network interfaces through advertisements. Peers automatically discover each other and become part of a self-organized peer group. Peers do not require point-to-point connections between themselves; intermediary peers can be used to route messages for those peers separated due to physical network connections, firewalls, NATs (Network Address Translation) or proxies. There are three types of peers: edge, rendezvous and relay.

2.5.3.2 Peer Groups

Peer groups implement a set of peer group services. Peers, self organize into peer groups, where each has a peer group ID. A peer can belong to more that one peer group; by default peers are part of the *Net Peer Group*. Figure 6 illustrates a peer group view of a JXTA network [6]. In this thesis, similar sets of Web Services will be part of the closed community of a peer group, with the following benefits:

- Securing Environment: a security policy can be imposed on peer groups. This policy can range from a username/password authentication to a complex public key cryptography. Peers can also publish encrypted content within the peer group.
- Scoping Environment: peer groups can form a specialized domain or limit the search within the scope of a peer group.
- Monitoring Environment: peer groups allow monitoring of a set of peers for a specific purpose.

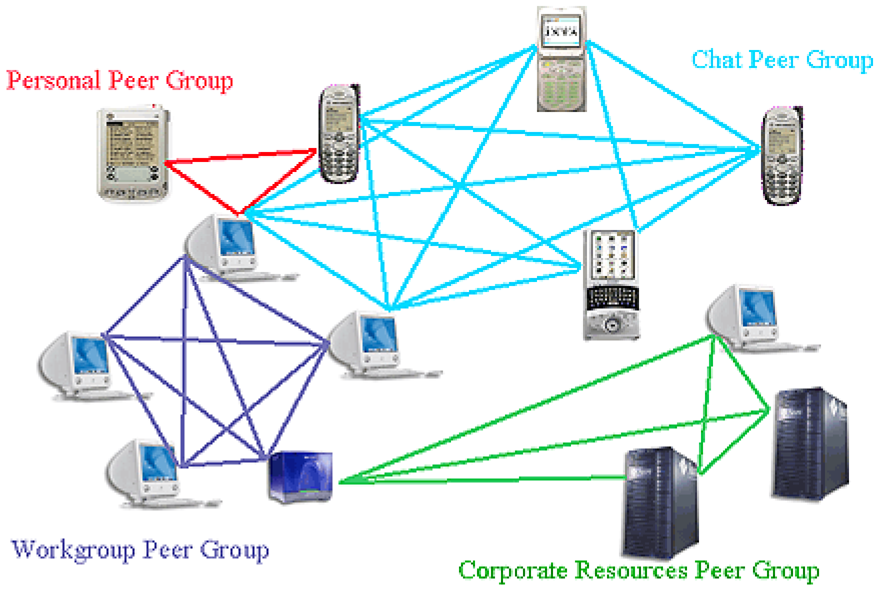


Figure 6: JXTA Peer Groups

2.5.3.3 Pipes

Pipes are the basic transport entity between peers, and they are virtual communication channels that send and receive messages. JXTA requires at least one asynchronous uni-directional pipe. The pipe end-points are referred to as input and output pipes.

2.5.3.4 IDs

As JXTA resources need to be uniquely identifiable, a JXTA ID is used. There are six types of JXTA resources: peer, peer group, pipes, contents, module classes and module specifications. An example of a JXTA ID is:

urn:jxta:uuid-59616261616162614A78746150326033F3BC76FF13C2413CBC0AB663676DA53902

The format of a JXTA ID is URN (Unified Resource Names) [7]. There is a facility in JXTA that create IDs randomly. There are two reserved IDs, NULL ID and Net Peer Group ID.

2.5.3.5 Advertisements

JXTA advertisements are the means of describing resources within the JXTA network and are the core of the JXTA infrastructure since it is concerned with assigning advertisements to various resources. The data in JXTA advertisements are presented in XML language, which is a powerful language-independent data and meta-data presenter. XML has the advantage of being strongly typed based on a defined XML schema. Use of XML schemas prevents syntax errors in the advertisement documents. Advertisements provide crucial benefits in this thesis as part of the framework. Components find each other by using advertisements. They are the means for publishing the services and the QoS information of Web Services. The search component of the framework uses JXTA discovery mechanism, which discovers published Web Services. Table 1 presents the common schema used in JXTA advertisements and the following subsections define different types of advertisements and their XML schemas.

```
<xs:simpleType name="JXTAID">
 <xs:restriction base="xs:anyURI">
  <xs:pattern value="([uU][rR][nN]:[jJ][xX][tT][aA]:)+\-+"/>
 </xs:restriction>
</xs:simpleType>

<xs:complexType name="serviceParam">
  <xs:sequence>
    <xs:element name="MCID" type="jxta:JXTAID"/>
    <xs:element name="Parm" type="xs:anyType"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="Cred">
  <xs:all>
  </xs:all>
</xs:complexType>
```

Table 1: Common Schema used in JXTA Advertisement

Table 1 defines JXTA ID as being a URI type with the pattern restriction provided. It also defines that *ServiceParam* contains two elements, MCID, which is module class ID and *Parm* element. This element defines a class ID and its parameter.

2.5.3.5.1 Peer Advertisement

Peer advertisement describes the peer and its resources to the peer group. Table 2 describes the schema of a peer advertisement. As shown in the schema, the peer advertisement contains the peer ID (PID), the peer Group ID (GID), an optional element representing the name of the peer (Name), an optional element presenting a description (Desc), an optional element that can go up to any number presenting the parameters of a service.

```
<xs:element name="PA" type="jxta:PA"/>
 <xs:complexType name="PA">
  <xs:sequence>
   <xs:element name="PID" type="JXTAID"/>
   <xs:element name="GID" type="JXTAID"/>
   <xs:element name="Name" type="xs:string" minOccurs="0"/>
   <xs:element name="Desc" type="xs:anyType" minOccurs="0"/>
   <xs:element name="Svc" type="jxta:serviceParams" minOccurs="0"
maxOccurs="unbounded"/>
  <xs:sequence>
</xs:complexType>
```

Table 2: Peer Advertisement Schema

2.5.3.5.2 Peer Group Advertisement

Peer group advertisement describes the GID, group module specification (MSID), optional name (Name), optional description (Desc) and the optional list of service parameters (Svc). Table 3 describes the schema of a peer group advertisement.

```
<xs:element name="PGA" type="jxta:PGA"/>

<xs:complexType name="PGA">
  <xs:sequence>
    <xs:element name="GID" type="jxta:JXTAID"/>
    <xs:element name="MSID" type="jxta:JXTAID"/>
    <xs:element name="Name" type="xs:string" minOccurs="0"/>
    <xs:element name="Desc" type="xs:anyType" minOccurs="0"/>
    <xs:element name="Svc" type="jxta:serviceParam" minOccurs="0"
maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
```

Table 3: Peer Group Advertisement

2.5.3.5.3 Module Class Advertisement

Module Class Advertisement documents the existence of a module class by its unique ID (MCID), optional name (Name) and description (Desc) can also be provided. Table 4 describes the schema of a module class advertisement.

```
<xs:element name="MCA" type="jxta:MCA"/>

<xs:complexType name="MCA">
  <xs:sequence>
    <xs:element name="MCID" type="jxta:JXTAID"/>
    <xs:element name="Name" type="xs:string" minOccurs="0"/>
    <xs:element name="Desc" type="xs:anyType" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
```

Table 4: Module Class Advertisement

2.5.3.5.4 Module Specification Advertisement

The Module Specification Advertisement describes the specifications of a module and contains all the required information to invoke a module. A JXTA module is an abstraction that represents an implementation of a specific behaviour in the JXTA environment. All peers group services such as discovery and membership are modules. This advertisement specifies an ID (MSID), a version for the module (Vers), an optional name (Name), an optional description (Desc), an optional creator name (Crtr), an optional URI for retrieving a document (SURI), optional parameter (Parm), an optional reference to the pipe advertisement for the module, so that a pipe connection can be made to the service provided by this module (ref="jxta:PipeAdvertisement), an optional proxy URI (Proxy) and an optional authenticator Spec ID (Auth). Table 5 describes the schema of a module specification advertisement.

```
<xs:element name="MSA" type="jxta:MSA"/>

<xs:complexType name="MSA">
  <xs:sequence>
    <xs:element name="MSID" type="jxta:JXTAID"/>
    <xs:element name="Vers" type="xs:string"/>
    <xs:element name="Name" type="xs:string" minOccurs="0"/>
    <xs:element name="Desc" type="xs:anyType" minOccurs="0"/>
    <xs:element name="Crtr" type="xs:string" minOccurs="0"/>
    <xs:element name="SURI" type="xs:anyURI" minOccurs="0"/>
    <xs:element name="Parm" type="xs:anyType" minOccurs="0"/>
    <xs:element ref="jxta:PipeAdvertisement" minOccurs="0"/>
    <xs:element name="Proxy" type="xs:anyURI" minOccurs="0"/>
    <xs:element name="Auth" type="jxta:JXTAID" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
```

Table 5: Module Specification Advertisement

2.5.3.5.5 Module Implementation Advertisement

The module Implementation Advertisement provides the required information to execute the implementation being described. This advertisement can be one of the implementations of a module specification. It presents an ID (MSID), the environment required to execute this module (Comp), the fully qualified class name or the source code to be executed (Code), an optional package location to be downloaded (PURI), an optional Provider name (Prov), an optional description (Desc) and an optional parameter to be used by the implementation (Parm). Table 6 describes the schema for a module implementation advertisement.

```
<xs:element name="MIA" type="jxta:MIA"/>

<xs:complexType name="MIA">
  <xs:sequence>
    <xs:element name="MSID" type="jxta:JXTAID"/>
    <xs:element name="Comp" type="xs:anyType"/>
    <xs:element name="Code" type="xs:anyType"/>
    <xs:element name="PURI" type="xs:anyURI" minOccurs="0"/>
    <xs:element name="Prov" type="xs:string" minOccurs="0"/>
    <xs:element name="Desc" type="xs:anyType" minOccurs="0"/>
    <xs:element name="Parm" type="xs:anyType" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
```

Table 6: Module Implementation Advertisement

2.5.4 JXTA Protocols

JXTA platform is based on a set of open protocols; these protocols enable the peers to:

- Discover each other.
- Advertise their services and discover other peer services.
- Communicate and dynamically route messages.
- Self-organize into peer groups, without the need of central management and regardless of their position in the network.

The JXTA protocols are independent of programming languages and also independent of transport protocols. They can be implemented on top of TCP/IP, HTTP, Bluetooth or other transport protocols. JXTA protocols are asynchronous and are based on a query/response model. The following is a brief description of the six JXTA protocols, for detailed specification of JXTA protocols refer to [8].

2.5.4.1 Peer Resolver Protocol (PRP)

The implementation of this protocol provides a facility for peers to send a query to one or more peers and receive responses. It is the mechanism by which a peer can send a query to one or more peers, and receive a response (or multiple responses) to the query. The PRP implements a query/response protocol. The response message is matched to the query via a unique id included in the message body. Queries can be directed to the whole group or to specific peers within the group.

2.5.4.2 Peer Discovery Protocol (PDP)

The implementation of this protocol provides facilities for peers to advertise their resources and discover resources from other peers. Peer resources such as peer groups, peer pipes and service are described in XML documents and are published as advertisements.

2.5.4.3 Peer Information Protocol (PIP)

This protocol provides a message to retrieve the status, capabilities and other information from another peer.

2.5.4.4 Peer Binding Protocol (PBP)

This protocol provides the mechanism for peers to communicate through a virtual channel or pipe. The protocol binds the end points between peers and is a means for peers to exchange messages.

2.5.4.5 Endpoint Routing Protocol (ERP)

ERP provides a set of messages to find routing information for peers. When a peer wants to send a message to another peer, if the route is not available in cache, it is discovered through route queries. The response to route queries contains the Peer ID of the destination, the Peer ID and the advertisement information of the router that know the route and a sequence of peer relays.

2.5.4.6 Rendezvous Protocol (PVP)

This protocol provides mechanisms to subscribe or be a subscriber to a propagation service. Within a peer group, a peer takes the responsibility of propagating messages among peers. This protocol is used by both PRP and PBP for purposes of propagation.

2.5.5 JXTA and Security

JXTA includes many built-in security features, which can provide a base for a secure application. JXTA platform provides Secure Transport Layer (TLS) as a medium for secure communication. TLS is based on public key technology and creates secure communications among pipes. TLS requires a certificate to operate; each peer generates its own certificate and becomes its own Certificate Authority (CA). Also a Peer ID and a password protect each peer from intruders [9].

2.6 Web Services

The Stencil Group defines web services as:

"Loosely coupled, reusable software components that semantically encapsulate discrete functionality and are distributed and programmatically accessible over standard Internet protocols."

Basically, Web Services are services that are offered by a service provider over the Web. The service an be an application or business logic that is accessible through internet protocols. Web Service technology enables the exchange and interaction of data and function in a distributed computing fashion. The technology enablers for Web Services are:

2.6.1 WSDL

The services provided by Web Service providers describe their services in a standard XML document called: Web Services Definition Language (WSDL). WSDL describes the location of the service and the operation it provides. The focus of the WSDL document is the functional information of one service. QoS properties of a service are not provided in WSDL and the definition does not provide a facility for that. One motivation of this thesis is to give the Web Service providers an opportunity to publish additional QoS information on top of functional values. This proposed framework also provides tools to users to query for specific QoS.

For a detailed definition of WSDL XML schema see [10]. The main structure of WSDL is presented in Table 7. The most important element of WSDL is *portType*. It defines the Web Service, the operations that can be performed and the messages that are exchanged.

```
<definitions>
<types>
   definition of types........
</types>
<message>
   definition of a message....
```

```
</message>
<portType>
  definition of a port.......
</portType>
<binding>
   definition of a binding....
</binding>
</definitions>
```

Table 7: WSDL Definition Main Elements

2.6.2 SOAP

SOAP stands for Simple Object Access Protocol; it is an XML based protocol for accessing Web Services over HTTP.

A SOAP message content is shown in Table 8.

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
<soap:Header>
  ...
  ...
</soap:Header>
<soap:Body>
  ...
  ...
  <soap:Fault>
    ...
    ...
  </soap:Fault>
</soap:Body>
</soap:Envelope>
```

Table 8: SOAP Message Main Elements

For full details of the SOAP definition see [11].

2.6.3 UDDI

UDDI stands for Universal Description, Discovery and Integration, it is a directory that stores the information about all Web Services. It uses WSDL to describe interfaces to a Web

Service. Web Services providers need to register their services in UDDI so that the clients can search and find their required services. UDDI enables providers to find customers and collaborate with other providers. UDDI is an initiative for businesses to perform e-commerce transactions.

2.7 Web Service Composition

Web Service composition has gained much attention over the past few years as collaboration and integration among suppliers, partners and consumers is becoming inevitable. Enterprise Application Integration (EAI) and Business-to-Business (B2B) application will greatly advance if they are able to dynamically provide integrated and aggregated services. Web Services provide some level of integration to businesses, but a business process involves transaction management, state tracking, fault management and other functionality.

Some research has been done toward standardization of a language that defines the aggregation and composition of Web Services to fulfill a business process execution.

This involves defining how the Web Services interact with each other, including business logic and execution order of the interactions (orchestration) and tracking the sequence of messages among different parties such as customers, partners and businesses (choreography).

Figure 7illustrates the high level choreography of businesses and partners interacting and exchanging messages in order to fulfill the business process of purchasing a commodity from a vendor [12]:

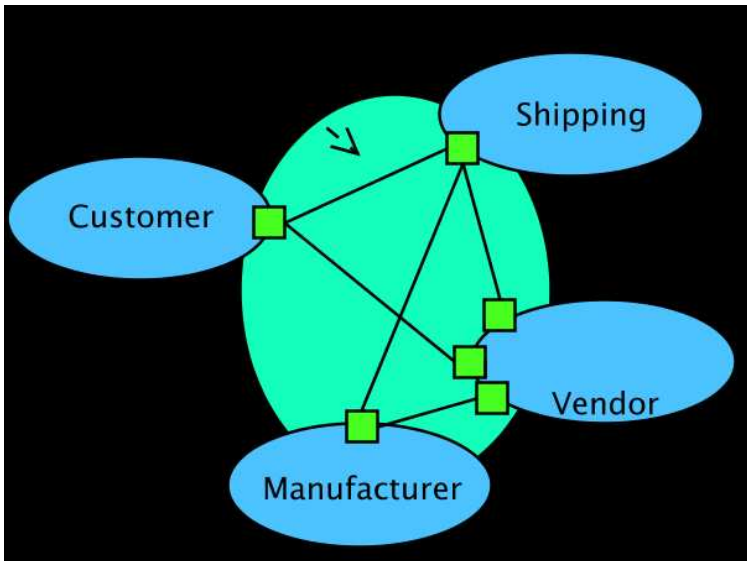


Figure 7: Web Service Choreography

Efforts towards this goal have resulted in various works, two approaches has been forthcoming. One is the definition of a language that defines a business process, such as BPML, XLANG, WSFL and BPEL where the latter is the evolution of the first three and has gained much attention.

All of these languages are used to orchestrate a business process in a static way. The designer of business processes is required to know exactly which service provider the process is going to interact with. One motivation of this thesis is to provide a tool for clients to choose among a set of service providers that offer similar services. This selection is going to be dynamic at run time and based on a set of criteria that the client specifies.

Since the focus of this thesis is the syntax-based approach, we will briefly describe BPML, XLANG and WSFL and more details will be provided for BPEL.

For an overview of comparison between these methods see [13].

2.7.1 BPML

Business Process Modeling Language (BPML) is an XML based specification for modeling of business processes and is maintained by Business Process Management Initiative (http://www.bpmi.org/). BPML supports the abstract definition of synchronous and asynchronous transactions, dataflow, messaging, scheduling events, security roles and exceptions.

2.7.2 XLANG

XLANG [14] is another XML based meta-language for defining business processes. XLANG creates stateful business processes based on interactions between Web Services; it is an extension of WSDL where it provides both a model of orchestration for a service and a model of collaboration between orchestrations. It supports control and data flow, messaging, contracts between businesses. XLANG was defined by Microsoft.

2.7.3 WSFL

Web Services Flow Language (WSFL) [15] is a similar language to XLANG from IBM. WSFL supports composition and choreography of Web Services. WSFL also relies on WSDL documents of Web Services that it intends to choreograph to business processes. WSFL provides two types of composition:

- *FlowModel*: The first type allows specifying an executable business process.
- *GlobalModel*: The second type allows specifying business collaborations.

2.7.4 BPEL

We will describe BPEL [16] in more detail since it is used in this thesis. The reason for choosing BPEL is because BPEL supersedes both WFSL and XLANG. It is endorsed by major vendors like IBM, Microsoft, SAP and BEA and is progressing toward standardization.

- BPEL defines a language for describing the behavior of a business process based on interactions between the process and its partners.
- The interaction with each partner occurs through Web Service interfaces, and the structure of the relationship at the interface level is encapsulated in *partner links*, defined in section 2.7.4.2.1. For this matter BPEL relies on other services' WSDL documents.
- The BPEL process defines how multiple service interactions with these partners are coordinated to achieve a business goal, as well as the state and the logic necessary for this coordination.
- BPEL provides systematic mechanisms for dealing with business exceptions and processing faults.
- Finally, BPEL introduces a mechanism to define how individual or composite activities within a process are to be compensated in cases where exceptions occur or a partner requests reversal.

BPEL is layered on top of the following specifications:

WSDL 1.1	http://schemas.xmlsoap.org/wsdl
XPath 1.0	http://www.w3.org/TR/xpath
XML Schema 1.0	http://www.w3.org/2001/XMLSchema

2.7.4.1 Example

A simple example of a business process is a loan request. A customer requests a loan, and receives a reply whether the loan has been approved or not. In this process the Web Services of a financial institution are invoked and a reply is received from the Web Service.

We will create the BPEL document for this scenario.

- The process involves two parties, customer and loan approver. They are referred to as partners in the BPEL specification.
- The process starts by receiving a message from a client, then invoking the financial institution's web service and replying to the client.
- These actions are defined as <receive>, <invoke>, and <reply> activities in BPEL, the structure activities in BPEL define the restrictions of how to run these activities, for example <sequence> activity determines the order of the execution which is receive, invoke and reply.
- BPEL also contains the WSDL descriptions for the web services that are going to be used as part of the process.

  The three WSDL documents used here are as follows:

  - Loan Definition WSDL - A unified set of messages for financial institutions that describe the loan information. See Table 9 for details of loan definition WSDL.

```
<definitions targetNamespace="http://tempuri.org/services/loandefinitions"
             xmlns:tns="http://tempuri.org/services/loandefinitions"
             xmlns:xsd="http://www.w3.org/2001/XMLSchema"
             xmlns="http://schemas.xmlsoap.org/wsdl/">
```

```
  <message name="creditInformationMessage">
     <part name="firstName" type="xsd:string"/>
     <part name="name" type="xsd:string"/>
     <part name="amount" type="xsd:integer"/>
  </message>

  <message name="loanRequestErrorMessage">
     <part name="errorCode" type="xsd:integer"/>
   </message>

 </definitions>
```

Table 9: Loan Definition WSDL

- Loan Approver WSDL - A financial institution that provides the loan approval service WSDL. It contains one operation that is <approve>, See Table 10 for details of the WSDL definition.

```
<definitions targetNamespace="http://tempuri.org/services/loanapprover"
                       xmlns:tns="http://tempuri.org/services/loanapprover"
                       xmlns:xsd="http://www.w3.org/2001/XMLSchema"

xmlns:loandef="http://tempuri.org/services/loandefinitions"
                       xmlns="http://schemas.xmlsoap.org/wsdl/">

   <import namespace="http://tempuri.org/services/loandefinitions"
            location="http://localhost:8080/bpws-
samples/loanapproval/loandefinitions.wsdl"/>

   <message name="approvalMessage">
     <part name="accept" type="xsd:string"/>
   </message>

   <portType name="loanApprovalPT">
     <operation name="approve">
       <input message="loandef:creditInformationMessage"/>
       <output message="tns:approvalMessage"/>
       <fault name="loanProcessFault"
              message="loandef:loanRequestErrorMessage"/>
     </operation>
   </portType>

 <binding ...> ... </binding>
 <service name="LoanApprover">....</service>
</definitions>
```

Table 10: Load Approver WSDL definition

- Loan Approval WSDL - The input and output messages for the process itself where the portTypes to the above services are defined. It also describes the serviceLinkTypes that links the customer to the process and the process to the loan approver. See Table 11 for details of loan approval WSDL definition.

```
<definitions
       targetNamespace="http://loans.org/wsdl/loan-approval"
       xmlns="http://schemas.xmlsoap.org/wsdl/"
       xmlns:slnk="http://schemas.xmlsoap.org/ws/2002/06/service-link/"
       xmlns:xsd="http://www.w3.org/2001/XMLSchema"
       xmlns:lns="http://loans.org/wsdl/loan-approval"
       xmlns:apns="http://tempuri.org/services/loanapprover">

   <import namespace="http://tempuri.org/services/loanapprover"
           location="http://localhost:8080/bpws-
samples/loanapproval/loanapprover.wsdl"/>
   <import namespace="http://tempuri.org/services/loandefinitions"
           location="http://localhost:8080/bpws-
samples/loanapproval/loandefinitions.wsdl"/>

   <slnk:serviceLinkType name="loanApprovalLinkType">
     <slnk:role name="approver">
       <portType name="apns:loanApprovalPT"/>
     </slnk:role>
   </slnk:serviceLinkType>

   <service name="loanapprovalServiceBP"/>
 </definitions>
```

Table 11: Loan Approval WSDL definition

- Table 12 presents the BPEL document that creates the process with references to the WSDL documents.

