CREATING SPECIFICATION TEMPLATES FOR CLIENT-SERVER FAMILIES IN SERVICE ORIENTED ARCHITECTURE

by

Jaya Bansal

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

ARIZONA STATE UNIVERSITY

August 2009

CREATING SPECIFICATION TEMPLATES FOR CLIENT-SERVER FAMILIES

IN SERVICE ORIENTED ARCHITECTURE

by

Jaya Bansal

has been approved

July 2009

Graduate Supervisory Committee:

Joseph Urban, Co-Chair
Hessam Sarjoughian, Co-Chair
Aviral Shrivastava

ABSTRACT

A service-oriented computing system is a collection of services. SOA presents a way for the providers to provide their services and the consumers to discover those services such that they fit their requirement criteria. The concept of SOA can be applied to distributed applications including client-server architecture, multi-tier architecture and peer-to-peer architecture. This thesis proposes structural specification templates which can be used in development of the above mentioned distributed applications. The templates are evaluated by comparing against the established standards used in the industry. The templates are defined for the abstract structural elements of SOA and do not include behavior and domain specific or implementation specific details The specification templates provide a base for implementing the distributed applications by introducing service orientation in them. In conclusion, this thesis can be used as a basis for the new research into making the process of creating SOA based applications simpler

ACKNOWLEDGEMENTS

First and foremost, I am extremely grateful to my thesis advisor Dr Joseph Urban, Professor, School of Computing and Informatics, Arizona State University. Without Dr Urban’s encouragement and guidance, this thesis would not have been possible. I offer my sincere thanks to Dr Aviral Shrivastava, Dr Hessam Sarjoughian for serving on my committee.

I would also like to thank Ms. Martha Vander Berg, Academic Specialist, School of Computing and Informatics, Arizona State University for helping me throughout the course of my thesis.

I am also thankful to Harneet Sidhana for being there, assisting me during my research work and always supporting me.

I am thankful to my brother who was always there by my side through thick and thin.

Last but not the least; I thank my parents, for always giving me their love and blessings, it is due to them that I stand at this point in my life.

TABLE OF CONTENTS

CHAPTER Page

APPENDIX Page

LIST OF TABLES

LIST OF FIGURES

1. Introduction

Service oriented architecture, also known as SOA, is a relatively new technology. The roots of SOA go back to the technologies such as Java beans, which has been defined by Sun Microsystems as "reusable software components that can be manipulated visually in a builder tool". In essence, although SOA is an emerging technology, it has been around for a few years. However, SOA still shows potential for further research in various aspects of software development.

The concept of SOA enables the applications to be more loosely coupled and interoperable. The loose coupling in SOA is achieved via a network-addressable interface where services act for various functions of an application. SOA has been applied to various applications, the bulk of which includes enterprise applications. The loosely coupled nature of SOA is an advantage for the enterprises implementing service orientation, since it enables them to respond promptly and efficiently to the ever changing needs of the market [8].

1.1. Service Oriented Architecture (SOA)

As defined by OASIS (Organization for the Advancement of Structured Information Standards), SOA is a "*paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. It provides a uniform means to offer, discover, interact with and use capabilities to produce desired effects consistent with measurable preconditions and expectations.*" [2].

1.2. The Key Components

Service is the most essential aspect of SOA and is controlled by three key constituents [3] –

a. Requestor (Consumer) – Requestor requests services;
b. Provider – Provides certain services; and
c. Directory Services – Provides a service repository, which is constantly updated as providers offer new services.

1.3. The Basic Model

The basic idea behind SOA is that a directory service provides a repository of services, which is updated when a service provider publishes its services. A client (also known as consumer or requestor) requests a service (find), which is then matched with the directory and on a positive match a point to point contact is established (bind) between the provider and the client [4]. Figure 1.1 [5] shows the basic model of a service oriented architecture. The figure shows communication between the various constituents in a service oriented architecture. The service provider publishes services on the service broker. The service requestor finds a service on the service broker. Once the right service is found by the consumer, it binds to the provider to utilize the services available on the provider.