```
        <process name="loanApprovalProcess"
          targetNamespace="http://acme.com/simpleloanprocessing"
          xmlns="http://schemas.xmlsoap.org/ws/2002/07/business-process/"
          xmlns:lns="http://loans.org/wsdl/loan-approval"
          xmlns:loandef="http://tempuri.org/services/loandefinitions"
          xmlns:apns="http://tempuri.org/services/loanapprover">
          <partners>
            <partner name="customer"
               serviceLinkType="lns:loanApproveLinkType"
               myRole="approver"/>
             <partner name="approver"
               serviceLinkType="lns:loanApprovalLinkType"
               partnerRole="approver"/>
           </partners>
           <containers>
             <container name="request"
                  messageType="loandef:CreditInformationMessage"/>
             <container name="approvalInfo"
                 messageType="apns:approvalMessage"/>
           </containers>
          <sequence>
            <receive name="receive1" partner="customer"
                    portType="apns:loanApprovalPT"
                    operation="approve" container="request"
                    createInstance="yes">
            </receive>
            <invoke name="invokeapprover"
                        partner="approver"
                        portType="apns:loanApprovalPT"
                        operation="approve"
                        inputContainer="request"
                        outputContainer="approvalInfo">
            </invoke>
            <reply name="reply" partner="customer"
                        portType="apns:loanApprovalPT"
                        operation="approve" container="approvalInfo">
            </reply>
          </sequence>
     </process>
```

Table 12: Loan process BPEL document

Loan request process is an example that demonstrates how various activities can be combined to provide a business process.

2.7.4.2 Definitions

Figure 8 shows an overview of the main BPEL definitions [17].

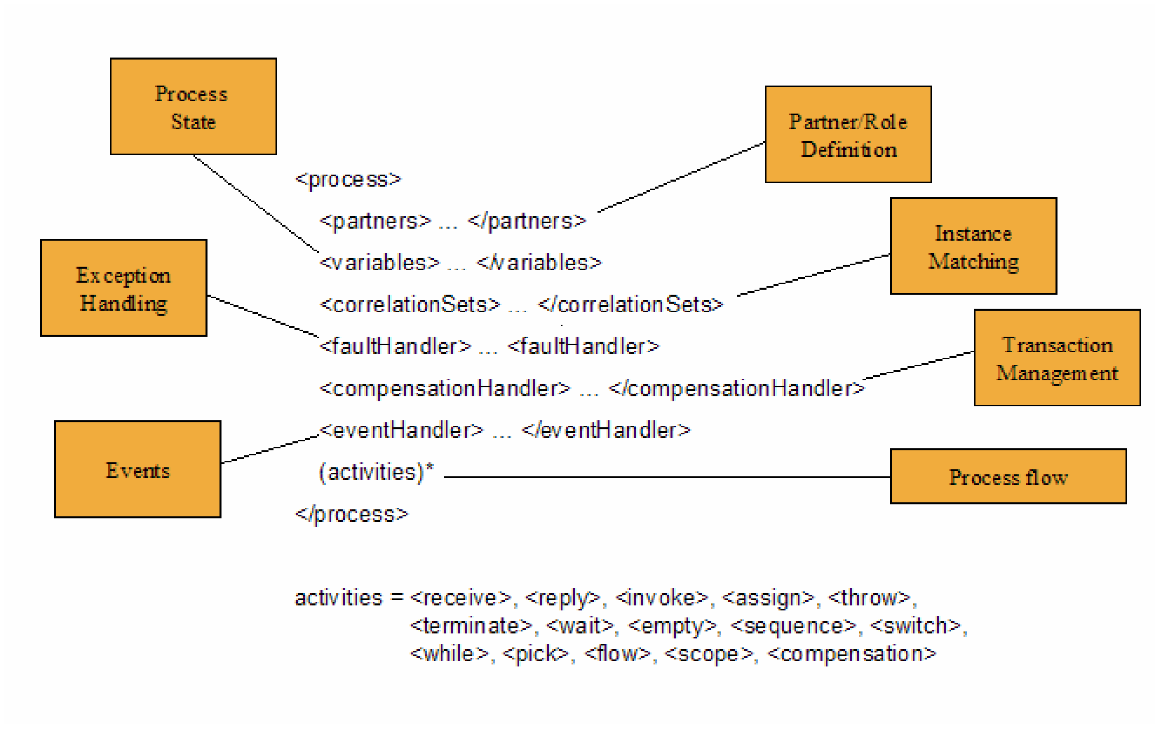

Process State
Partner/Role Definition
Instance Matching
Exception Handling
Transaction Management
Events
Process flow
<process>
<partners> ... </partners>
<variables> ... </variables>
<correlationSets> ... </correlationSets>
<faultHandler> ... <faultHandler>
<compensationHandler> ... </compensationHandler>
<eventHandler> ... </eventHandler>
(activities)*
</process>
activities = <receive>, <reply>, <invoke>, <assign>, <throw>, <terminate>, <wait>, <empty>, <sequence>, <switch>, <while>, <pick>, <flow>, <scope>, <compensation>

Figure 8: Overview of BPEL main definitions

The following subsections describe the main elements of the BPEL specification, for details on BPEL see [16].

2.7.4.2.1 Partner Links

The services that the business processes interact with are partner links.

One *partnerLinkType* can specify more than one partner links.

2.7.4.2.2 Partner Link Type

A partner link type determines the role of each service in a conversational relationship. Each role specifies only one WSDL type. *partnerLinkType* is defined by the extensibility mechanism of WSDL 1.1 as a new definition type to be placed as an immediate child element of a <wsdl:definitions> element in all cases. This allows reuse of the WSDL target namespace specification and its import mechanism to import *portTypes*. The Syntax of a *partnerLinkType* is presented in Table 13:

```
<definitions name="ncname" targetNamespace="uri"
     xmlns="http://schemas.xmlsoap.org/wsdl/"
     xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partner-
link/">
  ...
  <plnk:partnerLinkType name="ncname">
    <plnk:role name="ncname">
      <plnk:portType name="qname"/>
    </plnk:role>
    <plnk:role name="ncname">?
      <plnk:portType name="qname"/>
    </plnk:role>
  </plnk:partnerLinkType>
  ...
</definitions>
```

Table 13: PartnerLinkType Syntax in BPEL

2.7.4.2.3 Business Partner

The capabilities needed from a business partner are described in the partner element. *partner* definitions are not allowed to overlap. Only one *partner* link can appear in a partner definition. The Syntax of a *partner* is presented in Table 14:

```
<partners>
  <partner name="ncname">+
    <partnerLink name="ncname"/>+
  </partner>
</partners>
```

Table 14: Business Partner Syntax in BPEL

2.7.4.2.4 Message Properties

Message properties represent the data in the BPEL. The data is either application related or protocol related where protocols can be business or infrastructure type. A *property* definition creates a globally unique name. The examples of properties are social security number, price, response latency and so on. The Syntax of a property is shown in Table 15:

```
<wsdl:definitions name="ncname"
   xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/business-
process/">
    <bpws:property name="ncname" type="qname"/>
    ...
</wsdl:definitions>
```

Table 15: Message Property syntax in BPEL

2.7.5 BPWS4J Engine

IBM AlphaWorks has developed the IBM Business Process Execution Language for Web Services Java Run Time (BPWS4J) is a platform that validates and executes the BPEL. For each process, the BPWS4J engine takes:

- A BPEL document that describes the process to be executed.
- A WSDL document (without binding information) that describes the interface that the process will present to clients.
- WSDL documents that describe the Web Services that the process may or will invoke during its execution.

The engine includes a tool that validates the documents against their specifications, it also provides an editor to better view the documents. The BPWS4J server supports the messaging and transactions among various service providers.

This thesis uses BPWS4J engine to execute a BBEL document that is generated as part of the framework functionality.

2.8 Summary

This chapter provides an overview of the technologies that are used throughout this thesis. It describes P2P architecture and provides more details on JXTA, which is a P2P implementation and has been chosen by the proposed framework in this thesis. The Web Services and its enabling technologies are reviewed. A brief summary of Web Service composition definition languages and a more detailed description of BEPL were provided. These technologies are going to be used in the framework proposed in this thesis and they are referred to in chapters 4 and 5 extensively.

3 State of the Art

This chapter provides a review and analysis of the state of the art in dynamic Web Service composition based on QoS. An overview and analysis of QoS based dynamic Web Service composition is presented. It provides an analysis of several approaches towards solving this issue. This chapter then specifically selects 3 related works that aligns more with the goals of this thesis, and provides a detailed overview of them. A conclusion is drawn at the end that leads the thesis to the chosen architecture that is provided in the next chapter.

3.1 QoS based Dynamic Web Service composition

Web Service composition is the technology that enables businesses to provide business logic by integrating various Web Services within their organization or by collaborating with other organizations. Travel planning is an example of a Web Service composition. It involved the integration of several Web Services such as a booking a flight, booking a hotel, car rental and even booking different activities within the city. Designing the process is the first element of creating a composite service. Web Service composition language definitions described in section 2.7 are aiming at facilitating the design process. The other advantage of developing such languages is that they make the automation of the process execution possible. In section 2.7 we described some of these languages: XLANG, BPML and BPEL. BPEL was specifically described in section 2.7.4 as it has gained wider attention than the others. They describe the modelling of a business process in a syntactic way. We introduced BPWS4J engine in section 2.7.5, which executes a BPEL based business process. In fact Oracle has included a composite engine based on BPEL as part of its service architecture platform and several companies are building such composite engines.

Even with the standardization of one of such languages, the design of a business process is a challenging and time consuming task. The composition of Web Services relies heavily on the WSDL description of the participating Web Services. The process designers have to decide which Web Service provider's services to choose at design time. If a business decides to switch betweens providers, the design process has to change accordingly.

On the other hand, service providers publish different WSDL documents. Even for similar functionality, but they might define different operation names or signatures. That is one reason the changing of a process definition language requires more effort. An agreement on Web Service providers with similar services to provide their WSDL document based on a template, greatly saves time and effort for businesses.

Even if the same functionality and WSDL is agreed upon for similar services, providers follow different QoS. QoS specifies qualitative and quantitative aspects of a Web Service. Web Service composition designers and users will benefit significantly in their selection of Web Services if they know the QoS associated with a Web Service. One approach for providing QoS information of services has resulted in definitions of XML specifications that provide the details of QoS. Web Service Offerings Language (WSOL) [18] is one of such languages. WSOL enables formal specification of classes of service in a Web Service. Classes of service are referred to as service offerings of a Web Service. WSOL is a thorough and strongly typed language definition that allows specification of functional and QoS constraints and access rights and is useful in management of Web Services. Web Service Level Agreement language (WSLA), is another such language [19]. These languages provide a static approach to service definition of a Web Service. Service providers are strictly dependant on the definitions of the chosen language and less flexibility is provided. Several research projects have resulted in more dynamic approach to this issue and that is the goal of this thesis as well. A framework based on Semantic Web Service METEOR-S (Managing End-To-End OpeRations) is developed [20]. METEOR-S is a framework based on Semantic Web Service composition developed at University of Georgia. An overview METEOR-S is provided in section 3.3. A research has been conducted in parallel with this thesis, which enhances the METEOR-S framework by adding QoS characteristics to its Web Service discovery [21]. This research is examined in more detail in section 3.3.2. “Quality Driven Web Services Composition” [22], is another effort on taking QoS into account when selecting services. This project is based on Self_Serv [23], a P2P Web Service Orchestration middleware. A closer examination of this project is provided in section 3.2.

In order to enable dynamic Web Service a search and discovery mechanism is required. UDDI is used as a standard technology for dynamic search and discovery of Web Services. However, UDDI uses a central system. As the number of members in UUDI registry increases like other central distributed the performance will be affected. Sitthichai Laoveerakul et al. [25] has outlined the flaws of the original design of UDDI as: “

- First, if UDDI server fails, all the registered services will be gone.
- With only UDDI, you cannot control the flow of running multiple services continuously.

- In addition, for a normal user, registering with UDDI seems complicated and users need to do it manually every time they create a Web Service.
- Finally, UDDI server does not monitor the availability of the computing resources, which contains the underlying Web Services. Consequently, it does not guarantee if the registered Web Services would be readily executable.”

P2P environments also allow the creation of communities. This will provide the opportunity of categorizing Web Services based on their services. This will facilitate the flexibility of dynamic Web Service selection significantly and reduces the effort of business process designer when it comes to changes. Babak Esfandiari, Vladimir Tosic in defining the requirements of Web Service composition [26] have proposed a P2P frame work that creates communities based on schema definitions and describes the advantages as: “

- More effective search, as the schema provides structured metadata that can restrict the search domain to similar files.
- More efficient and scalable search, as the central registry bottleneck can be eliminated in favor of a Gnutella-type query propagation.”

Creation of communities in P2P environment also provides a scoping and monitoring environment and limited access among communities provide enhanced security which is another goal of this thesis.

METEOR-S Web Service Discovery Infrastructure (MWSDI) [27] is a project based on METEOR-S. MWSDI proposes a P2P network for discovery of services as opposed to previously based UDDI based discovery engine in METEOR-S. Now this research is part of the METEOR-S framework, which is described in section 3.3.1.

One goal of this thesis is the integration of existing technologies with new components. Web Service composition definition and execution is a complex technology, using existing technologies allows easier adoption of clients. Documented resources on technologies are much more available. Improvements and contributions to the technologies also provide a better quality infrastructure. David Mennie [28] has proposed an infrastructure on dynamic composition of service components by using existing technologies as well. Also a thorough survey on dynamic composition is provided as part of his research.

The JXTA platform (described in section 2.5) is one of the selected technologies in this proposal. The benefits of JXTA are outlined in section 2.5.1. Less effort has been done on combining JXTA with Web Service composition. A published paper on this subject is Triana, an open source problem-solving environment developed at Cardiff University that combines an intuitive visual interface with powerful data analysis tools. The paper proposes the use of Triana project for Web Services composition by using the JXTA component in Triana [29]. The following subsections provide more detail and analysis of some of the more relevant researches in this field.

3.2 Self-Serv

Self-Serv is a middleware infrastructure, which is based on P2P orchestration for Web Service composition [23]. Two main concepts used in Self-Serv for service compositions are the *composite service* and *service container*. A composite service can be a single service or an aggregation of single services. Self-Serv adopts the concept of state charts in composite modelling. State charts define the business logic of a composite service. The flow of operation and service invocations are described as state charts. Another concept used in Self-Serv is service containers. Service containers are dynamic services that aggregate a group of services that have common operations. They are services themselves and can be invoked by composite services. Service containers are enablers of dynamic service selection in Self-Serv, since the decision of the service invocation is made at run time. Self-Serv has adopted a P2P environment for the service composition orchestration, a performance evaluation experiment has shows this will provide greater scalability. The basic elements that support this orchestration are the *state coordinators* and *routing tables*. A state coordinator is generated for each state in a state chart, it receives notifications, informs the service labelling when all conditions are met and notifies the coordinators when the service execution is complete. Routing tables provide the routing information to the service coordinators.

Figure 9 [24] illustrates the layered architecture of Self-Serv. It is consists of five layers: Service, Conversational, Directory, Communication and User layers. Service layer contains a pool of services and containers. The coordination classes are assumed in this layer. The conversational layer enables business interactions and message exchanges by providing service templates. The directory layer stores the meta-data about the services and containers. User layer locates services by calling this layer's operations. Communications between services are based on SOAP (described in section 2.6.2) messaging over the Internet. User layer is the infrastructure's user interface.

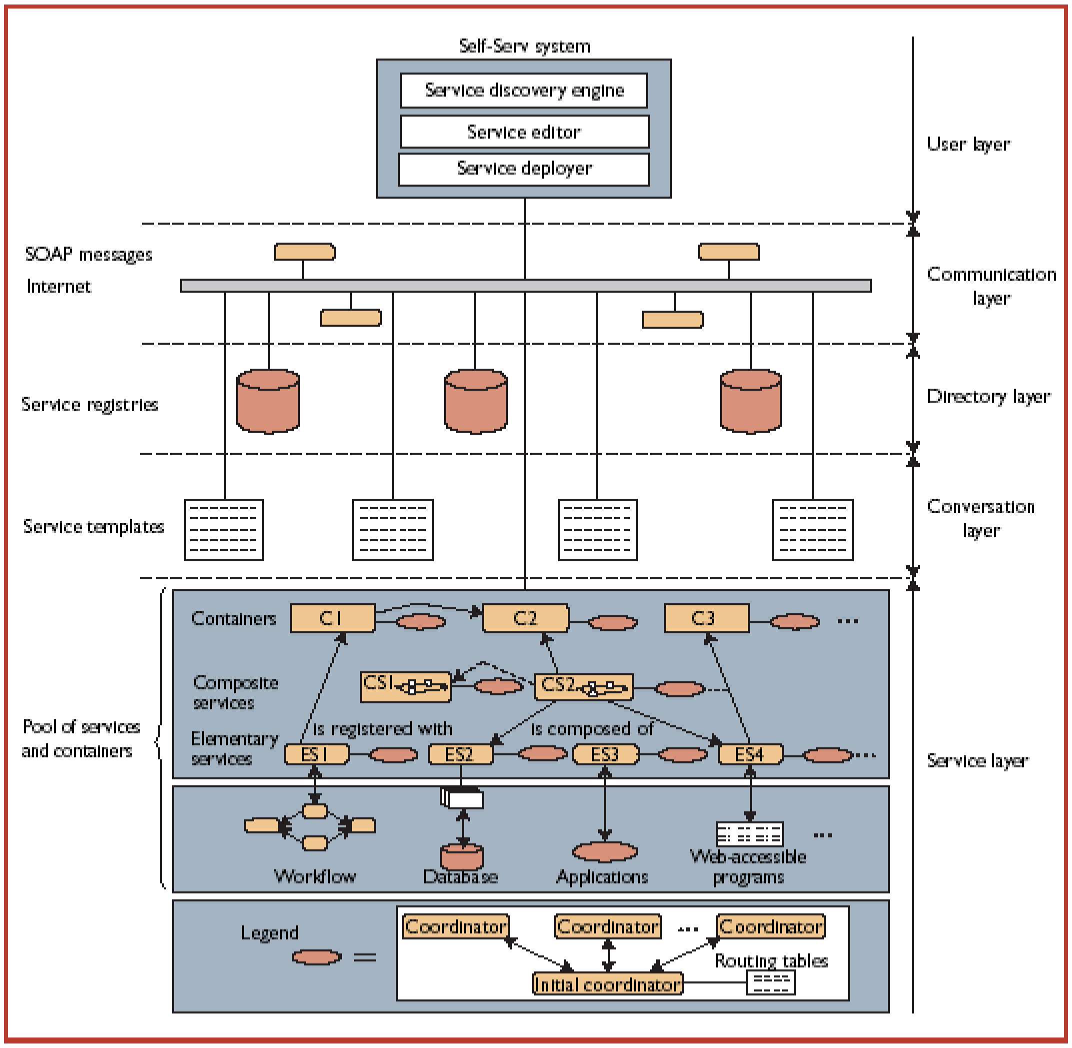


Figure 9: An Architecture view of Self-Serv

3.2.1 Quality Driven Web Services Composition based on Self-Serv

Research specifically on QoS has been done based on Self-Serve project [22]. The focus of this research is dynamic selection of services based on various criteria (price, duration, reliability). It will also advocate the selection of services based on the criteria of the composite service itself. A quality model is introduced that service selection takes place based on this model. This model takes qualities such as execution price, execution duration, reputation, reliability, and availability and performs aggregation functions on them to find the quality vector of service. Each of these attributes uses a specific algorithm for calculating the service's QoS. Another model is proposed to calculate the quality of vector of a composite service based the composite service execution price, execution duration, reputation, reliability, and availability. After the calculation of all quality vectors an optimisation algorithm is introduced to select the optimum execution plan. This approach not only takes the QoS of each service into account, also takes a global approach by considering the QoS of service compositions hence they adopt a global planning model.

3.2.2 Conclusions

One main advantage of our proposal to this work is that we have adopted the technologies such as BPEL and JXTA that have widely been accepted in the distributed computing world. The complexity of Self-Serv makes it harder for users to deploy it.

Service containers in Self-Serv provide facility for creating service communities. Service communities share the same operations with different QoS. We have also adopted the similar approach since it will bring much simplicity in composing large and dynamic collection of Web Services.

The quality driven approach built all the calculations upon a limited number of QoS properties that are execution price, execution duration, reputation, reliability, and availability. This thesis proposes QoS schema that is a flexible way to define any kind and number of QoS, without any necessary code changes. It is possible to extend the schema to provide an algorithm associated with each QoS definition. Hence the calculations can be complementary to our approach since the QoS properties can be of any type and value and the user can generically define them as name and value pairs. This can be regarded as a future work for enhancing the framework.

3.3 METEOR-S

One of the research projects focuses at the LSDIS (Large Scale Distributed Information Systems) lab at University of Georgia is Web Services and processes specifically in applying semantics in annotations, QoS, Discovery, composition and execution. Their work has resulted in a framework called METEOR-S (Managing End-To-End OpeRations) and various projects have been implemented to improve and enhance METEOR-S. The more relevant research to this thesis on METEOR-S is selected and examined in following subsections.

3.3.1 METEOR-S Web Service Composition Framework – MWSCF

MWSCF is a framework based on semantic descriptions of all Web related tasks, we may call such process as Semantic Web Processes (SWP) [30]. Figure 10 illustrates an architectural view of MWSCF. It is consists of four major components: Process Builder, Discovery Infrastructure (MWSDI), XML repositories and the Process Execution Engine. The process builder has a graphical user interface to design/open process templates. The process builder passes the template to the process generator. The discovery infrastructure of MWSCF was originally based on UDDI search and discovery registries. MWSDI is an enhanced infrastructure that utilizes P2P features for discovery of Web Services. The process generator MWSDI and data in XML repositories are used to convert the template into an executable process. The generated executable process is then executed using a process execution engine. The XML repositories in the architecture are used to store ontologies, activity interfaces and process templates.

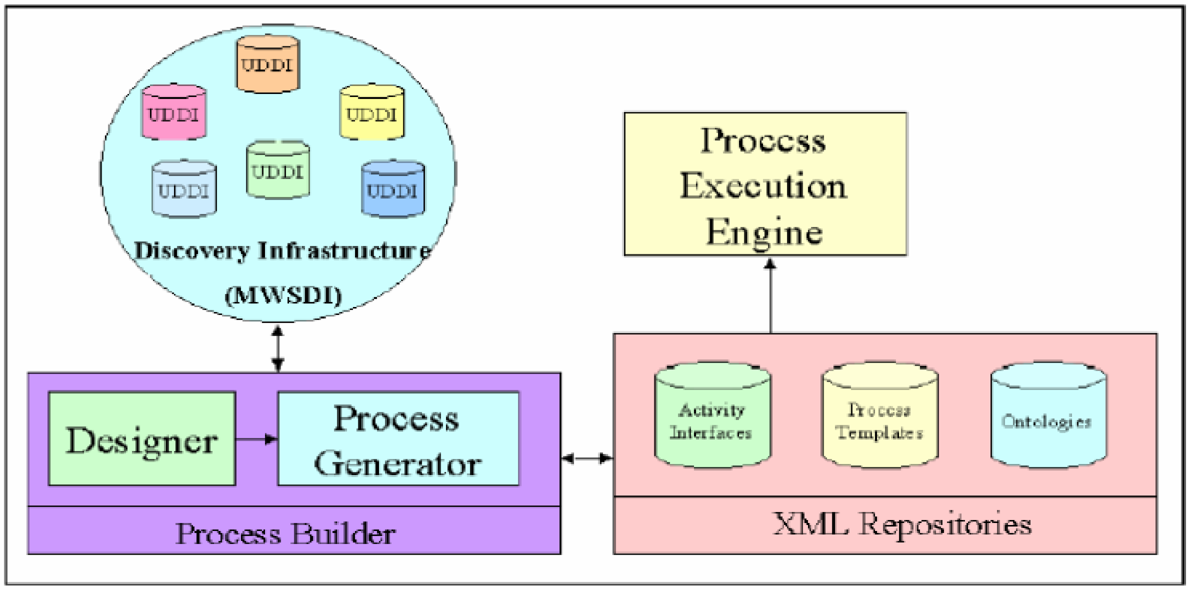


Figure 10: Web Service Composition Framework MWSCF

3.3.2 Constraint Driven Web Service Composition in METEOR-S

This project is also one of the enhancements of METEOR-S [21]. The focus of this research is service selection based on optimisation. An analysis of a service selection is performed in three phases with using three modules: service discovery, constraint analyser and optimiser, Figure 11 illustrates the order of the three phases.

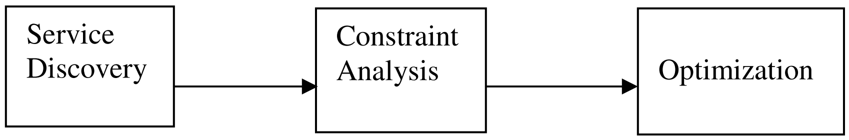


Figure 11: Three phases of Selection Process

These modules calculate time, reliability, cost and other properties to select the best optimal result. The service binding takes place after the calculations and is not dynamic.

3.3.3 Conclusions

The major difference between METEOR-S and this thesis approach is the use of semantic Web as opposed to static Web Service definition language. This required METEOR-S to develop a process engine versus our proposal will use the BPEL composite engines. Since the goal of this thesis is to use existing technologies using a BPEL composite engine is a more suitable choice. Several companies implemented BPEL engines and Oracle has one as part their enterprise solution. METEOR-S as advancement of its framework has replaced the UDDI registry component with MWSDI, which is a P2P, based infrastructure. A paper is published that explains the motivation of this enhancement [27]. MWSDI paper is published recently and a prototype is developed almost parallel in time with this thesis. This will more affirm our proposal for using P2P environment as the search and discovery bases.

The modules introduced in the constraint driven project of METEOR-S can be complimentary to our proposal, for example a service entity can be added to the JXTA network that utilizes the algorithms proposed in [21] to calculate the most optimised match. The optimisation is a future work to this thesis described in section 8.3.2.

3.4 General Conclusions

Based on problems, goals and analysis provided in this chapter, several assumptions and decisions are made with regards to this thesis proposal. In this proposal we assume that similar Web Services are categorized in a P2P community. The services in communities share the same WSDL document. Therefore they have the same pragmatic interfaces, although their underlying implementation might be different. This approach reduces the time and efforts used on designing of a Web Service composition process model. The concept of communities in P2P networks provides major benefits [26] such as limiting the number of queries within a community, scoping and monitoring features and also facilitates imposing of enhanced security features.

To address the problem of QoS enabled Web Service composition, a template is proposed and its schema is defined in chapter 5. This template is used both by Web Service providers and Web Service Composition clients, the first one define their functional and QoS information in the template, the latter provide the Web Service selection criteria by using the template. The template features a generic and flexible way of defining the QoS information. It uses name/value pair for associating a QoS property name and its value. It also needs to define the type of the value and the kind of comparison to be performed on it. The details of this design are provided in section 5.1. This approach with the above assumption reduces the work of the process designer considerably when a change is required in the selection process.

Next chapter 4 proposes an architecture that takes the above decisions into consideration.

3.5 Summary

This chapter explained the state of the art in dynamic Web Service composition. It provided a more in-depth view of the problem and examined several approaches and projects for solving this issue. Based on the analysis and examination some conclusions were drawn that are the basic enables of the framework proposed in this thesis. The next chapter provides the architecture of this framework.

4 Architecture

In this chapter the detailed architecture of the Web Service Composition QoS Enabler Framework (WSQEF) is presented. The following section 4.1 describes the requirements of the design. An overview of the third party components used in this platform is provided in section 4.2. Then a diagram is provided to illustrate all the components in the framework and their interactions followed by subsections that provide detailed descriptions of the components section 4.3.1. The last subsection 4.3.2 provides interaction diagrams describing the sequence of execution.

4.1 Requirement Analysis

WSQEF is a platform with the design goal of enabling the late binding of a series of services that can execute a composite service based on the specified criteria for each of the Web Service. The following lists the requirements that WSQEF will try to fulfill:

- Providing the service provider and client with a secure environment.
- Ability to provide QoS in a generic way for more flexibility.
- Ability to add service providers in a scalable way.
- An end-to-end solution to compose two or more web services.
- Enabling clients to select the services that best match their functional and QoS requirements.
- Enabling service providers to provide their QoS information in the network.
- Ability to find required web services with reasonable performance.

- Integrating the existing technologies with the new components to exploit latest technologies and standards.

Figure 12 illustrates the system requirement in an abstract way. At a high level, this system requires creating a Dynamic Web Service composition based on the client's functional and QoS criteria.

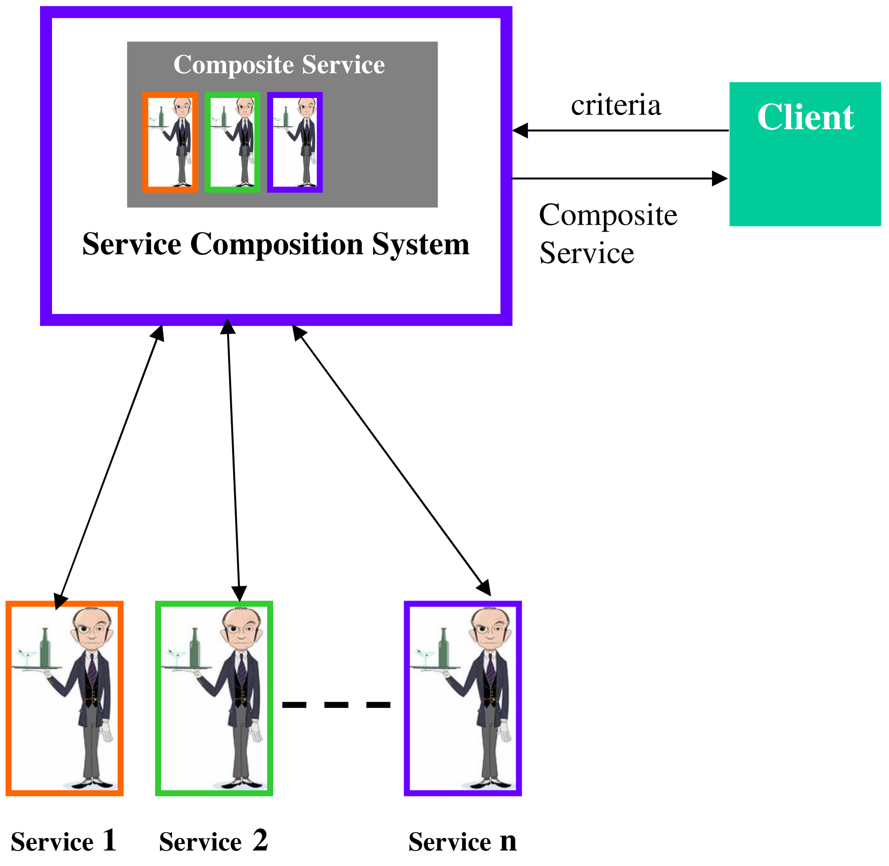


Figure 12: Web Service Composition

This thesis proposes a solution that runs in a P2P environment in order to meet some of the main requirements. Enhanced security is one main reason since most of P2P implementations provide an overlay network where security mechanism can be enforced.

Also P2P provides an environment that makes service discovery and publishing more scalable. Figure 13 models the behavior of the system within a P2P environment in the form of a UML Use Case Diagram. The P2P network is the core of the system and the components are service entities in the network.

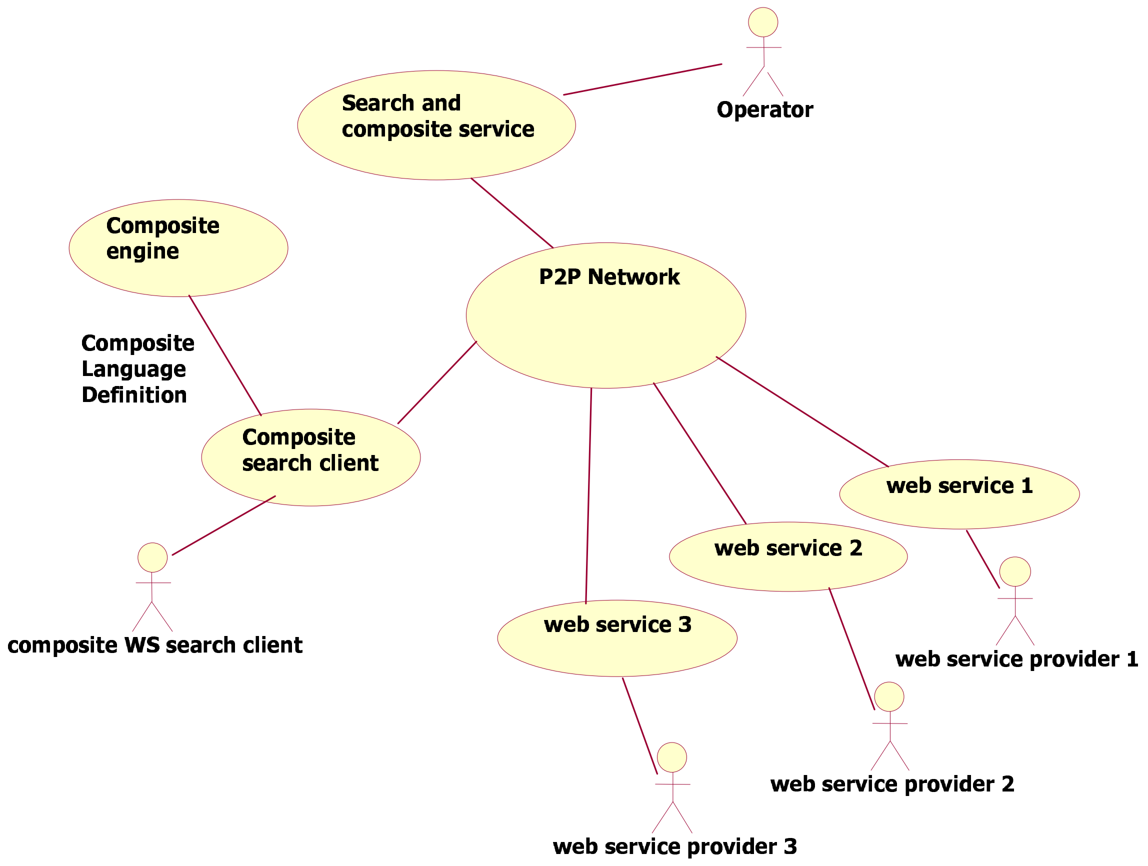


Figure 13: A Use Case Diagram of WSQEF framework

The main actors in this system are: Web Service providers and Web Service Composition Client. The following subsections provide the sequence of interactions that is expected between components.

4.1.1 Web Service Providers

The Web Service providers' responsibilities as part of this system are:

1. Publish Web Service they provide through the WSDL descriptions.

2. Publish the name of QoS properties they will provide values for at run time.

3. Provide functional and QoS Service Criteria of the Web Service it requires.

4. Join the P2P Network and advertise their functional Web Services along with QoS properties with the values they guarantee.

4.1.2 Web Service Composition Client

The Web Service Composition Client in order to reach a Web Service Composition with its required QoS at run time should:

1. Retrieve the WSDL documents.
2. Write the process WSDL document that describes interfaces used in the required business process. At design time the portTypes are in the form of placeholders. They will be replaced by a real portType value at run time.
3. Write the BPEL document, this BPEL document refers to the portTypes in the above document.
4. Write the composite search criteria by providing the list of Web Services that are required to be composed, the logical operator they are going to be composed based upon and the QoS property values they require for each of the Web Services.

5. Join the P2P network, find the *search and composite service* and sends the composite search request to it. Receive the search result and replaces the portType placeholders with the returned values.
6. Execute the BPEL and WSDL documents by using a composition engine.

In order to achieve the above requirements the WSQEF framework integrates and deploys third party components and the newly created components that represent the core contributions of this thesis. This framework runs in a peer-to-peer environment and uses Web Services, XML and JXTA technologies. The input to the framework is an XML document, which defines the search criteria for a set of web services that are going to be composed. The output is the completed BPEL and WSDL documents that can be executed by the IBM business process BPWS4J engine.

4.2 3rd party Components

4.2.1 JXTA

The JXTA platform described in detail in section 2.5 is proposed to be used as the P2P enabler technology. JXTA provides the P2P network shown in Figure 13 and provides the following features and functionality that fulfill some of the requirements of WSQEF framework.

- JXTA technology is a set of open protocols that allow any connected device on the network ranging from cell phones and wireless PDAs to PCs and servers to communicate and collaborate in a P2P manner [5].
- JXTA peers create a virtual network where any peer can interact with other peers and resources directly even when some of the peers and resources are behind firewalls and NATs or are on different network transports [5].
- JXTA offers the peer group concept, which defines a scope, which creates a virtual boundary of peers where an authentication policy can be enforced upon them for security.
- JXTA is platform independent and all the protocols are based on XML messaging.
- JXTA includes a set of core services such as peer pipes for communications between peers and peer discovery service.
- JXTA provides advertisements, which are XML documents that describe peers, peer groups, pipes and services in the network.

For detailed information on JXTA technology please refer to section 2.5.

WSQEF uses these capabilities to meet the requirement of providing a QoS-aware web service composition: Using JXTA provides the web service providers with a virtual overlay network where communication can be achieved regardless of being behind a firewall or use of heterogeneous transport protocols being used. Also secured communications communication can be imposed based on JXTA certificates. Each web service provider will be a peer that belongs to a specific peer group. This way providers can be categorized based on the services they provide to prevent higher traffic in the network. The service provider can use the advertisement capabilities of JXTA protocols to announce its services and the QoS it intends to provide to the users. Advertisements can be searched and discovered by using the JXTA discovery protocol. A service is implemented on a peer responsible for receiving client's criteria and searching the web service advertisements that match the QoS criteria; the communication between peers will be done by JXTA pipe services.

4.2.2 BPEL

The framework proposes BPEL as the composition language definition (shown in Figure 13) that both implements executable business processes and describes non-executable abstract processes. For a detail description of BPEL, refer to section 2.7.4.

The clients of WSQEF create process WSDL and BPEL documents that have the port types as placeholders since originally they don't know which web service in the network matches their QoS requirement.

4.2.3 BPWS4J

The framework proposed Business Process Execution Language for Web Services Java Run Time (BPWS4J) as the composite engine (shown in Figure 13). BPWS4J is a platform developed by IBM that takes in a BPEL document, a WSDL document (without binding information) that describes the interface, which the process will present to clients (partners in BPEL terms), and WSDL documents that describe the services that the process may or will invoke during its execution. From this information, the process is made available as a Web Service with a SOAP interface. A WSDL file that describes the process's interface may be retrieved from the run-time.

4.3 Web Service Composition QoS Enabler Framework (WSQEF)

Figure 14 illustrates the logical view of the components and their interactions within the framework. The following is a general description of the flow of information in the framework; a detailed description of each component is provided in the next subsections.

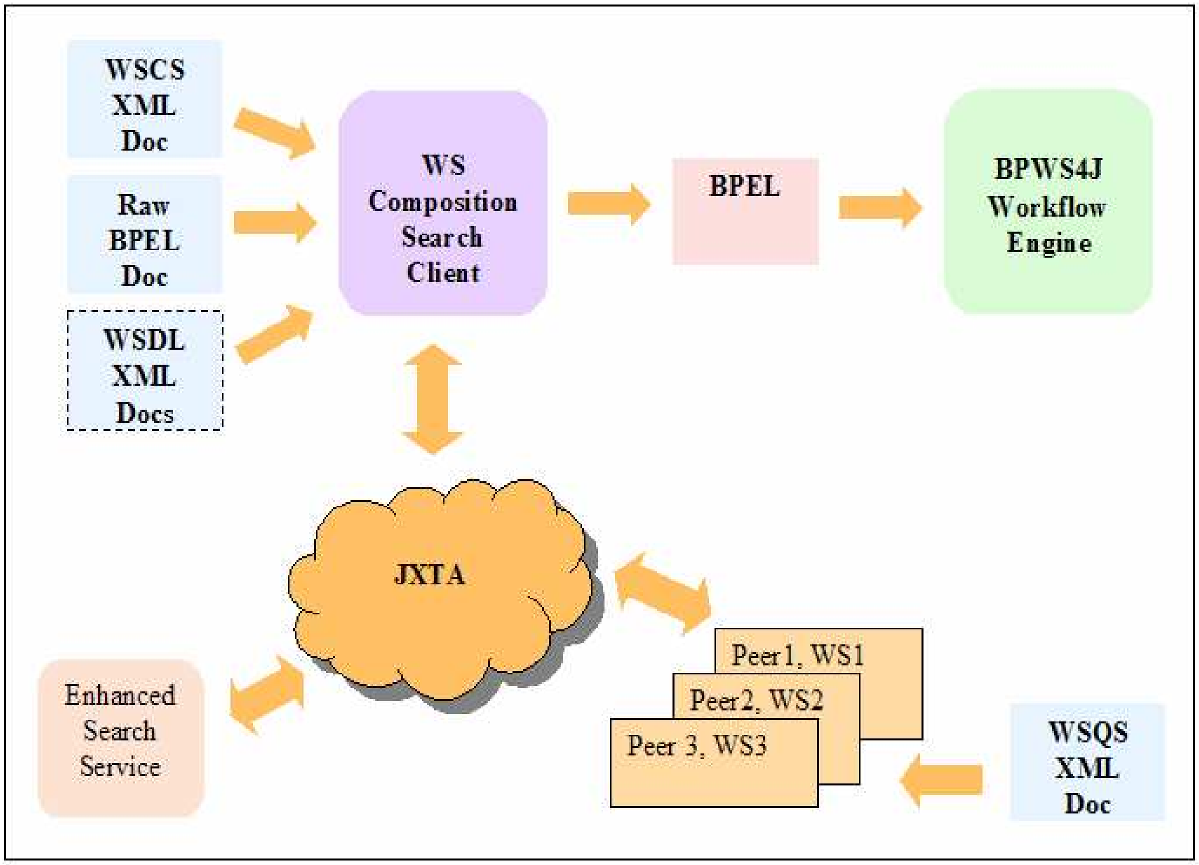


Figure 14 Logical View Of Web Service Composition QoS Enabler Framework

4.3.1 WSQEF Components

The framework execution flow involves the following components:

4.3.1.1 XML Input Documents

Raw BPEL: This document describes the interfaces of the business process that needs to be executed in order to fulfill the client's requirement. It will have all the information that the document requires for execution except the port type of the web services used in this business process. The port type is eventually provided by the framework based on the criteria specified by client in the next document.

WSCS: This is also an XML document called Web Service Criteria Search; this XML document conforms to a schema that is described in detail in section 5.1 and is one of the contributions of the thesis. The client specifies a list of web services the desired web service and the required QoS for that web service.

WSDL: WSDL documents are standardized web service documents that are published by their providers. The BPEL4J engine requires the WSDL documents of each web service referred to in the BPEL document in order to execute the BPEL document. The location of the WSDL documents are provided by the Web Service Providers (WS1, WS2, WS3, …) as part of their advertisement in the JXTA network. This way they are not required to be in a central directory.

WSQS: This is also an XML document called Web Service QoS (WSQS); this XML document conforms to a schema that is described in detail in section 5.1 and is one of the contributions of the thesis. The service provider specifies functional information of a Web Service and a list of QoS attributes of the Web Service that it guarantees.

4.3.1.2 **Web Service Composition Search (WSCS) Client**

The WSCS client connects to the JXTA network first, reads the search criteria XML document and sends the query to the JXTA enhanced search service. The WSCS client uses a JXTA bi-directional pipe to receive the result from the search service. It deciphers the received message that contains the port types for the specified web services with the defined criteria in the XML document. It then updates the Raw BPEL file and create an executable BPEL document. The implementation details of the component are described in section 5.4.

4.3.1.3 **Enhanced Search Service**

The enhanced search service is a JXTA service that receives the query XML requests in the form of the XML document and will search the JXTA network for the advertisements matching the requested criteria. It will then create an XML document that contains the specified service with the correct port type and sends the result back to the client pipe listener. The implementation details of this component are described in section 5.3.

4.3.1.4 **Peers**

Peers shown in

Figure 14 are the peers that are hosted by service providers and have the web service information and the QoS defined for that web service. The peers advertise this information using the advertisement mechanism provided by JXTA. The search service will find the matching criteria by reading these advertisements and comparing the criteria with the advertised message. The details of JXTA advertisements are described in section 2.5.3.5.

4.3.2 WSQEF Execution Flow

The following four subsections provide the flow of execution in WSQEF:

4.3.2.1 Web Service Providers Execution Sequence

Figure 15 illustrates the service provider flow of execution; the following are the steps for advertising the service and its QoS attributes in a JXTA network.

1- The web service provider will start the web service server by providing the data about the service that is offering. The information is the functional web service information that uniquely specifies that web service in the network along with a set of QoS attributes that it is capable to fulfill.

2- The web service server joins a peer group in the JXTA P2P network. Services join peer groups based on being similar in functionalities they provide. In JXTA platform peer groups can also communicate with each other through gateway peers.

3- The web service server will populate the provided data in an advertisement document and publishes that advertisement in the network.

4- This will be a repetitive process for all the web service providers.

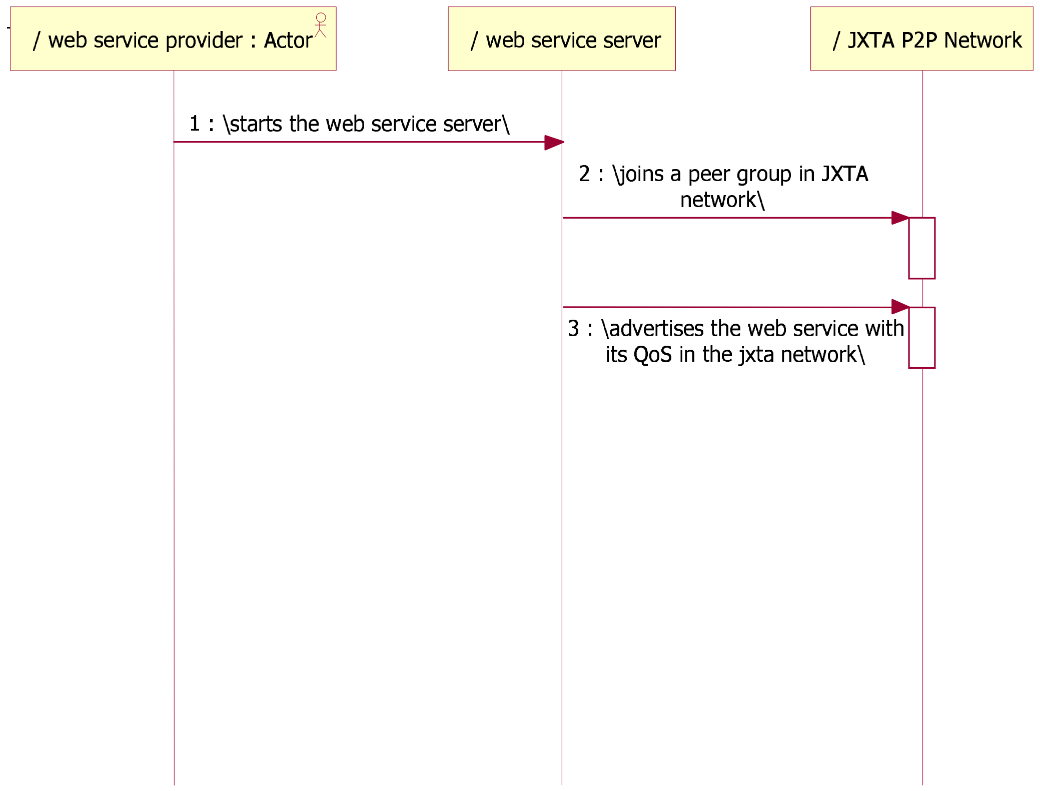

/ web service provider : Actor
/ web service server
/ JXTA P2P Network
1 : \starts the web service server\
2 : \joins a peer group in JXTA network\
3 : \advertises the web service with its QoS in the jxta network\

Figure 15: Web Service Provider Execution Sequence

4.3.2.2 Enhanced Search Service Startup Execution Sequence

Figure 16 illustrates the service provider flow of execution; the following are the steps for starting the enhanced search service in the JXTA network.

1- The Operator executes the Enhanced Search Engine Server.

2- The server joins a peer group in the JXTA P2P network.

3- The server creates a bi-directional pipe to send and receive messages.

4- The server creates an advertisement document for its services and sends it to the JXTA network, this way the clients can find the search engine when they know the name of this service in advance and they can start sending requests to it.

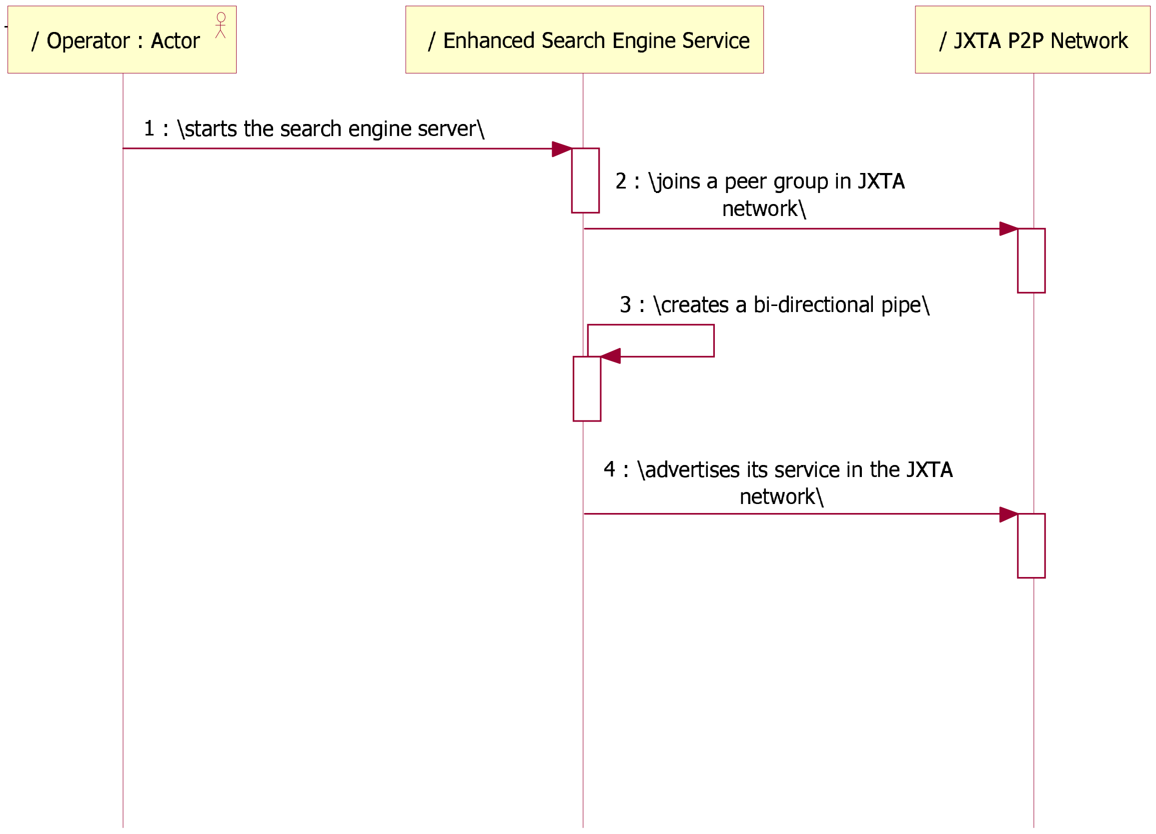

/ Operator : Actor
/ Enhanced Search Engine Service
/ JXTA P2P Network
1 : \starts the search engine server\
2 : \joins a peer group in JXTA network\
3 : \creates a bi-directional pipe\
4 : \advertises its service in the JXTA network\

Figure 16: Enhanced Search Service Setup Execution Sequence

4.3.2.3 Enhanced Search Engine Execution Sequence

Figure 17 illustrates the execution flow of the enhanced search engine. The following describes the steps for the search engine server when it receives a search request:

1- A search request is sent to the search server from the JXTA network.

2- The message will be sent to a message processor to decipher it and extract the information that is searched for.

3- The server then requests the search engine perform the search with the defined criteria.
4- The search engine will query the JXTA network and collects all the services advertised.
5- The search engine compares the results with the criteria and finds the services that match the criteria and returns the matching information.
6- The server sends this information to the message processor to construct the response message.
7- The response message is sent back to the requester in the JXTA network.

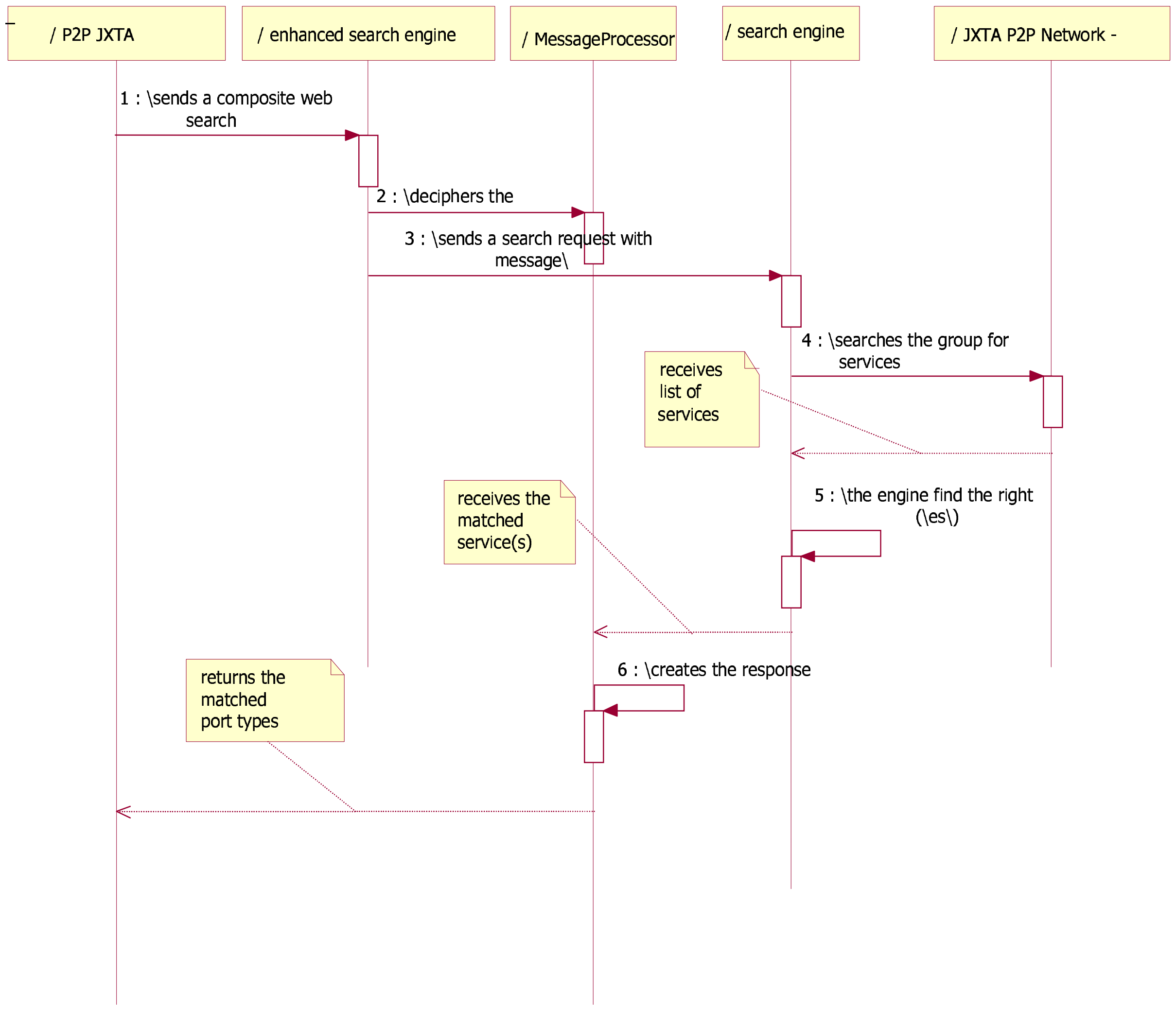

/ P2P JXTA
/ enhanced search engine
/ MessageProcessor
/ search engine
/ JXTA P2P Network -
1 : \sends a composite web search
2 : \deciphers the
3 : \sends a search request with message\
4 : \searches the group for services
receives list of services
5 : \the engine find the right (\es\)
receives the matched service(s)
6 : \creates the response
returns the matched port types

Figure 17: Enhanced Search Service Execution Sequence Diagram

4.3.2.4 Web Service Composition Search (WSCS) Client Execution Sequence

Figure 18 illustrates the execution flow of the client of the composition web service. The following describes the steps for the client to send a request to the P2P network to find a set of web services with the criteria defined by the client:

1- The user starts the WSC search client and provides an XML document that describes a set of the web services functional and QoS criteria.
2- The client joins a peer group in JXTA network.
3- The client creates a bi-directional pipe to send and receive messages.
4- The client parses the search criteria provided by user.
5- It searches for the Enhanced Search Service in the JXTA network.
6- Sends the search request to the search service that was found.
7- BPEL parser receives the result message.
8- BPEL parser updates the RAW BPEL document to create an executable BPEL document.

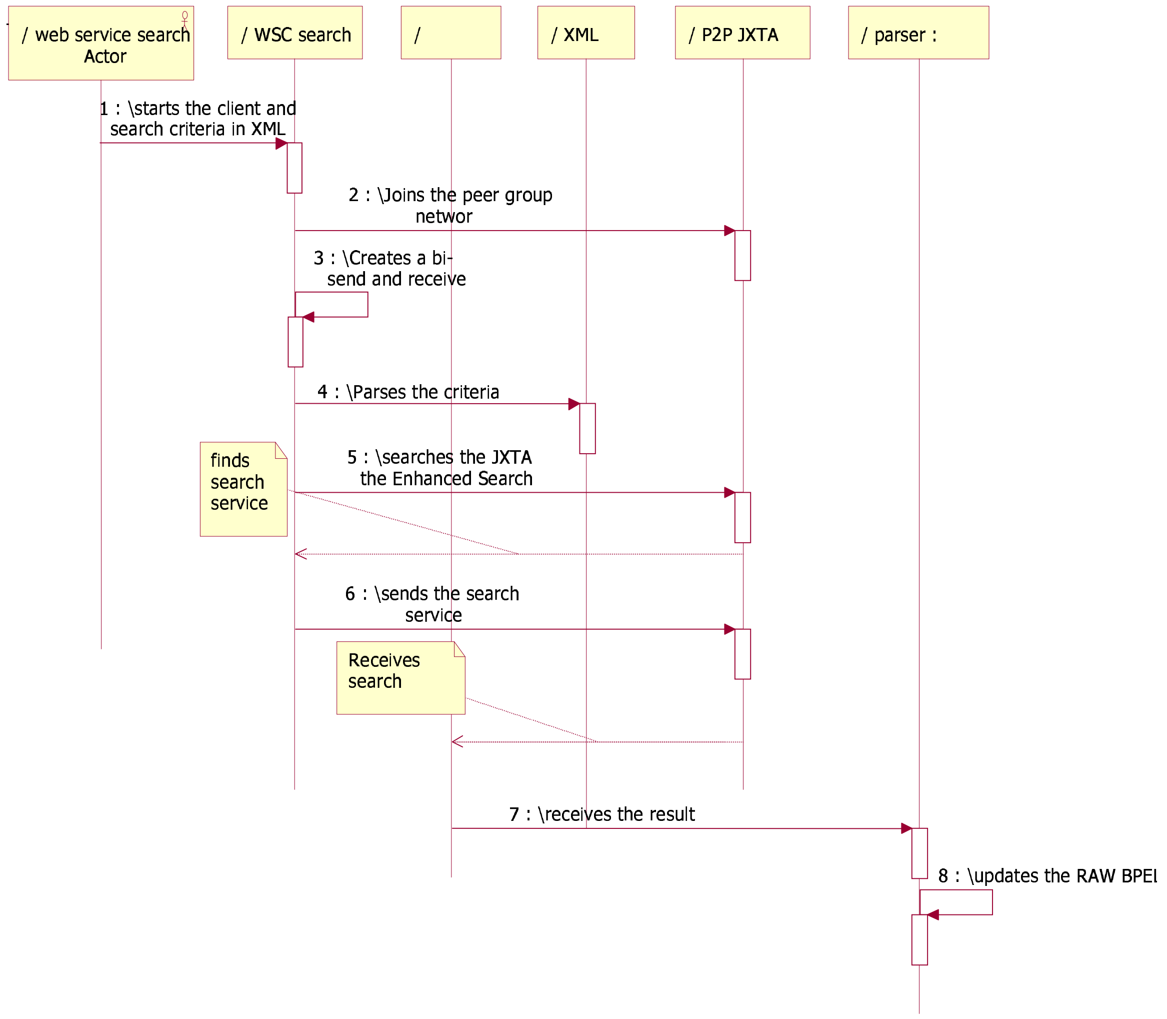

/ web service search Actor
/ WSC search
/
/ XML
/ P2P JXTA
/ parser :
1 : \starts the client and search criteria in XML
2 : \Joins the peer group networ
3 : \Creates a bi-send and receive
4 : \Parses the criteria
finds search service
5 : \searches the JXTA the Enhanced Search
6 : \sends the search service
Receives search
7 : \receives the result
8 : \updates the RAW BPEl

Figure 18: WSCS Client Execution Sequence Diagram

4.4 Summary

The chapter starts by providing the requirements and requirement analysis. An abstract view of the required system and a UML Use case diagram is presented. A framework is proposed which integrates existing technologies with new components to dynamically compose Web Services based on the functional and QoS criteria of the service components. An overview of the existing technologies that are deployed in the framework is presented; they are JXTA, BPEL and BPWS4J. An architectural view of the proposed framework is illustrated and the new components that are the main contributions in this thesis are described. These components include the XML documents proposed to facilitate definitions of search criteria for clients and service providers. The main architectural components in the system are an enhanced search service as a service entity in the JXTA platform to perform queries, match criteria and returns the selected services portType information and a Web Service composite search client program that sends a request to the search service in order to select the right services dynamically based on the client's functional and QoS criteria. The result of the client execution is a set of WSDL and BPEL documents required by BWPSJ, a composite engine that executes a Web Service composition description in the form of BPEL document. Chapter 5 provides more detailed design of the entire infrastructure proposed in this chapter. The schema definition for XML documents are outlined first, followed by detailed design of other components.

5 Implementation

In this chapter implementation details of the WSQEF framework are provided. A diagram is presented that shows an overview of the implementation details followed by subsections describing the components in the diagram. The main class diagrams and sequence diagrams of the implementation are presented in this section as well. Figure 19 presents an implementation view of the WSQEF with regards to JXTA architecture:

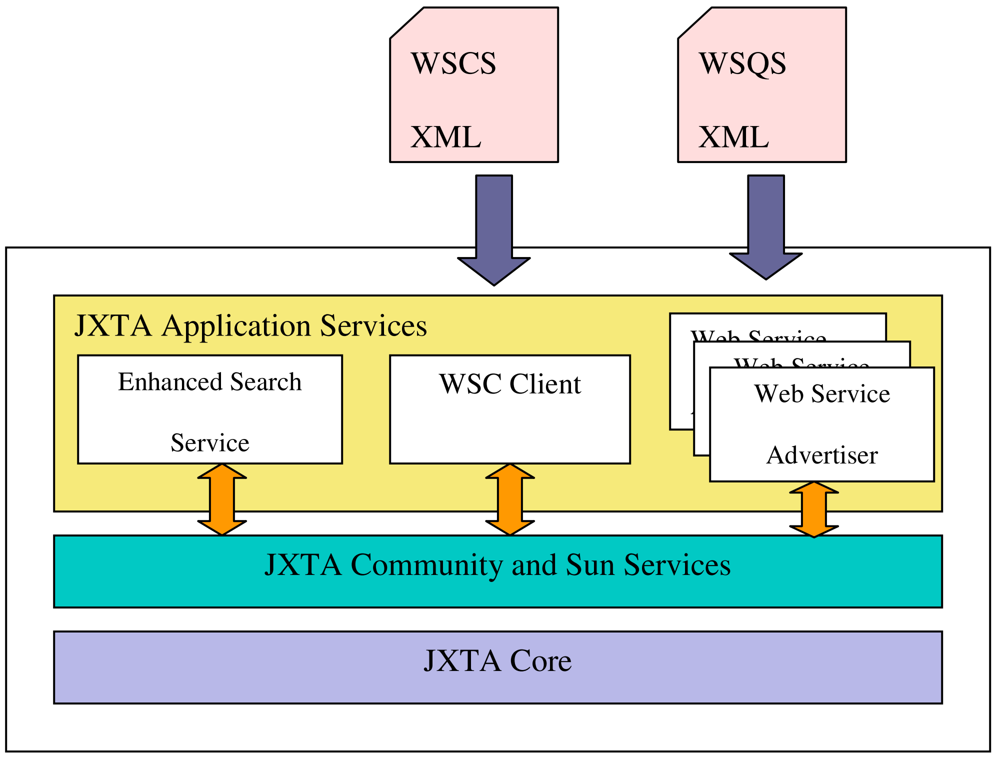


Figure 19: WSQEF Implementation View

The main components of WSQEF are:

- WSCS (Web Service Composite Search) and WSQS (Web Service QoS) XML documents. These two documents are based on Composite Web Service Search Criteria schema, the complete design of the schema is provided in section 5.1.
- The Web Service QoS Advertisers, they will be executed as peers in the JXTA network and advertise the WSQS document along with the WSDL location of the service they are advertising. They use "parm" element of the module specification advertisement that is described in section 2.5.3.5.4 to advertise their WSQS information. An example of WSQS is provided in Table 19. The design details of this component are provided in section 5.2.
- The Enhanced Search Service, the search service will be executed as a peer in the JXTA network, the design details of this component are provided in section 5.3.
- The Web Service Composition Client, the client will also join the JXTA network as a peer, the design details of this component are provided in section 5.4.

These components are part of the JXTA application layer, and use the services provided in the JXTA community layer (discovery and advertisement) and JXTA Core layer (peer, peer groups and pipes). JXTA Architecture layers are described in section 2.5.2.

5.1 Composite Web Service Search Criteria XML Schema

This section describes the XML schema for searching composite web services based on the specified QoS criteria in the JXTA environment.

The XML schema enables the clients to create their composite web service search in an XML document that conforms to this schema. The schema is designed in a way that each Web Service is defined with functional and QoS criteria. The functional definition is for finding the Web Services that provide similar functionality. The QoS criteria definition is designed in a flexible way by using name/value pair definition. The schema is broken down into 3 main sections as follows, in each section a description of the components, their types and tables representing schema definition and examples are provided.

5.1.1 Web Service composition

Figure 20 illustrates the main components of the schema definition.

It consists of the root element and a list of web service search elements, each web service search contains a sequence of functional criteria and a list of QoS criteria that can be logically combined based on AND/OR operators. The web service search elements enable the user to define a list of web services that are part of their composite web service process and meet the QoS criteria they require. Each web service element has functional and QoS criteria. The following subsections describe these criteria definitions in detail.

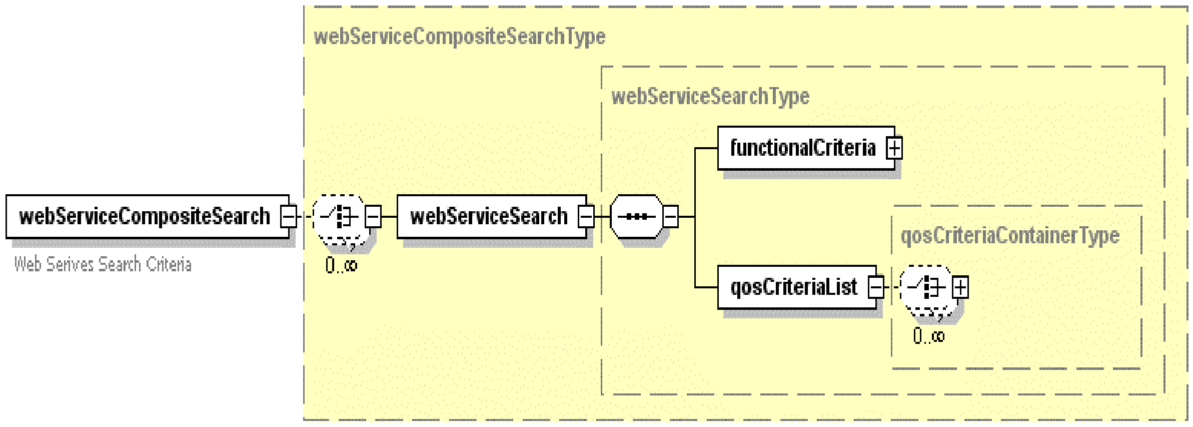


Figure 20: Main Component of Web Service Composite Search Schema Definition

5.1.2 Functional Criteria

Figure 21 presents the functional criteria component of a web service search; functional criteria will uniquely define a web service in the network. It can be either the list of operations of a web service or the URL of a web service.

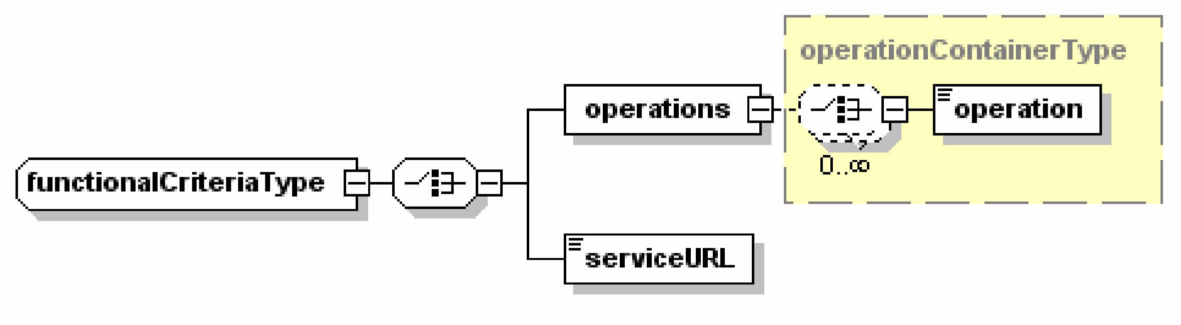


Figure 21: Function Criteria Definition View

5.1.3 QoS criteria

The QoS criteria component enables the clients to define a list of constraints for each web service that they are searching for.

Figure 22 provides a general view of each QoS Criteria within the list of criteria, each QOS criteria search consists of QOS name, QOS condition and a criteria logical operator that determines the AND or OR logical combination with the next criteria. Brackets are not supported.

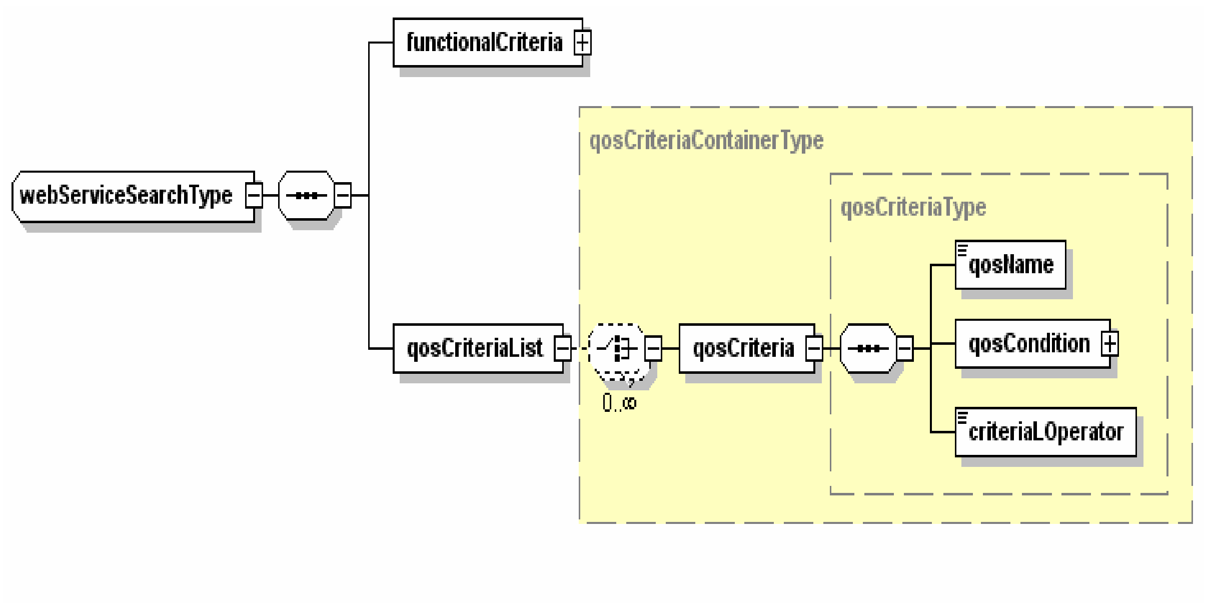


Figure 22: QoS Criteria Definition View

Figure 23 provides a general view of the qosCondition component that defines the condition that a web service search is required to meet. The conditions can be based on number, string, date and time comparisons. In each condition, there is a value definition along with the comparison definition that is suitable for that value type.

The details of the schema definitions are provided in a separate chapter 7 in order to more easily follow the rest of the implementation details.

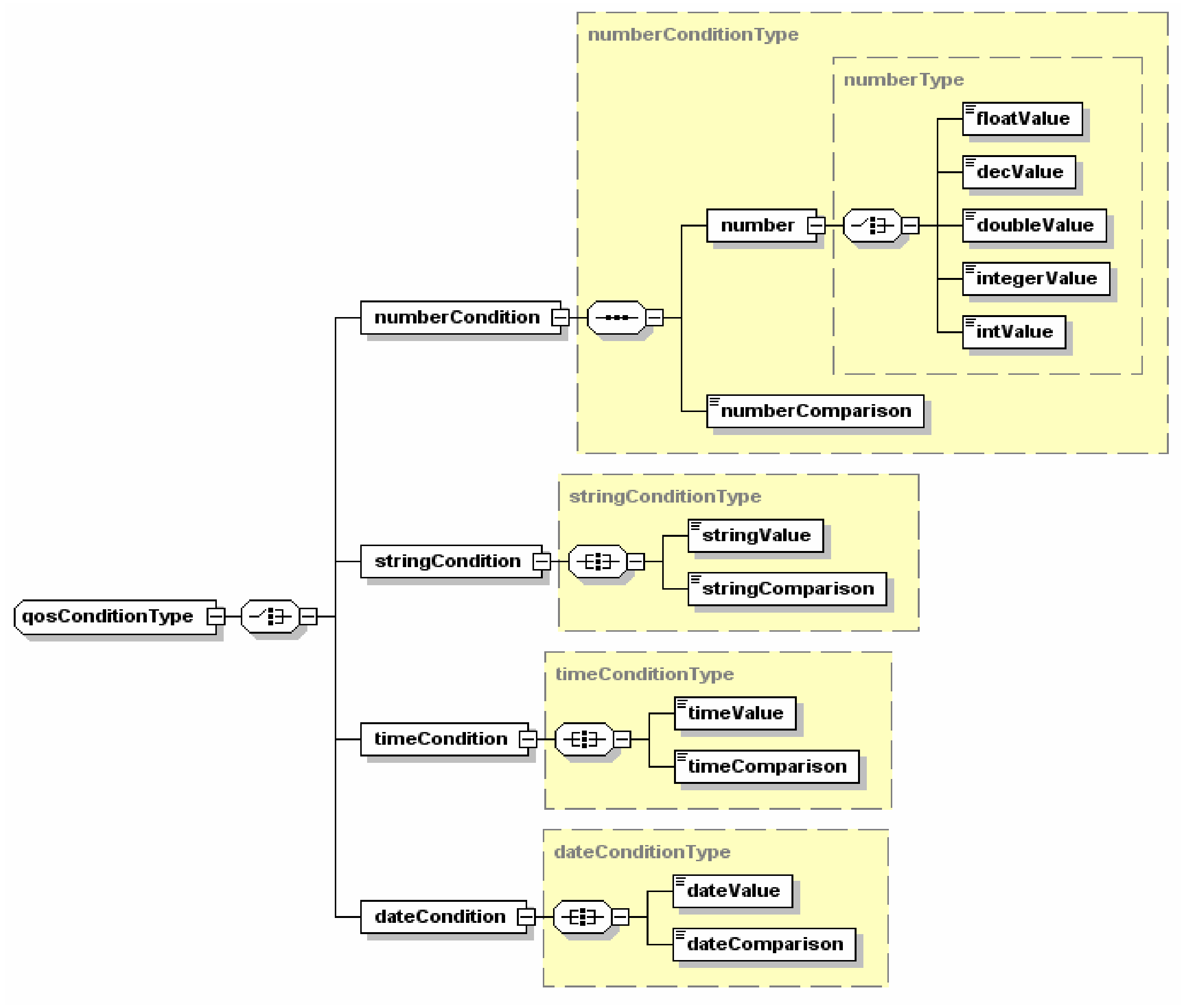


Figure 23: QoS Condition Definition View

5.2 Web Service Advertiser

Figure 24 illustrates the class diagram for the Web Service Advertiser followed by description of its main classes.

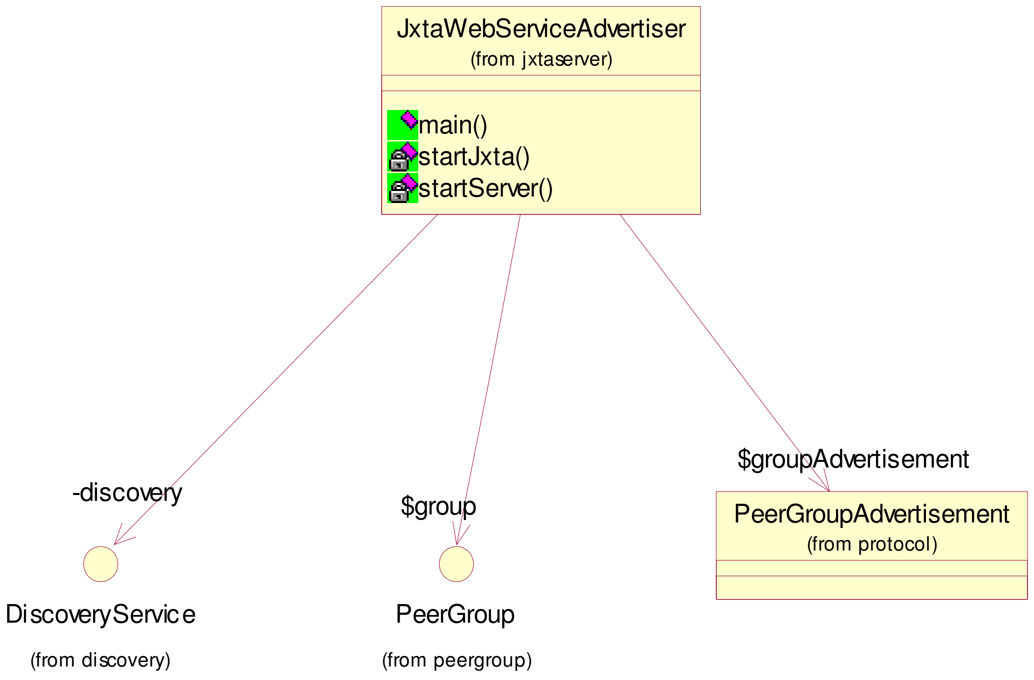


Figure 24: Class Diagram of Web Service Advertiser

JxtaWebServiceAdvertiser: This class starts a server that advertises the Web Service information in the JXTA community. It reads the functional and QoS information of the Web Service from the XML document provided by Web Service Provider.

This class is responsible for joining the specified peer group, creating the Peer Group advertisement document and publishing the advertisement to the JXTA peer group by using the JXTA discovery service.

5.3 Enhanced Search Service

Figure 25 illustrates the class diagram for the Enhanced Search Service component of the WSQEF framework followed by descriptions of its main classes.

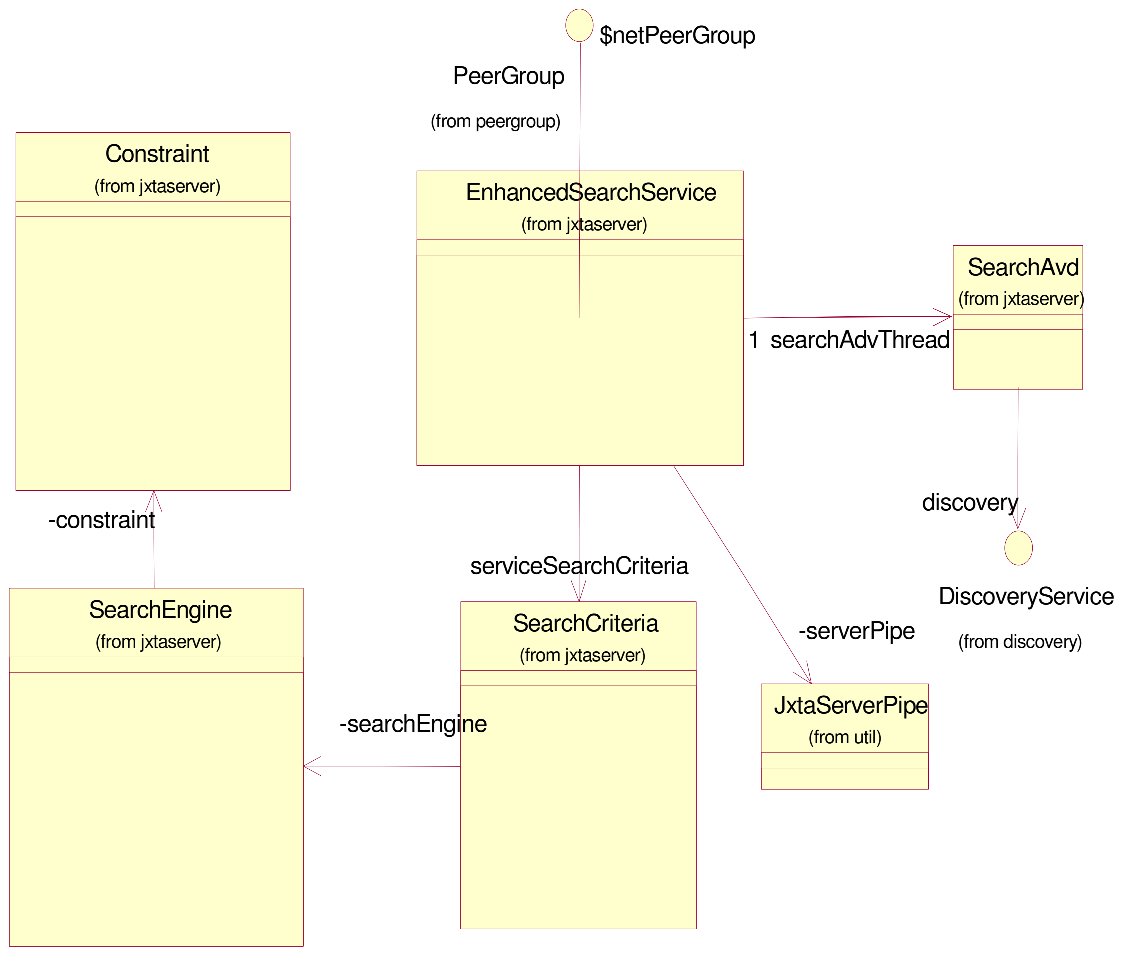


Figure 25: Enhanced Search Service Class Diagram

EnhancedSearchService: This class is responsible for the following:

- Starting the server
- Joining the peer group in the JXTA network
- Listening for new requests for web service searches.
- Processing the incoming requests and sending the request to other classes.
- Wrapping the query result in a structured document and sending it back to the requester.

SearchAdv: This class starts a thread that constantly retrieves the advertisements that are available in the peer group.

JxtaServerPipe: This is a JXTA class that provides the facility to send and receive messages through pipes. The search requests and responses are communicated through a JxtaServerPipe.

SearchCriteria: This class holds the functional and QoS criteria of one of the Web Services in the Web Service Composition. It is responsible for querying the JXTA network for services and if the functional information was matched with the advertised services, it will ask the SearchEngine to verify the QoS information. The QoS criteria contain a list of constraints that the requester needs to fulfill.

This class also wraps all the values found for each Web Service in a JxtaSearchResult object and returns it to the server.

JxtaSearchResult: This class sends a query to the JXTA network through discovery services and finds the services that match the functional information of a web service.

SearchEngine: The search engine is responsible for verifying that the constraints of a request match the advertised QoS for a specific service.

It will perform a constraint-based comparison depending on the type of the QoS, the value and the condition that needs to be met. This class basically implements the WSCS XML Schema that was described in section 5.1. It will support the types and comparisons that are defined in the schema:

The search engine extracts the Web Service functional and QoS information for each service found, and compares it to the request criteria. The comparison is performed based on the type defined in the criteria: number (float, integer, decimal and double), date, time or string. For each type a comparison mechanism is provided in the engine.

The comparison will be performed for each of the Web Services in the Web Service composition and the result is composed based on the logical operator that the user has provided in the criteria.

Constraint: This class keeps a QoS name, value, type and the condition that it requires to meet.

5.4 Web Service Composition Search Client

Figure 26 illustrates the class diagram for the Web Service Composition followed by description of its main classes.

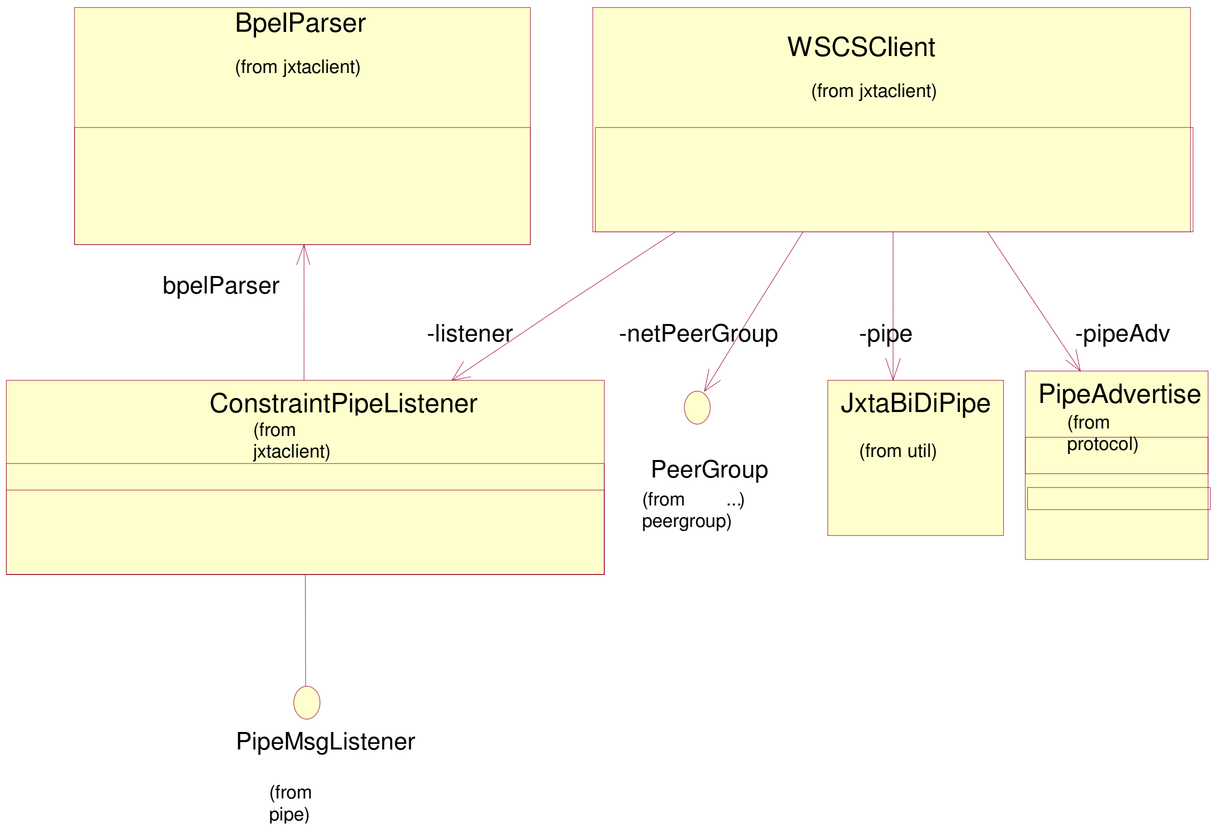


Figure 26: Web Service Composition Client Class Diagram

WCSCClient: This is the main class that is responsible for reading the XML search criteria document that the user provides, it will join the JXTA peer group, creates a bi-directional pipe and publishes a pipe advertisement.

ConstraintPipeListener: This class is instantiated by a WCSCClient and listens for messages that are sent to the client from the peers within the joined peer group in a JXTA network.

BpelParser: This class parses the result of search that is received by the ConstraintPipeListener and updates the BPEL document with the found PortType for each service.

5.5 Summary

Chapter 5 provides implementation details of the thesis contributions. It starts by providing the design details of the Composite Web Service Search Criteria XML Schema. Main elements of the schema are described and model views for each of them are presented. Web Service Advertiser (section 5.2), Enhanced Search Service (section 5.3) and Web Service Composition Client (section 5.4) are described from implementation point of view, providing class diagrams and class descriptions. The next chapter provides a description of a prototype written based on the implementation.

6 Assessment

This chapter presents an experiment of the concepts and designs that were described in previous sections. It is a prototype implementation of the WSQEF framework. JAVA, XML and JXTA and BPEL technologies are used throughout the implementation. In this section we walk through an example of a Web Service search composition in JXTA in WSQEF framework. The chapter describes an example scenario in section 6.1 and provides the necessary steps that take place for the scenario to execute a Web Service composition based on WSQEF architecture.

6.1 Example Scenario

In this example, the process of a loan request is implemented. The process begins with a customer requesting for a loan. The business process then sends the request to a financial institution and receives the result. Then it will send the request to an assessment company and asks for the risk associated with the loan.

If we were going to implement this business process without using WSQEF framework, we would have to choose a fixed financial institution and a fixed assessment company in the business process that define a Web Service composition scenario. In contrast, in the WSQEF framework, a number of financial institutions and assessment companies can take part in the business process provided that they agree on a set of interfaces and the same service names. The framework decides at run time, which one of the service providers qualifies for taking part in business process execution. PortTypes determine what service to bind at execution time. The framework assigns them values that are selected Web Services based on the customer's criteria. The following subsections describe the steps that fulfill this business process execution.

6.2 Raw BPEL Document

A set of WSDL documents and one BPEL document are required to be written for this business process. At design time we assume that the WSDL documents of the financial institutions that provide loan services through a Web Service are available and they all support the same operations and messages, see Table 24 for the WSDL document of this Web Service example. The same applies to assessment companies; they all have the same interface for providing assessments on loans, see Table 25 for the WSDL document of this Web Service example. A WSDL document that defines the process interfaces is called the raw BPEL document. In this document the portTypes define the service bindings at run time. In WSQEF the portTypes are placeholders that will be filled during the business process execution.

A BPEL is required to define the process of this Web Service composition scenario, see Table 26 for this BPEL document. In the Raw BPEL WSDL document (Table 16) these elements are highlighted, the value of portType elements are the Web Service names at this point. A Web Service name is a name contracted between the consumers and service providers, each set of Web Services that provide similar services will agree on the same name.

```
<definitions
            targetNamespace="http://loans.org/wsdl/loan-approval"
            xmlns="http://schemas.xmlsoap.org/wsdl/"
            xmlns:slnk="http://schemas.xmlsoap.org/ws/2003/03/service-link/"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:lns="http://loans.org/wsdl/loan-approval"
            xmlns:asns="http://tempuri.org/services/loanassessor"
            xmlns:apns="http://tempuri.org/services/loanapprover"
            xmlns:loandef="http://tempuri.org/services/loandefinitions">

    <import  namespace="http://tempuri.org/services/loanassessor"
                    location="http://localhost:9080/bpws4j-
samples/loanapproval/loanassessment.wsdl"/>
    <import  namespace="http://tempuri.org/services/loanapprover"
                    location="http://localhost:9080/bpws4j-samples/loanapproval/loanapprover.wsdl"/>
    <import  namespace="http://tempuri.org/services/loandefinitions"
                    location="http://localhost:9080/bpws4j-samples/loanapproval/loandefinitions.wsdl"/>

    <slnk:serviceLinkType  name="loanApprovalLinkType">
        <slnk:role  name="approver">
            <portType  name="loan  service"/>
        </slnk:role>
    </slnk:serviceLinkType>

    <slnk:serviceLinkType  name="riskAssessmentLinkType">
        <slnk:role  name="assessor">
            <portType  name="risk  assessment service"/>
        </slnk:role>
    </slnk:serviceLinkType>
    <!-- The service name and the TNS represent my service ID QName -->
    <service  name="loanapprovalServiceBP"/>
```

Table 16 Process WSDL document

6.3 WSQS XML

The WSQS document contains the functional and QoS information of the particular web service that the provider intends to advertise. This document conforms to the schema that was proposed as part of WSQEF and described in detail in section 5.1. Each provider determines the values of functional and QoS information it will advertise. Table 17 presents an example of the QoS criteria for a loan service. Table 19 XML document is an example of a WSQS for a loan service.

Service name: loan service	
Name	**Value**
delivery	http
ServiceCharge	4
portType	apns:loanApprovalPT

Table 17 Service name: loan service

Table 19 presents an example of the QoS criteria for a loan assessment service.

Service name: loan assessment service	
Name	**Value**
security	encrypted
AdminFee	150
portType	apns:loanAssessApprovalPT

Table 18 Service name: loan assessment service

```xml
<?xml version="1.0" encoding="UTF-8"?>
<webServiceCompositeSearch xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="C:\thesis\ServiceSearch.xsd">
<webServiceSearch>
        <functionalCriteria>
                <serviceName>loan service</serviceName>
        </functionalCriteria>
        <qosCriteriaList>
                <qosCriteria>
                        <qosName>security</qosName>
                        <qosCondition>
                                <stringCondition>
                                        <stringComparison>Equals</stringComparison>
                                        <stringValue>encrypted</stringValue>
                                </stringCondition>
                        </qosCondition>
                        <criteriaLOperator>And</criteriaLOperator>
                </qosCriteria>
                <qosCriteria>
                        <qosName>serviceCharge</qosName>
                        <qosCondition>
                                <numberCondition>
                                        <number>
                                                <integerValue>4</integerValue>
                                        </number>
                                <numberComparison>Equals</numberComparison>
                                </numberCondition>
                        </qosCondition>
                        <criteriaLOperator>And</criteriaLOperator>
                </qosCriteria>
        </qosCriteriaList>
</webServiceSearch>
</webServiceCompositeSearch>
```

Table 19 Web Service QoS (WSQS) Document

6.4 Web Service Advertisers

Web Service advertisers are peers that host Web Services in JXTA network. They will join the same peer group and are part of a closed community within the P2P environment. Peer services that are not part of the same peer group can also be discovered through a gateway peer. Figure 27 illustrates the sequence diagram of a Web Service advertiser:

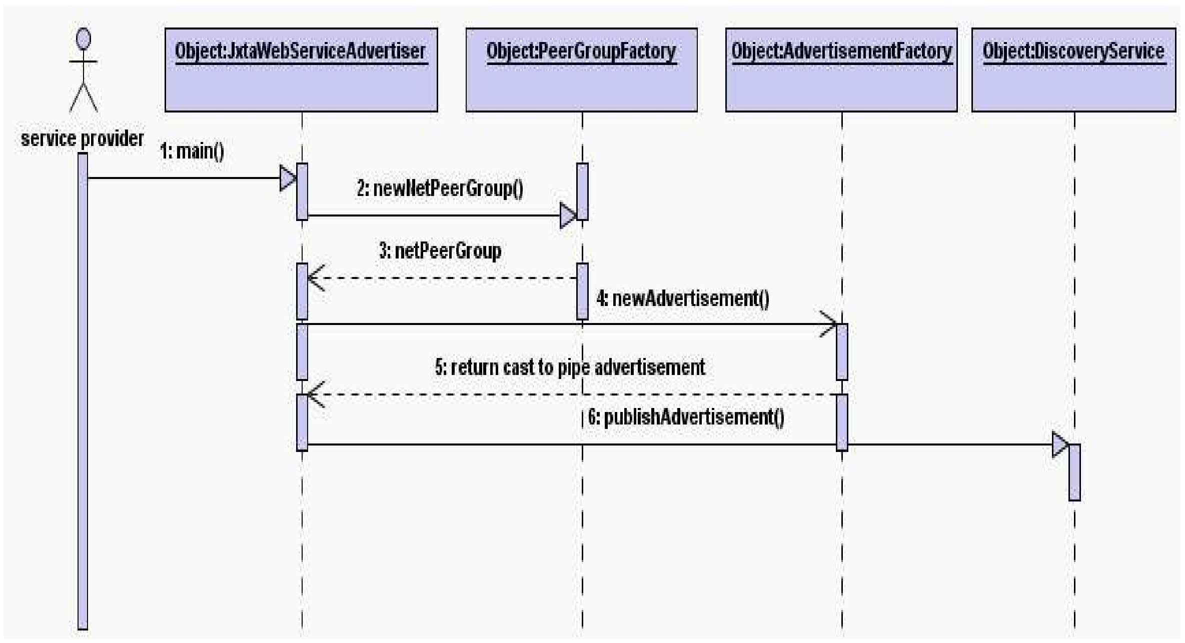


Figure 27 Sequence Diagram of Web Service Advertiser

The advertiser peer reads the WSQS XML document at start up and then joins a peer group. There are two peer groups that already exist, one is the Net peer group and the other is the world peer group, we used the Net for the example. In a real world example service providers that belong to the similar communities create their own group. JXTA provides an infrastructure to retrieve services from other groups through a gateway peer. The peer then puts the QoS information along with the web service name in a pipe advertisement and publishes the advertisement in the peer group it belongs to by using the publish method of the discovery service. Web Service providers can use the Advertiser program to join the group and publish QoS information for the services they provide.

To be able to advertise the information read from the WSQS, the Pipe Module Class and Model Spec advertisements have been used. Module Spec advertisement (described in section 2.5.3.5.4) has a “*parm*” element that can be extended and can hold an XML document structure. The QoS information will be assigned as name/value pairs in this element. This will provide a generic and flexible way to define any QoS name. Table 20 is the advertisement document that will be published in JXTA network for this example. We can setup a number of peers as Web Service advertisers from different financial and assessment institutions. All JXTA advertisements are XML messages.

```
<?xml  version="1.0"  encoding="UTF-8"?>
<!DOCTYPE  jxta:MSA>
<jxta:MSA  xmlns:jxta="http://jxta.org">
        <MSID>
        urn:jxta:uuid-
8725727DFEF34DA7A6C0D5D7B93B1F2B14A1C2AAB7E64FC1A149A84CCA15BCCB06
        </MSID>
        <Name>JXTASPEC:JXTA-WEBSERVICE-LOAN</Name>
        <Crtr>sun.com</Crtr>
        <SURI>http://www.jxta.org/WebService</SURI>
        <Vers>Version  1.0</Vers>
        <Parm>
                <service>loan  service</service>
                <delivery>http</delivery>
                <ServiceCharge>4</ServiceCharge>
                <portType>apns:loanApprovalPT</portType>
        </Parm>
  </jxta:MSA>
```

Table 20 Web Advertiser's Advertisement Document

6.5 Enhanced Search Service

This component is another peer service that provides composite search services. Upon start up, it joins the peer group and listens for composite search requests by creating a JXTA pipe server facility. Figure 28 illustrates the sequence diagram of an enhanced search service when it receives a request:

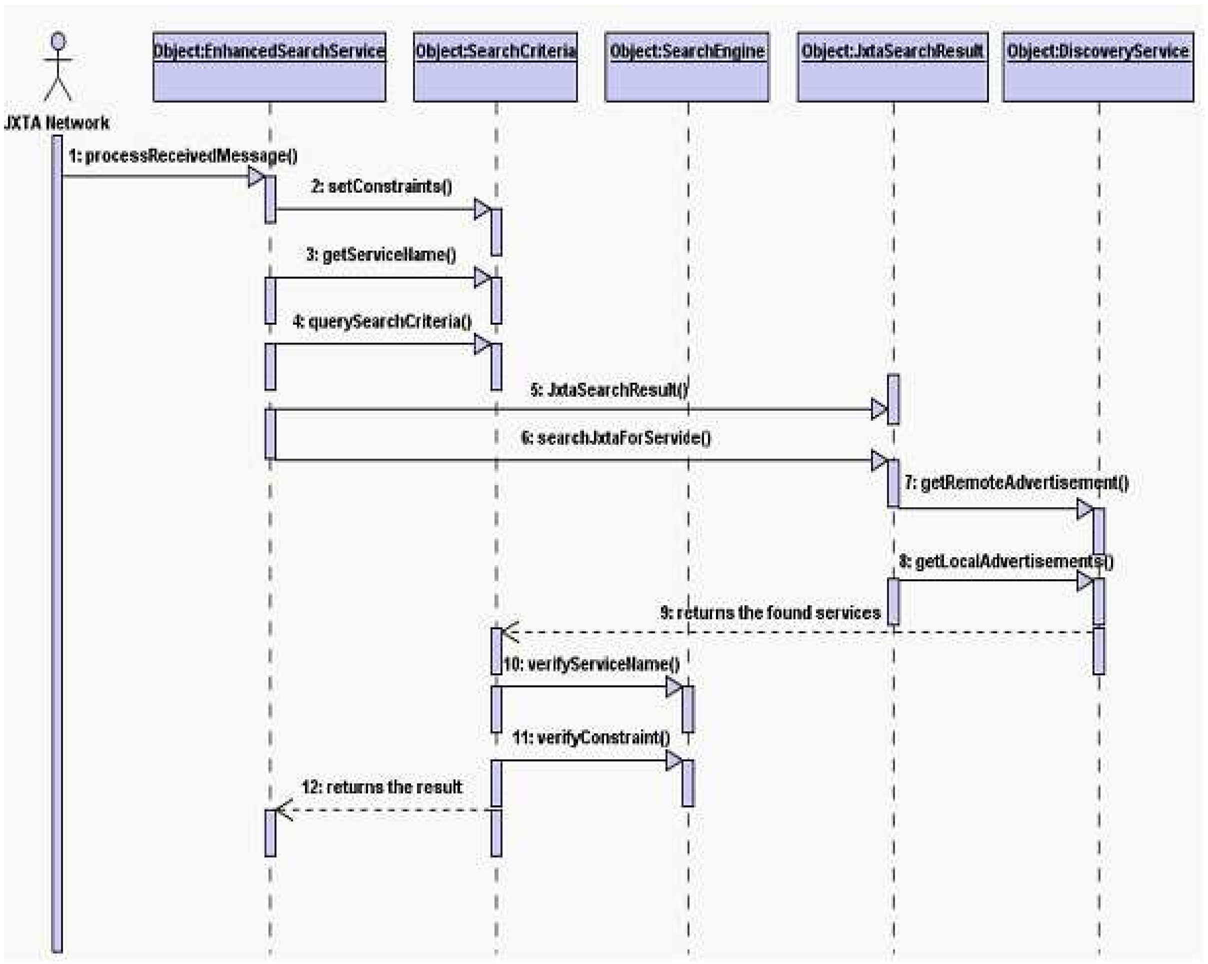


Figure 28 Sequence Diagram of Receiving a Request by Enhanced Search Service

On arrival of a request (message 1 in Figure 28), the message processor parses the message and extracts the search request information. Then it gets all the advertisements that were advertised by Web Service providers. By contract, the name element of those advertisements are all "JXTASPEC:JXTA-WEBSERVICE-LOAN" in this experiment. The search engine matches the service name first, if the search was successful, then a comprehensive comparison takes place for matching the QoS constraints. For example in the loan approval scenario, we assume the search request criteria is only looking for a loan service that has less than $200 administration fee, the XML snippet of this criteria is (the complete WSCS is provided in Table 27):

```
<qosCriteria>
        <qosName>AdminFee</qosName>
        <qosCondition>
                <numberCondition>
                        <number>
                                <integerValue>200</integerValue>
                        </number>
                        <numberComparison>LessThan</numberComparison>
                </numberCondition>
        </qosCondition>
        <criteriaLOperator>And</criteriaLOperator>
</qosCriteria>
```

The search engine has mechanisms to perform a constraint match based on the type of the constraint, the comparison element and the value. Based on the XML schema definition for WSCS, the QoS can be a number, string, date or time. See Figure 23 for a complete listing of QoS types. The engine supports each type that is defined in the WSCS schema. In the above example, first the search verifies the QoS property name ("*AdminFee*"), if found, it checks for the type which is an integer number, then it will perform the comparison, configured as "*LessThan*", so it will look for the services with the same name that offer administration fee of less than $200. If more than one search applies, it will choose the first one and if none exists, the composite search will return nothing and the process will be stopped.

This will be repeated for the entire search service request; the results are composed and put together in an XML document that will be sent back to the customer through the bi-directional pipe. The result includes the name of the service and the portType, which is the binding information for that service. Table 21 is the example of the XML message that it returned back to the client:

```
Message : <?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE services>
<services>
        <service>
                <name>loan service</name>
                <portType>apns:loanApprovalPT</portType>
        </service>
        <service>
                <name>loan assessment service</name>
                <portType>asns:loanInsPT</portType>
        </service>
</services>
```

Table 21: Returned XML Message from Search Service

6.6 Web Service composition Client

A Web Service composition client is a program that clients execute to send a composite search request to the JXTA network, the request is configured by clients in an XML document called Web Service Criteria Search (WSCS), which conforms to the schema described in section 5.1. Figure 29 illustrates a sequence diagram of a search request in the Web Service composition Client:

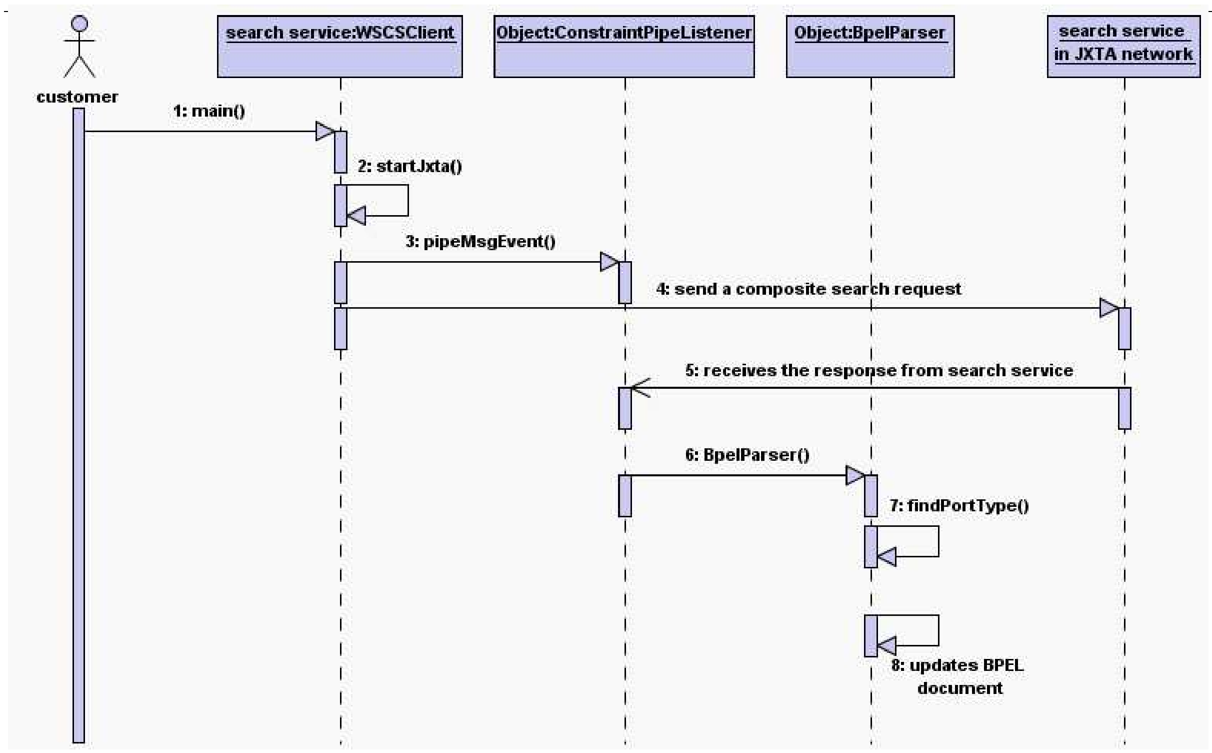

customer
search service:WSCSClient
Object:ConstraintPipeListener
Object:BpelParser
search service in JXTA network
1: main()
2: startJxta()
3: pipeMsgEvent()
4: send a composite search request
5: receives the response from search service
6: BpelParser()
7: findPortType()
8: updates BPEL document

Figure 29 Sequence Diagram of Sending a Request by Composite Search Client

The client program in the experiment connects to the search service through a pipe and sends the request document in XML format. Table 27 is the WSCS XML document that is used in the example. This document is configured in a way that it searches for the following web services with the QoS criteria in Table 22:

Service name: loan service	
Name	**Value**
delivery	http
ServiceCharge	<5
Service name: loan assessment service	
Name	**Value**
security	encrypted
AdminFee	<200

Table 22 Service name: loan service

After receiving the result, it will decipher the result and if a successful match was found, it will extract the portTypes for each service and replaces it with the returned value. The received result is as shown in Table 23. The client generates this WSDL document by using the Raw BPEL document and replacing the portTypes, here is the WSDL process generated document that can be executed by BPWS4J's engine developed by IMB AlphaWorks.

```
<definitions
            targetNamespace="http://loans.org/wsdl/loan-approval"
            xmlns="http://schemas.xmlsoap.org/wsdl/"
            xmlns:slnk="http://schemas.xmlsoap.org/ws/2003/03/service-link/"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:lns="http://loans.org/wsdl/loan-approval"
            xmlns:asns="http://tempuri.org/services/loanassessor"
            xmlns:apns="http://tempuri.org/services/loanapprover"
            xmlns:loandef="http://tempuri.org/services/loandefinitions">



    <import  namespace="http://tempuri.org/services/loanassessor"
                    location="http://localhost:9080/bpws4j-samples/loanapproval/loanassessor.wsdl"/>
    <import  namespace="http://tempuri.org/services/loanapprover"
                    location="http://localhost:9080/bpws4j-samples/loanapproval/loanapprover.wsdl"/>
    <import  namespace="http://tempuri.org/services/loandefinitions"
                    location="http://localhost:9080/bpws4j-samples/loanapproval/loandefinitions.wsdl"/>

    <slnk:serviceLinkType  name="loanApprovalLinkType">
        <slnk:role  name="approver">
            <portType  name="apns:loanApprovalPT"/>
        </slnk:role>
    </slnk:serviceLinkType>

    <slnk:serviceLinkType  name="riskAssessmentLinkType">
        <slnk:role  name="assessor">
            <portType  name="asns:riskAssessmentPT"/>
        </slnk:role>
    </slnk:serviceLinkType>
    <!-- The service name and the TNS represent my service ID QName -->
    <service  name="loanapprovalServiceBP"/>

</definitions>
```

Table 23: The Updated WSDL Process Document

6.7 Summary

This chapter implements a prototype as a proof of concept for the WSQEF framework proposal. The prototype includes the schema design, the extensions to JXTA advertisement to support QoS advertisements, an example of XML documents based on a scenario, the Web Service composite search client and the enhanced search service. The chapter begins with describing a scenario, which is from an example in BPWS4J. The XML documents are developed to define the functional and QoS criteria required by the customer. The sequence diagrams of the components are provided and are followed by messaging that occurs between components at the time of execution. The end result of the document is a WSDL document that includes the portTypes of the services to be called during BPEL execution by BPWS4J composite engine. The next chapter provides the detailed definition of the XML schema for reference purposes. Readers may continue to chapter 8, which outlines the conclusions, contributions of this thesis and several potential future work, without significant loss of thesis comprehension. It is included here as it represents a contribution of the thesis.

7 Composite Web Service Search Criteria Schema Definition

This chapter provides the detailed definition of Composite Web Search Criteria XML schema. Each particle is described by an element definition table, a type definition table is also provided if the particle is a complex type. The element table provides:

- A diagram that presents the content model of that element.
- The type of the element
- Properties of the element such as: simple or complex type, reference
- Annotation for the root element.
- Source, the text view of this element in the schema.

  Each

The type definition table provides:

- A diagram that presents the content model of that type.
- A list of children for the particular element type.
- The list of elements that use this type.
- Source, which is the text view of the type definition.

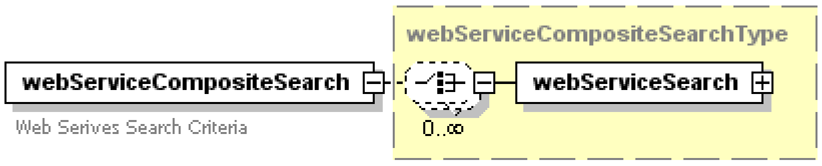
The definition starts by defining the root element:

7.1 WebServiceCompositeSearch

Element: WebServiceCompositeSearch

webServiceCompositeSearch is the root definition for the composite web service search criteria. It consists of a list of web services search that are going to be composed to create a composite web service that meets the search criteria for each of the web service. The following table provides a presentation and schema definition for *webServiceCompositeSearch*.

element webServiceCompositeSearch

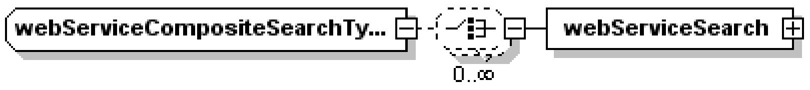
diagram	 webServiceCompositeSearchType webServiceCompositeSearch Web Serives Search Criteria 0..∞ webServiceSearch
type	**webServiceCompositeSearchType**
properties	content complex
children	**webServiceSearch**
annotation	documentation Web Services Search Criteria

source	`<xsd:element name=”webServiceCompositeSearch” type=”webServiceCompositeSearchType”>`
	`<xsd:annotation>`
	`<xsd:documentation>Web Services Search Criteria</xsd:documentation>`
	`</xsd:annotation>`
	`</xsd:element>`

Type: webServiceCompositeSearchType

webServiceCompositeSearch type is a 0 to unbounded choice of webServiceSearch, the following table provides the schema definition for this type.

complexType webServiceCompositeSearchType

diagram	
children	**webServiceSearch**
used by	element **webServiceCompositeSearch**

source	`<xsd:complexType name=”webServiceCompositeSearchType”>`
	`  <xsd:choice minOccurs=”0” maxOccurs=”unbounded”>`
	`    <xsd:element name=”webServiceSearch” type=”webServiceSearchType”/>`
	`  </xsd:choice>`
	`</xsd:complexType>`

7.2 WebServiceSearch

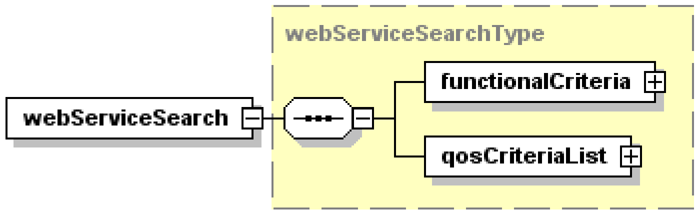
Element: WebServiceSearch

webServiceSearch enables the definition of each element of a web service composition search criteria. It consists of two elements:

functionalCriteria: which specifies the information that uniquely defines a web service.

qosCriteriaList: which defines the specification to determine the criteria for a web service QoS.

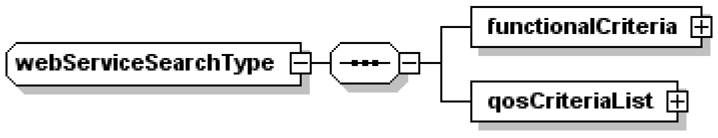
element webServiceCompositeSearchType/webServiceSearch

diagram	 webServiceSearchType webServiceSearch functionalCriteria qosCriteriaList
type	webServiceSearchType
properties	isRef 0 content complex
children	functionalCriteria qosCriteriaList
source	<xsd:element name="webServiceSearch" type="webServiceSearchType"/>

Type: webServiceSearchType

webServiceSearchType is a sequence of two complex elements that define the *functionalCriteria* and *qosCriteriaList*.

complexType webServiceSearchType

diagram	 webServiceSearchType functionalCriteria qosCriteriaList
children	functionalCriteria qosCriteriaList
used by	element webServiceCompositeSearchType/webServiceSearch

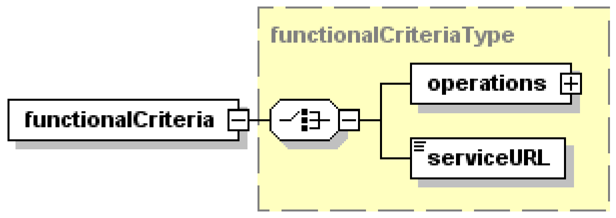
source	`<xsd:complexType name="webServiceSearchType">`
	`<xsd:sequence>`
	`<xsd:element name="functionalCriteria" type="functionalCriteriaType"/>`
	`<xsd:element name="qosCriteriaList" type="qosCriteriaContainerType"/>`
	`</xsd:sequence>`
	`</xsd:complexType>`

7.3 functionalCriteria

Element: functionalCriteria

functionalCriteria is an element of webServiceSearch that enables the clients to define the functional information of the web service they are searching for. To determine this web service they can either list the operations of that web service or the location of it by specifying the URL of that web service.

element webServiceSearchType/functionalCriteria

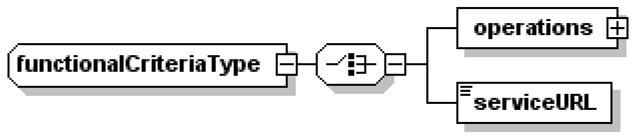
diagram	 functionalCriteriaType functionalCriteria operations serviceURL
type	functionalCriteriaType
properties	isRef 0 content complex
children	operations serviceURL
source	`<xsd:element name="functionalCriteria" type="functionalCriteriaType"/>`

Type: functionalCriteriatype

functionalCriteriatype is a choice of *operationContainerType* complex type and *anyURI* simple type.

The following table provides the schema definition for this type.

complexType functionalCriteriaType

diagram	
children	operations serviceURL
used by	element webServiceSearchType/functionalCriteria

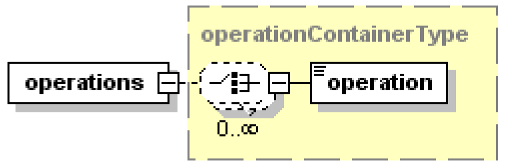
source	`<xsd:complexType name="functionalCriteriaType">`
	  `<xsd:choice>`
	    `<xsd:element name="operations" type="operationContainerType"/>`
	    `<xsd:element name="serviceURL" type="xsd:anyURI"/>`
	  `</xsd:choice>`
	`</xsd:complexType>`

7.4 operations

Element: operations

Operations element contains a list of operation elements and enables the definition of a list web service operations. The table below presents this element.

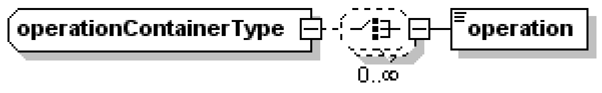
element functionalCriteriaType/operations

diagram	
type	**operationContainerType**
properties	isRef 0 content complex
children	**operation**
source	<xsd:element name="operations" type="operationContainerType"/>

Type: operationContainerType

operationContainerType is a choice of 0 to unbounded of operation elements. The table below presents the definition of this complex type.

complexType operationContainerType

diagram	
children	**operation**
used by	element **functionalCriteriaType/operations**

source	<xsd:complexType name="operationContainerType">
	  <xsd:choice minOccurs="0" maxOccurs="unbounded">
	    <xsd:element name="operation" type="xsd:string"/>
	  </xsd:choice>
	</xsd:complexType>

7.5 operation

operation is a simple type of String.

element operationContainerType/operation

diagram	
type	**xsd:string**
properties	isRef 0 content simple
source	<xsd:element name="operation" type="xsd:string"/>

7.6 serviceURL

serviceURL is a simple type of String.

element functionalCriteriaType/serviceURL

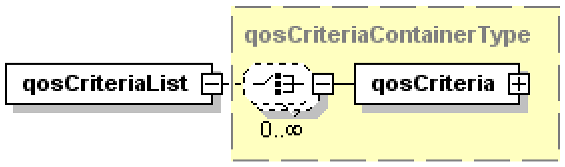
diagram	
type	**xsd:anyURI**
properties	isRef 0 content simple
source	<xsd:element name=”serviceURL” type=”xsd:anyURI”/>

7.7 qosCriteriaList

Element: qosCriteriaList

qosCriteriaList element enables the client to perform a search for a web service based on a list of QoS constraints.

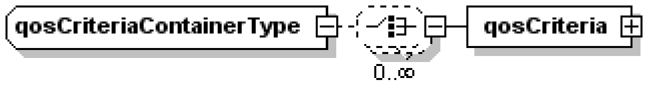
element webServiceSearchType/qosCriteriaList

diagram	 qosCriteriaContainerType qosCriteriaList qosCriteria 0..∞
type	**qosCriteriaContainerType**
properties	isRef 0 content complex
children	**qosCriteria**
source	<xsd:element name=”qosCriteriaList” type=”qosCriteriaContainerType”/>

Type: qosCriteriaContainerType

qosCriteriaContainerType is a complex type of a choice of 0 to unbounded number of qosCriteria elements. The table below provides the type definition in the schema for this complex type.

complexType qosCriteriaContainerType

diagram	
children	qosCriteria
used by	element webServiceSearchType/qosCriteriaList

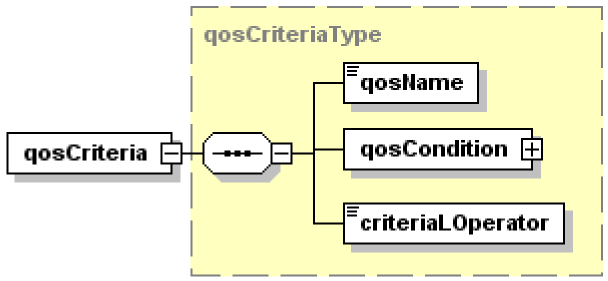
source	<xsd:complexType name="qosCriteriaContainerType">
	  <xsd:choice minOccurs="0" maxOccurs="unbounded">
	    <xsd:element name="qosCriteria" type="qosCriteriaType"/>
	  </xsd:choice>
	</xsd:complexType>

7.8 qosCriteria

Element: qosCriteria

qosCriteria element enables the specification of a QoS constraint for the web service search. It has three elements that together define a constraint for a search. The name of the quality service, the condition that is required to be met for that service and the relation of this service with the next constraint is defined in this element.

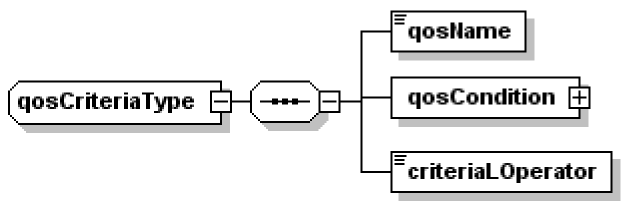
element qosCriteriaContainerType/qosCriteria

diagram	 qosCriteriaType qosName qosCriteria qosCondition criterialOperator
type	qosCriteriaType
properties	isRef 0 content complex
children	qosName qosCondition criterialOperator
source	<xsd:element name="qosCriteria" type="qosCriteriaType"/>

Type: qosCriteriaType

qosCriteriaType is a complex type. It is a sequence of a string type, qosConditionType and criteriaLOperator type. The table below presents the definition of this type in the composite web search schema.

complexType qosCriteriaType

diagram	 qosCriteriaType qosName qosCondition criteriaLOperator
children	qosName qosCondition criteriaLOperator
used by	element qosCriteriaContainerType/qosCriteria

source	`<xsd:complexType name="qosCriteriaType">`
	`  <xsd:sequence>`
	`    <xsd:element name="qosName" type="xsd:string"/>`
	`    <xsd:element name="qosCondition" type="qosConditionType"/>`
	`    <xsd:element name="criteriaLOperator" type="criteriaLOperatorType"/>`
	`  </xsd:sequence>`
	`</xsd:complexType>`

7.9 qosName

qosName is a simple type that enables the definition of the name of a QoS.

element qosCriteriaType/qosName

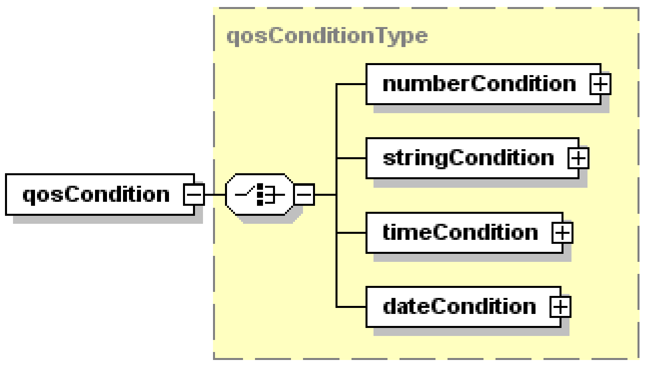
diagram	
type	**xsd:string**
properties	isRef 0 content simple
source	`<xsd:element name="qosName" type="xsd:string"/>`

7.10 qosCondition

Element: qosCondition

qosCondition element enables the specification of the type of QOS and the comparison that is required to perform on the specified QOS and the relation with the next criteria.

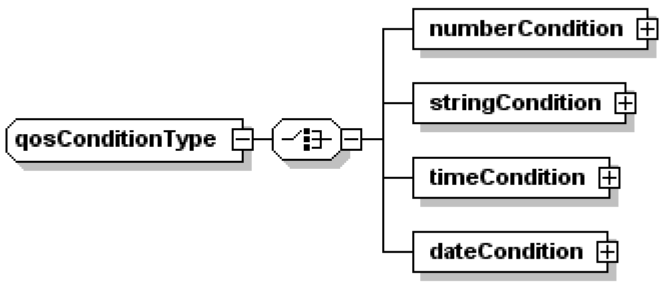
element qosCriteriaType/qosCondition

diagram	 qosConditionType qosCondition numberCondition stringCondition timeCondition dateCondition
type	qosConditionType
properties	isRef 0 content complex
children	numberCondition stringCondition timeCondition dateCondition
source	`<xsd:element name="qosCondition" type="qosConditionType"/>`

Type: qosConditionType

qosConditionType is a complex type. It is a choice between numberCondition, stringCondition, timeCondition and dateCondition. The table below presents the definition of this type in the composite web search schema.

complexType qosConditionType

diagram	 qosConditionType numberCondition stringCondition timeCondition dateCondition
children	numberCondition stringCondition timeCondition dateCondition
used by	element qosCriteriaType/qosCondition
source	see below

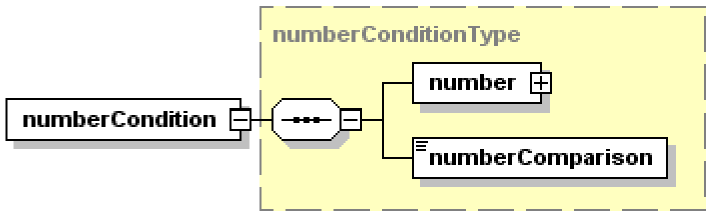
```
<xsd:complexType name="qosConditionType">
  <xsd:choice>
    <xsd:element name="numberCondition" type="numberConditionType"/>
    <xsd:element name="stringCondition" type="stringConditionType"/>
    <xsd:element name="timeCondition" type="timeConditionType"/>
    <xsd:element name="dateCondition" type="dateConditionType"/>
  </xsd:choice>
</xsd:complexType>
```

7.11 numberCondition

Element: numberCondition

numberCondition element determines that the QOS being search for is of date, its value will be defined in *numberValue* and the comparison condition is defined in *numberComparison* element as described in the table below.

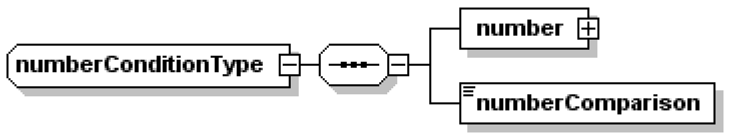
element qosConditionType/numberCondition

diagram	 numberConditionType number numberCondition numberComparison
type	numberConditionType
properties	isRef 0 content complex
children	number numberComparison
source	<xsd:element name="numberCondition" type="numberConditionType"/>

Type: numberConditionType

numberConditionType is a complex type. It is a sequence of *number* and *numberComparison* elements. The table below presents the definition of this type in the composite web search schema.

complexType numberConditionType

diagram	
children	number numberComparison
used by	element qosConditionType/numberCondition

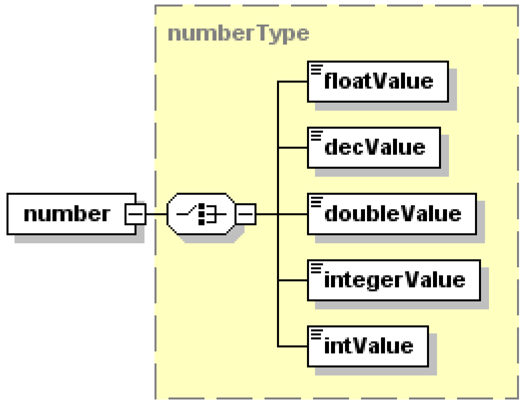
source	`<xsd:complexType name="numberConditionType">`
	`<xsd:sequence>`
	`<xsd:element name="number" type="numberType"/>`
	`<xsd:element name="numberComparison" type="numberComparisonType"/>`
	`</xsd:sequence>`
	`</xsd:complexType>`

7.12 number

Element: number

The number element determines the value of the quality service and can be any of *floatValue*, *decValue*, *doubleValue*, *integerValue* and *intValue* depending of the QOS.

Element: numberConditionType/number

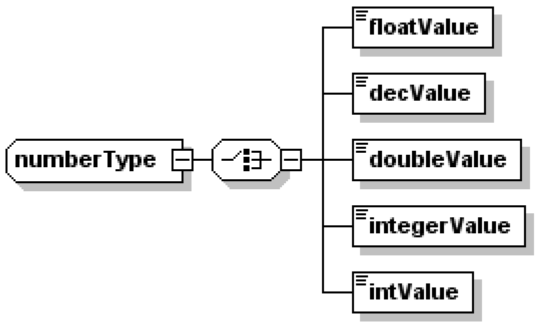
diagram	 numberType floatValue decValue number doubleValue integerValue intValue
type	numberType
properties	isRef 0 content complex
children	floatValue decValue doubleValue integerValue intValue
source	<xsd:element name="number" type="numberType"/>

Type: numberType

numberType is a choice of one of the simple types: float, decimal, double, integer and int.

The following tables present the definition of these types in the schema.

complexType numberType

diagram	 floatValue decValue numberType doubleValue integerValue intValue
children	floatValue decValue doubleValue integerValue intValue

used by	element **numberConditionType/number**

source	<xsd:complexType name=”numberType”>
	 <xsd:choice>
	  <xsd:element name=”floatValue” type=”xsd:float”/>
	  <xsd:element name=”decValue” type=”xsd:decimal”/>
	  <xsd:element name=”doubleValue” type=”xsd:double”/>
	  <xsd:element name=”integerValue” type=”xsd:integer”/>
	  <xsd:element name=”intValue” type=”xsd:int”/>
	 </xsd:choice>
	</xsd:complexType>

element numberType/floatValue

diagram	
type	**xsd:float**
properties	isRef 0 content simple
source	<xsd:element name=”floatValue” type=”xsd:float”/>

element numberType/decValue

diagram	
type	**xsd:decimal**
properties	isRef 0 content simple
source	<xsd:element name=”decValue” type=”xsd:decimal”/>

element numberType/doubleValue

diagram	
type	**xsd:double**
properties	isRef 0 content simple
source	<xsd:element name=”doubleValue” type=”xsd:double”/>

element numberType/integerValue

diagram	
type	**xsd:integer**
properties	isRef 0 content simple
source	<xsd:element name=”integerValue” type=”xsd:integer”/>

element numberType/intValue

diagram	
type	**xsd:int**
properties	isRef 0 content simple
source	<xsd:element name="intValue" type="xsd:int"/>

7.13 numberComparison

Element: numberComparison

This element determines the type of the comparison that is required to be performed on the value provided in the number element for this QoS, the comparison along with the value provides the constraint that the client is looking for in a web service QOS.

element numberConditionType/numberComparison

diagram	
type	numberComparisonType
properties	isRef 0 content simple
facets	enumeration LessThan enumeration GreaterThan enumeration Equals enumeration NotEquals enumeration LessThanOrEquals enumeration GreaterThanOrEquals
source	<xsd:element name="numberComparison" type="numberComparisonType"/>

Type: numberComparisonType

numberComparisonType is an enumeration of LessThan, GreaterThan, Equals, NotEquals, LessThanOrEquals, and GreaterThanOrEquals. The following tables presents the definition of this type in the schema.

simpleType numberComparisonType

type	restriction of **xsd:string**
used by	element numberConditionType/numberComparison
facets	enumeration LessThan enumeration GreaterThan enumeration Equals enumeration NotEquals enumeration LessThanOrEquals

	enumeration GreaterThanOrEquals

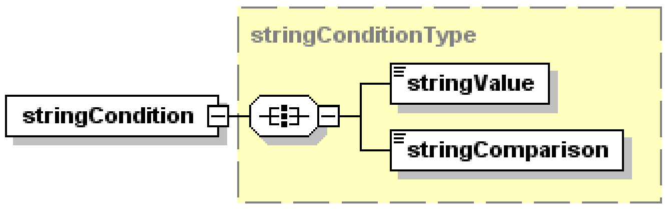
source	`<xsd:simpleType name=”numberComparisonType”>`
	` <xsd:restriction base=”xsd:string”>`
	`  <xsd:enumeration value=”LessThan”/>`
	`  <xsd:enumeration value=”GreaterThan”/>`
	`  <xsd:enumeration value=”Equals”/>`
	`  <xsd:enumeration value=”NotEquals”/>`
	`  <xsd:enumeration value=”LessThanOrEquals”/>`
	`  <xsd:enumeration value=”GreaterThanOrEquals”/>`
	` </xsd:restriction>`
	`</xsd:simpleType>`

7.14 stringCondition

Element: stringCondition

stringCondition element determines that the QOS being search for is of string, its value will be defined in *stringValue* and the comparison condition is defined in *stringComparison* element as described in the table below.

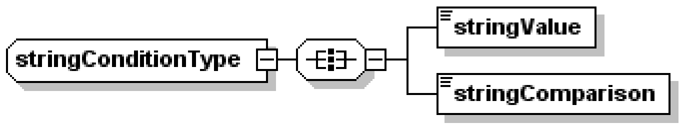
element qosConditionType/stringCondition

diagram	 stringConditionType stringCondition stringValue stringComparison
type	stringConditionType
properties	isRef 0 content complex
children	stringValue stringComparison
source	`<xsd:element name=”stringCondition” type=”stringConditionType”/>`

Type: stringConditionType

stringConditionType is a complex type. It is a sequence of *stringValue* and *stringComparison* elements. The table below presents the definition of this type in the composite web search schema.

complexType stringConditionType

diagram	
children	stringValue stringComparison
used by	element qosConditionType/stringCondition

source	`<xsd:complexType name=”stringConditionType”>`
	  `<xsd:all>`
	    `<xsd:element name=”stringValue” type=”xsd:string”/>`
	    `<xsd:element name=”stringComparison” type=”stringComparisonType”/>`
	  `</xsd:all>`
	`</xsd:complexType>`

7.15 stringValue

element stringConditionType/stringValue

diagram	
type	**xsd:string**
properties	isRef 0 content simple
source	`<xsd:element name=”stringValue” type=”xsd:string”/>`

7.16 stringComparison

Element: stringComparison

This element determines the type of the comparison that is required to be performed on the value provided in the *stringValue* element for this QoS, the comparison along with the value provides the constraint that the client is looking for in a web service QOS.

element stringConditionType/stringComparison

diagram	
type	**stringComparisonType**
properties	isRef 0 content simple
facets	enumeration LessThan enumeration GreaterThan enumeration LessThanOrEquals enumeration GreaterThanOrEquals enumeration Equals enumeration NotEquals enumeration Contains enumeration BeginsWith
source	`<xsd:element name=”stringComparison” type=”stringComparisonType”/>`

Type: stringComparisonType

stringComparisonType is an enumeration of LessThan, GreaterThan, Equals, NotEquals, LessThanOrEquals, GreaterThanOrEquals, Contains and BeginsWith. The following tables present the definition of these types in the schema.

simpleType stringComparisonType

type	restriction of **xsd:string**
used by	element **stringConditionType/stringComparison**
facets	enumeration LessThan enumeration GreaterThan enumeration LessThanOrEquals enumeration GreaterThanOrEquals enumeration Equals enumeration NotEquals enumeration Contains enumeration BeginsWith

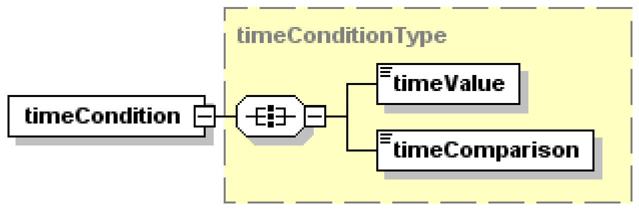
source	`<xsd:simpleType name="stringComparisonType">`
	` <xsd:restriction base="xsd:string">`
	`  <xsd:enumeration value="LessThan"/>`
	`  <xsd:enumeration value="GreaterThan"/>`
	`  <xsd:enumeration value="LessThanOrEquals"/>`
	`  <xsd:enumeration value="GreaterThanOrEquals"/>`
	`  <xsd:enumeration value="Equals"/>`
	`  <xsd:enumeration value="NotEquals"/>`
	`  <xsd:enumeration value="Contains"/>`
	`  <xsd:enumeration value="BeginsWith"/>`
	` </xsd:restriction>`
	`</xsd:simpleType>`

7.17 timeCondition

Element: timeCondition

timeCondition element determines that the QOS being search for is of time, its value will be defined in *timeValue* and the comparison condition is defined in *timeComparison* element as described in the table below.

element qosConditionType/timeCondition

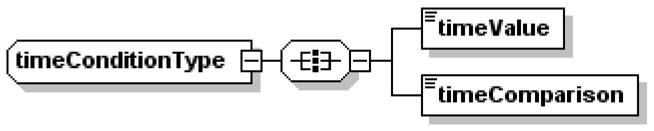
diagram	 timeConditionType timeCondition timeValue timeComparison
type	**timeConditionType**
properties	isRef 0 content complex

children	timeValue timeComparison
source	<xsd:element name="timeCondition" type="timeConditionType"/>

Type: timeConditionType

timeConditionType is a complex type. It is a sequence of *timeValue* and *timeComparison* elements. The table below presents the definition of this type in the composite web search schema.

complexType timeConditionType

diagram	
children	timeValue timeComparison
used by	element qosConditionType/timeCondition

source	<xsd:complexType name="timeConditionType">
	  <xsd:all>
	    <xsd:element name="timeValue" type="xsd:time"/>
	    <xsd:element name="timeComparison" type="timeComparisonType"/>
	  </xsd:all>
	</xsd:complexType>

7.18 timeValue

timeValue is a simple type element that defines a time value.

element timeConditionType/timeValue

diagram	
type	**xsd:time**
properties	isRef 0 content simple
source	<xsd:element name="timeValue" type="xsd:time"/>

7.19 timeComparison

Element: timeComparison

This element determines the type of the comparison that is required to be performed on the value provided in the time element for this QoS, the comparison along with the value provides the constraint that the client is looking for in a web service QOS.

element timeConditionType/timeComparison

diagram	
type	timeComparisonType
properties	isRef 0 content simple
facets	enumeration LessThan enumeration GreaterThan enumeration LessThanOrEquals enumeration GreaterThanOrEquals enumeration Equals enumeration NotEquals
source	`<xsd:element name=”timeComparison” type=”timeComparisonType”/>`

Type: timeComparisonType

timeComparisonType is an enumeration of LessThan, GreaterThan, Equals, NotEquals, LessThanOrEquals, and GreaterThanOrEquals. The following tables present the definition of these types in the schema.

simpleType timeComparisonType

type	restriction of **xsd:string**
used by	element timeConditionType/timeComparison
facets	enumeration LessThan enumeration GreaterThan enumeration LessThanOrEquals enumeration GreaterThanOrEquals enumeration Equals enumeration NotEquals
source	see below

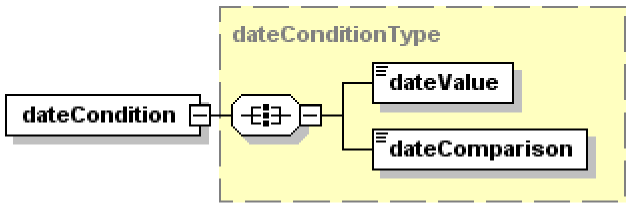
```
<xsd:simpleType name=”timeComparisonType”>
  <xsd:restriction base=”xsd:string”>
    <xsd:enumeration value=”LessThan”/>
    <xsd:enumeration value=”GreaterThan”/>
    <xsd:enumeration value=”LessThanOrEquals”/>
    <xsd:enumeration value=”GreaterThanOrEquals”/>
    <xsd:enumeration value=”Equals”/>
    <xsd:enumeration value=”NotEquals”/>
  </xsd:restriction>
</xsd:simpleType>
```

7.20 dateCondition

Element: dateCondition

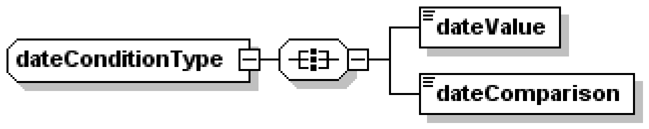
dateCondition element determines that the QOS being search for is of date, its value will be defined in *dateValue* and the comparison condition is defined in *dateComparison* element as described in the table below.

element qosConditionType/dateCondition

diagram	 dateConditionType dateCondition dateValue dateComparison
type	dateConditionType
properties	isRef 0 content complex
children	dateValue dateComparison
source	<xsd:element name="dateCondition" type="dateConditionType"/>

Type: dateConditionType

complexType dateConditionType

diagram	
children	dateValue dateComparison
used by	element qosConditionType/dateCondition

source	<xsd:complexType name="dateConditionType">
	  <xsd:all>
	    <xsd:element name="dateValue" type="xsd:date"/>
	    <xsd:element name="dateComparison" type="dateComparisonType"/>
	  </xsd:all>
	</xsd:complexType>

7.21 dateValue

element dateConditionType/dateValue

diagram	
type	**xsd:date**
properties	isRef 0 content simple
source	<xsd:element name="dateValue" type="xsd:date"/>

7.22 dateComparison

Element: dateComparison

This element determines the type of the comparison that is required to be performed on the value provided in the dateValue element for this QoS, the comparison along with the value provides the constraint that the client is looking for in a web service QOS.

element dateConditionType/dateComparison

diagram	
type	dateComparisonType
properties	isRef 0 content simple
facets	enumeration LessThan enumeration GreaterThan enumeration LessThanOrEquals enumeration GreaterThanOrEquals enumeration Equals enumeration NotEquals
source	<xsd:element name="dateComparison" type="dateComparisonType"/>

Type: dateComparisonType

dateComparisonType is an enumeration of LessThan, GreaterThan, Equals, NotEquals, LessThanOrEquals, and GreaterThanOrEquals. The following tables present the definition of these types in the schema.

simpleType dateComparisonType

type	restriction of **xsd:string**
used by	element dateConditionType/dateComparison
facets	enumeration LessThan enumeration GreaterThan

	enumeration LessThanOrEquals enumeration GreaterThanOrEquals enumeration Equals enumeration NotEquals

source	`<xsd:simpleType name="dateComparisonType">`
	`  <xsd:restriction base="xsd:string">`
	`    <xsd:enumeration value="LessThan"/>`
	`    <xsd:enumeration value="GreaterThan"/>`
	`    <xsd:enumeration value="LessThanOrEquals"/>`
	`    <xsd:enumeration value="GreaterThanOrEquals"/>`
	`    <xsd:enumeration value="Equals"/>`
	`    <xsd:enumeration value="NotEquals"/>`
	`  </xsd:restriction>`
	`</xsd:simpleType>`

7.23 criteriaLOperator

Element: criteriaLOperator

criteriaLOperator enables the specification of an Or or And operation. Based on this operator each qosCriteria is going to be combined with the next qosCriteria defined in the XML document.

element qosCriteriaType/criteriaLOperator

diagram	
type	**criteriaLOperatorType**
properties	isRef 0 content simple
facets	enumeration Or enumeration And
source	`<xsd:element name="criteriaLOperator" type="criteriaLOperatorType"/>`

Type: criteriaLOperatorType

criteriaLOperatorType is an enumeration of Or and And. The following tables present the definition of these types in the schema.

simpleType criteriaLOperatorType

type	restriction of **xsd:string**
used by	element **qosCriteriaType/criteriaLOperator**
facets	enumeration Or enumeration And

source	`<xsd:simpleType name="criteriaLOperatorType">`
	`  <xsd:restriction base="xsd:string">`
	`    <xsd:enumeration value="Or"/>`
	`    <xsd:enumeration value="And"/>`
	`  </xsd:restriction>`

	`</xsd:simpleType>`

7.24 Summary

Chapter 7 provides the detail definition of an XML schema that the WSQEF framework XML documents conform to. The XML documents are used by clients to define a set of Web Service components that they require to compose along with their functional and QoS requirements for each of the Web Services. The Web sevice providers also use this schema to create XML documents to define their service's functional and QoS criteria.

8 Conclusions, Summary of Contributions

8.1 Conclusions

The focus of this thesis is to address a set of problems described in section 1.2. In this section we review the problems and outline the approaches taken to address them.

The main problem stated in section 1.2.1 is the need to enable QoS aware dynamic Web Service composition. This thesis proposes a framework to solve this issue. The framework is the integration of existing technologies and newly designed components that can interoperate. The WSQEF framework architecture is explained in detail in chapter 4.

Figure 14 illustrates a logical view of the framework and the following subsections describe the components and how they interact. As one of the requirements of this thesis, the proposal considers the use of existing technologies where possible. WSQEF integrates P2P and Web Services composition technologies, by exploiting JXTA and BPEL. The benefits of using JXTA are outlined in section 2.5.1. BPEL is selected as the Web Service composition definition language, described in section 2.7.4. WSQEF introduces a mechanism for clients to provide their required QoS in XML documents. Service providers as a service entity in JXTA platform publish their services. A search service in designed as another service entity in JXTA platform that has the responsibility of discovering the Web Services. The input to the framework is the QoS requirements defined in XML documents by clients and the output is the required BPEL documents that are generated based on dynamic selection. The BPWS4J executes the generated BPEL documents. There is no further dynamic selection when the engine executes the Web Service Composition that is described in BPEL documents. An XML schema is developed as one of the thesis contributions to provide the clients to define their functional and QoS Web Service selection criteria in a generic and flexible way, the schema is described in detail in section 5.1

Another problem stated in section 1.2.2 is the issue of security that arises when businesses need to communicate with each other through the Internet. A P2P environment has been selected in this framework for the security benefits it provides. The entities in this framework are part of an overlay network where secure communication can be enforced. JXTA provides some security mechanisms described in section 2.5.5 and also provides a base for additional security features to provide privacy, integrity and authentication.

Using the JXTA platform brings the benefit of search and discovery of services in a decentralized environment addressing the problem stated in 1.2.3. JXTA also provides advertisement documents (described in 2.5.3.5) that enable all service entities to publish their information. Extensibility of advertisements is used in the proposal to enable service providers to publish their functional and QoS information. As the result of this thesis a prototype is implemented to prove the main components of the architecture are interoperable and the goal of the framework can be met, this experiment is discussed in chapter 6.

8.2 Summary of Contributions

The contributions to this thesis are as follows. A framework that integrates the existing technologies with newly created components that fulfil a dynamic Web Service composition. The selection of Web Services is based on the client’s specification of functional and QoS requirements for each of the Web Services in the composition. The other contribution is an XML schema to enable clients to specify their Web Service search criteria based on functionality and QoS, the schema is provides a generic approach to define QoS criteria. An enhanced search service to search and select the Web Services based on the criteria provided by requester was designed as part of this thesis. Also prototypes are implemented as proof of concept for the contributions that are made above.

8.3 Future Work

Various future researches can be undertaken to improve the functionality of the proposed framework. Some possibilities are outlined here:

8.3.1 Dynamic QoS Measurement Service

The QoS attributes of the current framework are at user level and are based on what the service providers announce, a measuring service entity can be added to the framework as a JXTA service that constantly measures the system level QoS attributes that change dynamically, such as response time of the Web Service, traffic and bandwidth. The Web Service advertiser can use this service to periodically advertise the new values for changeable QoS properties.

8.3.2 Optimised Search Service

The Enhanced Search Service in the proposed framework, query the JXTA network and conducts a thorough match based on the XML search criteria provided. If the service provider numbers within a category increase significantly, the search service performance decreases and causes a bottleneck. An optimised search mechanism can be introduced to filter through a set of services and perform the matching among subsets of services advertised in the JXTA network. We described a constraint driven project in section 3.3.2. The modules of optimisation in this project can be integrated with this thesis as a service entity in JXTA.

8.3.3 QoS aware BPEL documents

The result of the WSQEF framework is the BPEL documents that fulfil a Web Service Composition. The selection of the Web Services is performed based on the QoS defined by the clients and providers. The BPEL documents can be extended in order to include the QoS information for the Web Service Composition. This way the created documents can be reused if the QoS criteria are met.

8.3.4 Management of Web Service Composition

A management layer can be designed on top of the Web Service Composition framework to enable more dynamism by initiating composite searches, monitoring and fault handling of composite execution. For example initiating a Web Service search request if a Web Service is unavailable or generates error while accessing it, this provides re-composition on the fly. Also a management layer can provide more dynamism by execution of the BPEL documents dynamically as soon as they are created.

8.3.5 Improvement of User interface

A Graphical User Interface can facilitate XML configurations for both clients and Web Service providers. The components in the framework are all started as JXTA peers, the start-up of peers can be simplified by providing a GUI and some underlying work.

9 References

Note: The Internet links in the references were valid on January 17, 2005.

[1] Krishnan N. “The JXTA solution to P2P”, JavaWorld Articles, Oct 2001, PP. 1, on-line at: http://www.javaworld.com/javaworld/jw-10-2001/jw-1019-jxta.html

[2] Schollmeier R., “A definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Applications”, Sweden. IEEE Computer Society 2001, PP. 101-102

[3] Sean McCarthy et al. “Survey on P2P File Sharing System”, University of California, Irvine, School of Information and Computer Science, ICS 243a, Student Presentations, 2001, PP. 7, online-at: http://netresearch.ics.uci.edu/classes/2001/243A/P2P%20group%201%20presentation4.pdf

[4] Sean McCarthy et al. “Survey on P2P File Sharing System”, University of California, Irvine, School of Information and Computer Science, ICS 243a, Student Presentations, 2001, PP. 10, online-at: http://netresearch.ics.uci.edu/classes/2001/243A/P2P%20group%201%20presentation4.pdf

[5] Sun Microsystems, Inc., JXTA Project, definition, on-line at: http://www.jxta.org/

[6] JXTAKOREA Inc., JXTA Intro, JXTA Overview, on-line at: http://www.jxtakorea.net/contents/jxta1.php

[7] Moats R. The Internet Engineering Task Force (IETF), Network Working Group, URN Syntax, on-line at: http://www.ietf.org/html.charters/urn-charter.html

[8] Sun Microsystems, Inc. JXTA v2.1 Protocols Specification http://spec.jxta.org/nonav/v1.0/docbook/JXTAProtocols.html

[9] Brookshier D., Krishnan N. "JXTA: Java P2P Programming", Book, First Edition, Sams Inc., March 2002, Chapter 8, "JXTA and Security": available at: http://java.sun.com/developer/Books/networking/jxta/jxtap2pch08.pdf

[10] Christensen, E., Curbera, F., Meredith, G., Weerawarana S. "Web Services Description Language (WSDL) 1.1", W3C Note, Ariba, International Business Machines Corporation and Microsoft, March 15, 2001, on-line at: http://www.w3.org/TR/wsdl

[11] Box D., Ehnebuske D., Kakivaya G. "Simple Object Access Protocol (SOAP) 1.1", World Wide Web Consortium, on-line at: http://www.w3.org/TR/soap/

[12] Philippe Le Hégaret , "Web Services: SOAP, WSDL and Choreography and Integration of XML Technologies", W3C Day Japan 2003 at Keio University, PP. 19, on-line at: http://www.w3.org/2003/Talks/1114-W3CDay-Japan/plh-ws/

[13] Cheng Yushi, Lee Eng Wah, Dilip Kumar Limbu: "Web Service Composition – An overview of Standards", Synthesis 2004 Journal, Section 4, Oct 2004, on-line at: http://www.itsc.org.sg/synthesis/2004/4_WS.pdf

[14] S.Thatte. XLANG: Web Services for Business Process Design. Microsoft, 2001, on-line at: http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm

[15] Leymann F., IBM Software Group, May 2001, WWW page, on-line at: http://www-306.ibm.com/software/solutions/webservices/pdf/WSFL.pdf

[16] Andrews T., Curbera F., Dholakia H., Business Process Execution Language for Web Services Version 1.1, developerWorks, IBM, May 2003, on-line at: http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/

[17] Christoph Schittko, "Web Service Orchestration with BPEL", Momentum Software Inc, XML Conference and Exposition, Dec 2003, on-line at: http://www.idealliance.org/papers/dx_xml03/index/title/37a1e4c99a1521ef30aa63c901.html

[18] Tosic, V., Patel, K., Pagurek, B. "Web Service Offerings Language", In Proceedings Of the Workshop on Web Services, e-Business, and the Semantic Web at CaiSE'02, Toronto, Canada, May 2002, PP. 468-484.

[19] Dan, A., Franck, R., Keller, A., King, R., Ludwig, H.: Web Service Level Agreement (WSLA) Language Specification. In Documentation for the Web Services Toolkit,

Ver. 3.2.1. Aug. 9, 2002. IBM (Int. Business Machines) Corporation (2002), on-line at: http://www.research.ibm.com/wsla/WSLASpecV1-20030128.pdf

[20] Sivashanmugam K. et al., “Framework for Semantic Web Process Composition”, Technical Report 03-008, LSDIS Lab, Computer Science Dept., UGA, June 2003, on-line at: http://lsdis.cs.uga.edu/lib/download/TR03-008.pdf

[21] Aggarwal R. et al., Constraint Driven Web Service Composition in METEOR-S, In Proceedings of IEEE SCC, Shanghai, Sep 2004, PP. 23-30.

[22] Zeng L., Benatallah B., Dumas M., “Quality Driven Web Services Composition”, In Proceedings of the twelfth international conference on World Wide Web, Web engineering Session, Hungary, 2003, PP. 411 – 421.

[23] Sheng Q, Benatallah B, Dumas2 Eileen M., Mak O. “SELF-SERV: A Platform for Rapid Composition of Web Services in a Peer-to-Peer Environment”, In Proceedings Demonstration Session of the 28th International Conference on Very Large Databases (VLDB). Hong Kong, China, August 2002. Morgan Kaufmann, on-line at: http://www.cs.ust.hk/vldb2002/VLDB2002-proceedings/papers/S33P03.pdf

[24] Benatallah, B., Sheng, Q., Dumas, M., “The self-serv environment for web services composition”, IEEE Internet Computing 7, 2003, PP. 40-48.

[25] Laoveerakul S.et al., “Decentralized UDDI based on P2P”, APAN 2002 Conference in Shanghai, P2P/NSCD session, PP. 2, on-line at: http://www.hpcc.nectec.or.th/C4/grid/UDDI.pdf

[26] Esfandiari B., Tosic V., "Requirements for Web Service Composition Management, Requirement 1.1", In Proceedings Of the Hewlett-Packard OpenView University Association Workshop, June 2004, France, Hewlett-Packard, on-line at: http://www.hpovua.org/PUBLICATIONS/PROCEEDINGS/11_HPOVUAWS/hpov2004/www/ConferenceProgramme.htm

[27] Verma K.et al., "METEOR–S WSDI: A Scalable P2P Infrastructure of Registries for Semantic Publication and Discovery of Web Services", Journal of Information Technology and Management, 2004, on-line at: http://lsdis.cs.uga.edu/lib/download/MWSDI.pdf

[28] Mennie D., "An Architecture to Support Dynamic Composition of Service Components and its Applicability to Internet Security", Carleton University, Ottawa, M.Eng. Thesis, September 2000, on-line at: http://www.sce.carleton.ca/netmanage/papers/MennieThesis.pdf

[29] Majithia S., Taylor I., Shields M., Wang I., "Triana as a Graphical Web Services Composition Toolkit", IEEE International Conference on Web Services (ICWS'04), June 2004, San Diego, PP. 514, on-line at: http://www.wesc.ac.uk/resources/publications/pdf/TrianaAHM2003.pdf

[30] University of Georgia, Large Scale Distributed Information Systems lab, METEOR-S: Semantic Web Services and Processes, WWW page, on-line at: http://swp.semanticweb.org/

[31] Project JXTA v2.0: Java Programmer's Guide

http://www.jxta.org/docs/JxtaProgGuide_v2.pdf

10 Glossary

BPEL	Business Process Execution Language
BPML	Business Process Modelling Language
QoS	Quality of Service
NAT	Network Address Translation
PC	Personal Computer
PDA	Personal Digital Assistant
SOAP	Simple Object Access Protocol
UML	Unified Modeling Language
UDDI	Universal Description, Discovery and Integration.
WSDL	Web Service Definition Language
WSFL	Web Services Flow Language
XML	eXtensible Mark-up Language

Appendix A –

Here is the XML Schema for Web Service Composite Search with QoS:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSPY v2004 rel. 4 U (http://www.xmlspy.com) by Mo tan (mometan) --
>
<xsd:schema elementFormDefault="qualified" attributeFormDefault="unqualified"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:element name="webServiceCompositeSearch"
type="webServiceCompositeSearchType">
	<xsd:annotation>
		<xsd:documentation>Web Serives Search Criteria</xsd:documentation>
	</xsd:annotation>
	</xsd:element>
	<xsd:complexType name="webServiceCompositeSearchType">
		<xsd:choice minOccurs="0" maxOccurs="unbounded">
			<xsd:element name="webServiceSearch"
type="webServiceSearchType"/>
		</xsd:choice>
	</xsd:complexType>
	<xsd:complexType name="webServiceSearchType">
		<xsd:sequence>
			<xsd:element name="functionalCriteria"
type="functionalCriteriaType"/>
			<xsd:element name="qosCriteriaList"
type="qosCriteriaContainerType"/>
		</xsd:sequence>
	</xsd:complexType>
	<xsd:complexType name="functionalCriteriaType">
		<xsd:choice>
			<xsd:element name="operations" type="operationContainerType"/>
			<xsd:element name="serviceURL" type="xsd:anyURI"/>
		</xsd:choice>
	</xsd:complexType>
	<xsd:complexType name="operationContainerType">
		<xsd:choice minOccurs="0" maxOccurs="unbounded">
			<xsd:element name="operation" type="xsd:string"/>
		</xsd:choice>
	</xsd:complexType>
	<xsd:complexType name="qosCriteriaContainerType">
		<xsd:choice minOccurs="0" maxOccurs="unbounded">
```

```
                    <xsd:element name="qosCriteria" type="qosCriteriaType"/>
            </xsd:choice>
    </xsd:complexType>
    <xsd:complexType name="qosCriteriaType">
            <xsd:sequence>
                    <xsd:element name="qosName" type="xsd:string"/>
                    <xsd:element name="qosCondition" type="qosConditionType"/>
                    <xsd:element name="criteriaLOperator"
type="criteriaLOperatorType"/>
            </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="qosConditionType">
            <xsd:choice>
                    <xsd:element name="numberCondition"
type="numberConditionType"/>
                    <xsd:element name="stringCondition" type="stringConditionType"/>
                    <xsd:element name="timeCondition" type="timeConditionType"/>
                    <xsd:element name="dateCondition" type="dateConditionType"/>
            </xsd:choice>
    </xsd:complexType>
    <xsd:complexType name="numberConditionType">
            <xsd:sequence>
                    <xsd:element name="number" type="numberType"/>
                    <xsd:element name="numberComparison"
type="numberComparisonType"/>
            </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="numberType">
            <xsd:choice>
                    <xsd:element name="floatValue" type="xsd:float"/>
                    <xsd:element name="decValue" type="xsd:decimal"/>
                    <xsd:element name="doubleValue" type="xsd:double"/>
                    <xsd:element name="integerValue" type="xsd:integer"/>
                    <xsd:element name="intValue" type="xsd:int"/>
            </xsd:choice>
    </xsd:complexType>
    <xsd:simpleType name="numberComparisonType">
            <xsd:restriction base="xsd:string">
                    <xsd:enumeration value="LessThan"/>
                    <xsd:enumeration value="GreaterThan"/>
                    <xsd:enumeration value="Equals"/>
                    <xsd:enumeration value="NotEquals"/>
                    <xsd:enumeration value="LessThanOrEquals"/>
                    <xsd:enumeration value="GreaterThanOrEquals"/>
            </xsd:restriction>
```

```
        </xsd:simpleType>
        <xsd:complexType name="stringConditionType">
                <xsd:all>
                        <xsd:element name="stringValue" type="xsd:string"/>
                        <xsd:element name="stringComparison"
type="stringComparisonType"/>
                </xsd:all>
        </xsd:complexType>
        <xsd:simpleType name="criteriaLOperatorType">
                <xsd:restriction base="xsd:string">
                        <xsd:enumeration value="Or"/>
                        <xsd:enumeration value="And"/>
                </xsd:restriction>
        </xsd:simpleType>
        <xsd:simpleType name="stringComparisonType">
                <xsd:restriction base="xsd:string">
                        <xsd:enumeration value="LessThan"/>
                        <xsd:enumeration value="GreaterThan"/>
                        <xsd:enumeration value="LessThanOrEquals"/>
                        <xsd:enumeration value="GreaterThanOrEquals"/>
                        <xsd:enumeration value="Equals"/>
                        <xsd:enumeration value="NotEquals"/>
                        <xsd:enumeration value="Contains"/>
                        <xsd:enumeration value="BeginsWith"/>
                </xsd:restriction>
        </xsd:simpleType>
        <xsd:complexType name="dateConditionType">
                <xsd:all>
                        <xsd:element name="dateValue" type="xsd:date"/>
                        <xsd:element name="dateComparison"
type="dateComparisonType"/>
                </xsd:all>
        </xsd:complexType>
        <xsd:simpleType name="dateComparisonType">
                <xsd:restriction base="xsd:string">
                        <xsd:enumeration value="LessThan"/>
                        <xsd:enumeration value="GreaterThan"/>
                        <xsd:enumeration value="LessThanOrEquals"/>
                        <xsd:enumeration value="GreaterThanOrEquals"/>
                        <xsd:enumeration value="Equals"/>
                        <xsd:enumeration value="NotEquals"/>
                </xsd:restriction>
        </xsd:simpleType>
        <xsd:complexType name="timeConditionType">
                <xsd:all>
```

```
                        <xsd:element name="timeValue" type="xsd:time"/>
                        <xsd:element name="timeComparison"
type="timeComparisonType"/>
                </xsd:all>
        </xsd:complexType>
        <xsd:simpleType name="timeComparisonType">
                <xsd:restriction base="xsd:string">
                        <xsd:enumeration value="LessThan"/>
                        <xsd:enumeration value="GreaterThan"/>
                        <xsd:enumeration value="LessThanOrEquals"/>
                        <xsd:enumeration value="GreaterThanOrEquals"/>
                        <xsd:enumeration value="Equals"/>
                        <xsd:enumeration value="NotEquals"/>
                </xsd:restriction>
        </xsd:simpleType>
</xsd:schema>
```

Appendix B –

B.1 System Requirement

JXTA platform is used in implementing the framework. The current Project JXTA J2SE platform binding requires a platform that supports the Java Run-Time Environment (JRE) or Software Development Kit (SDK) 1.3.1 release or later. JRE or SDK can be downloaded from: http://java.sun.com/j2se/downloads/index.html. JXTA libraries and demos can be downloaded from: http://download.jxta.org/index.html. Eclipse has been used in this project as the IDE to facilitate the development process, Eclipse is an open platform and can be downloaded from: http://www.eclipse.org/downloads/index.php

B.2 XML documents of the assessment

B.2.1 Loan Approver WSDL

Table 19 is the WSDL document for the loan approver Web Service.

```
<definitions  targetNamespace="http://tempuri.org/services/loanapprover"
                         xmlns:tns="http://tempuri.org/services/loanapprover"
                         xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                         xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
                         xmlns:loandef="http://tempuri.org/services/loandefinitions"
                         xmlns="http://schemas.xmlsoap.org/wsdl/">

   <import  namespace="http://tempuri.org/services/loandefinitions"
                  location="http://localhost:8080/bpws4j-samples/loanapproval/loandefinitions.wsdl"/>

   <message  name="approvalMessage">
      <part  name="accept"  type="xsd:string"/>
   </message>

   <portType  name="loanApprovalPT">
      <operation  name="approve">
         <input  message="loandef:creditInformationMessage"/>
         <output  message="tns:approvalMessage"/>
         <fault  name="loanProcessFault"
                        message="loandef:loanRequestErrorMessage"/>
      </operation>
   </portType>

   <binding  name="SOAPBinding"  type="tns:loanApprovalPT">
      <soap:binding  style="rpc"
                              transport="http://schemas.xmlsoap.org/soap/http"/>
      <operation  name="approve">
         <soap:operation  soapAction=""  style="rpc"/>
         <input>
```

```
                <soap:body  use="encoded"  namespace="urn:loanapprover"
                            encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
            </input>
            <output>
                <soap:body  use="encoded"  namespace="urn:loanapprover"
                                encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
            </output>
          </operation>
    </binding>

    <service  name="LoanApprover">
        <documentation>Loan  Approver  Service</documentation>
        <port  name="SOAPPort"  binding="tns:SOAPBinding">
            <soap:address  location="http://localhost:8080/bpws4j-samples/servlet/rpcrouter"/>
        </port>
    </service>

</definitions>
```

Table 24: Loan Approver WSDL document

B.2.2 Loan Assessor WSDL

Table 19 is the WSDL document for the assessor Web Service.

```
<definitions  targetNamespace="http://tempuri.org/services/loanassessor"
              xmlns:tns="http://tempuri.org/services/loanassessor"
              xmlns:xsd="http://www.w3.org/2001/XMLSchema"
              xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
              xmlns:loandef="http://tempuri.org/services/loandefinitions"
              xmlns="http://schemas.xmlsoap.org/wsdl/">


   <import  namespace="http://tempuri.org/services/loandefinitions"
            location="http://localhost:8080/bpws4j-samples/loanapproval/loandefinitions.wsdl"/>

   <message  name="riskAssessmentMessage">
      <part  name="risk"  type="xsd:string"/>
   </message>

   <portType  name="riskAssessmentPT">
      <operation  name="check">
         <input  message="loandef:creditInformationMessage"/>
         <output  message="tns:riskAssessmentMessage"/>
         <fault  name="loanProcessFault"
                 message="loandef:loanRequestErrorMessage"/>
      </operation>
   </portType>

   <binding  name="SOAPBinding"  type="tns:riskAssessmentPT">
      <soap:binding  style="rpc"
                     transport="http://schemas.xmlsoap.org/soap/http"/>
      <operation  name="check">
         <soap:operation  soapAction=""  style="rpc"/>
         <input>
            <soap:body  use="encoded"  namespace="urn:loanassessor"
                        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
            </input>
            <output>
                <soap:body  use="encoded"  namespace="urn:loanassessor"
                                encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
            </output>
          </operation>
    </binding>

    <service  name="LoanAssessor">
        <documentation>Loan  Assessor  Service</documentation>
        <port  name="SOAPPort"  binding="tns:SOAPBinding">
            <soap:address  location="http://localhost:8080/bpws4j-samples/servlet/rpcrouter"/>
        </port>
    </service>

</definitions>
```

Table 25: Loan Assessor WSDL document

B.2.3 Loan Approval BPEL

Table 19 is the BPEL document for the loan and assessment business scenario.

```
<process  name="loanApprovalProcess"
                targetNamespace="http://acme.com/loanprocessing"
                suppressJoinFailure="yes"
                xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
                xmlns:lns="http://loans.org/wsdl/loan-approval"
                xmlns:loandef="http://tempuri.org/services/loandefinitions"
                xmlns:asns="http://tempuri.org/services/loanassessor"
                xmlns:apns="http://tempuri.org/services/loanapprover">

    <variables>
        <variable  name="request"
                            messageType="loandef:creditInformationMessage"/>
        <variable  name="riskAssessment"
                            messageType="asns:riskAssessmentMessage"/>
        <variable  name="approvalInfo"
                            messageType="apns:approvalMessage"/>
        <variable  name="error"
                            messageType="loandef:loanRequestErrorMessage"/>
    </variables>

    <partnerLinks>
        <partnerLink  name="customer"
                           partnerLinkType="lns:loanApprovalLinkType"
                           myRole="approver"/>
        <partnerLink  name="approver"
                           partnerLinkType="lns:loanApprovalLinkType"
                           partnerRole="approver"/>
        <partnerLink  name="assessor"
                           partnerLinkType="lns:riskAssessmentLinkType"
                           partnerRole="assessor"/>
    </partnerLinks>

    <faultHandlers>
        <catch  faultName="lns:loanProcessFault"
                      faultVariable="error">
            <reply  partnerLink="customer"
                           portType="apns:loanApprovalPT"
                           operation="approve"
                           variable="error"
                           faultName="invalidRequest"/>
        </catch>
    </faultHandlers>

    <flow>
        <links>
            <link  name="receive-to-assess"/>
            <link  name="receive-to-approval"/>
            <link  name="approval-to-reply"/>
            <link  name="assess-to-setMessage"/>
            <link  name="setMessage-to-reply"/>
            <link  name="assess-to-approval"/>

        </links>

        <receive  name="receive1"  partnerLink="customer"
```

```
             portType="apns:loanApprovalPT"
             operation="approve"  variable="request"
             createInstance="yes">
    <source  linkName="receive-to-assess"
              transitionCondition="bpws:getVariableData('request',  'amount')&lt;10000"/>
    <source  linkName="receive-to-approval"
              transitionCondition="bpws:getVariableData('request',  'amount')&gt;=10000"/>
</receive>

<invoke  name="invokeAssessor"  partnerLink="assessor"
            portType="asns:riskAssessmentPT"
            operation="check"
            inputVariable="request"
            outputVariable="riskAssessment">
    <target  linkName="receive-to-assess"/>
    <source  linkName="assess-to-setMessage"
              transitionCondition="bpws:getVariableData('riskAssessment',  'risk')='low'"/>
    <source  linkName="assess-to-approval"
              transitionCondition="bpws:getVariableData('riskAssessment',  'risk')!='low'"/>
</invoke>

<assign  name="assign">
    <target  linkName="assess-to-setMessage"/>
    <source  linkName="setMessage-to-reply"/>
    <copy>
        <from  expression="'yes'"/>
        <to  variable="approvalInfo"  part="accept"/>
    </copy>
</assign>

<invoke  name="invokeapprover"
            partnerLink="approver"  portType="apns:loanApprovalPT"
            operation="approve"
            inputVariable="request"
            outputVariable="approvalInfo">
    <target  linkName="receive-to-approval"/>
    <target  linkName="assess-to-approval"/>
    <source  linkName="approval-to-reply"  />
</invoke>
```

```
        <reply  name="reply"  partnerLink="customer"  portType="apns:loanApprovalPT"
                     operation="approve"  variable="approvalInfo">
            <target  linkName="setMessage-to-reply"/>
            <target  linkName="approval-to-reply"/>
        </reply>
    </flow>
</process>
```

Table 26: BPEL document

B.2.4 Web Service Composite Search Example

Table 27 provides a complete example of a Web Service composite search request:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--Sample XML file generated by XMLSPY v2004 rel. 4 U (http://www.xmlspy.com)-->
<webServiceCompositeSearch xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="C:\thesis\ServiceSearch.xsd">
	<webServiceSearch>
		<functionalCriteria>
			<serviceName>loan service</serviceName>
		</functionalCriteria>
		<qosCriteriaList>
			<qosCriteria>
				<qosName>delivery</qosName>
				<qosCondition>
					<stringCondition>
						<stringValue>http</stringValue>
						<stringComparison>Equals</stringComparison>
					</stringCondition>
				</qosCondition>
				<criteriaLOperator>And</criteriaLOperator>
			</qosCriteria>
			<qosCriteria>
				<qosName>serviceCharge</qosName>
				<qosCondition>
					<numberCondition>
						<number>
							<integerValue>50</integerValue>
						</number>
					    <numberComparison>LessThan</numberComparison>
					</numberCondition>
				</qosCondition>
				<criteriaLOperator>And</criteriaLOperator>
```

```
            </qosCriteria>
        </qosCriteriaList>
    </webServiceSearch>
    <webServiceSearch>
        <functionalCriteria>
            <serviceName>loan   assessment</serviceName>
        </functionalCriteria>
        <qosCriteriaList>
            <qosCriteria>
                <qosName>security</qosName>
                <qosCondition>
                    <stringCondition>
                        <stringValue>encrypted</stringValue>
                        <stringComparison>Equals</stringComparison>
                    </stringCondition>
                </qosCondition>
                <criteriaLOperator>And</criteriaLOperator>
            </qosCriteria>
            <qosCriteria>
                <qosName>AdminFee</qosName>
                <qosCondition>
                    <numberCondition>
                        <number>
                            <integerValue>200</integerValue>
                        </number>
                    <numberComparison>LessThan</numberComparison>
                    </numberCondition>
                </qosCondition>
            </qosCriteria>
        </qosCriteriaList>
    </webServiceSearch>
</webServiceCompositeSearch>
```

Table 27: An example of WSCS