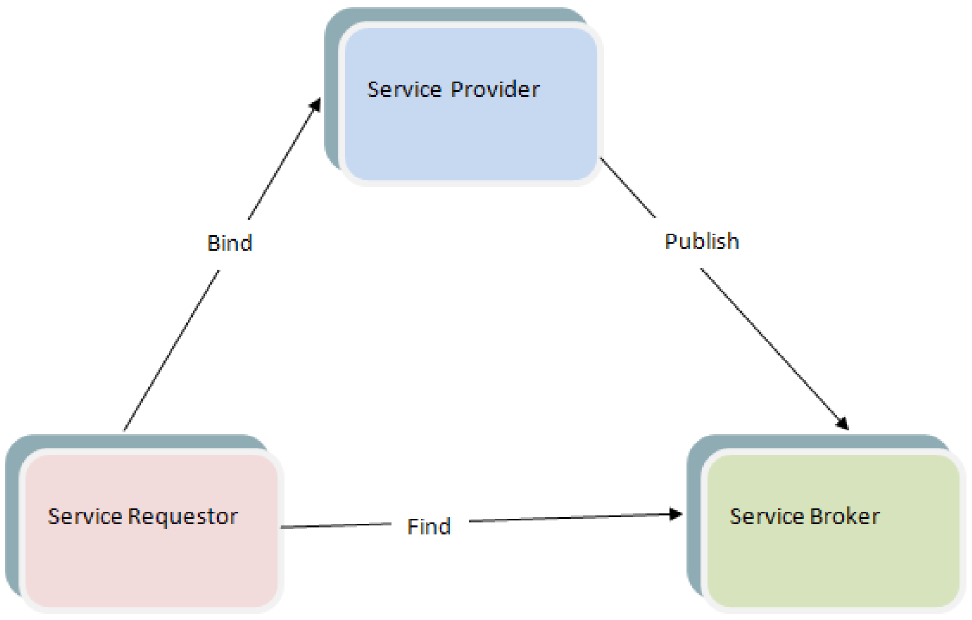


Figure 1.1 Service Oriented Architecture

SOA has three important characteristics [6, 7] -

a. all functions available are defined as services that have invoke-able interfaces;

b. all services defined are independent of each other as well as any external components that may communicate with the services. The services maintain awareness about each other’s existence, but are completely abstract to the external components; and

c. all the interfaces are platform independent and hence can be invoked from any device, running any operating system and using any language.

1.4. Benefits

Service-oriented architecture has many benefits. These benefits are directly concerned with the improvements in the creation of automated solutions. Some of these benefits are listed below [8]–

a. SOA allows the application to interact with other applications with minimum cost and effort required;

b. services in a service oriented architecture are reusable and this property of SOA increases the return on investment that goes into the application; and

c. SOA gives service providers the freedom to build the services the way they want to, while not binding to a particular development or a middleware platform.

1.5. Contribution

The major advantage of SOA lies in the fact that the services are re-usable. Thus, for an effective use of services, it is imperative that there is a clear understanding of the functions provided by each service as well as the environment in which the service operates. A significant portion of the existing research is related to the application of service orientation towards development of specific kinds of applications such as e-commerce applications and hence explains the specifications which are restricted to these applications; resulting in a wide-ranging set of specifications. The variations in the specifications make implementing an environment, where similar services can substitute one another, difficult.

This thesis effort produced a technique to break out of the process of constructing new specifications for every application implemented with SOA. The cycle of creating a new set of specifications can be broken by exploring various existing applications of SOA, to identify various domains that these applications can be categorized into. Once identified, these domains will present a generic implementation model which can be further used to create specification templates. Well-written service specifications will make it possible to improve the testing strategies of SOA based applications.

1.6. Research Objectives

The latest research in the field of software engineering is primarily aimed at minimizing the limitations of fixed requirements obtained upfront in service oriented architecture. The benefits of SOA specified in Section 1.2, have resulted in a large number of organizations that are adopting SOA as a business enabler. The organizations building these applications are focused towards building SOA applications with specific business goals resulting in a disparate set of specifications which are particular to these applications. While the exponential adoption of SOA has broadened the concept of service-orientation by including its application to different aspects of technology, it has also resulted in making the specification process exclusive to every application. Thus, to create a service oriented solution for an application, one would have to develop a new set of specifications every time, making the process of developing specifications cumbersome, time consuming and costly.

The thesis results provide a structural specification template to avoid the cumbersome process of developing a new set of specifications for every service oriented application. The different applications that SOA has been applied to, can be categorized into various domains. The identified domains are then used to create specification guides which can prove to be helpful in implementing service oriented applications.

1.7. Research Methodology

During the course of this thesis research effort, various techniques such as web services, UDDI registry specifications and service description specifications were used to achieve acceptable results. A majority of the work went towards understanding services which form the base of a service oriented solution. In order to gain a better understanding of services, technologies including web services were used along with the understanding of the existing specifications of WSDL [9] and UDDI [10]. The research towards understanding the services was further used to develop a classification criterion which was distinct from that available in the existing literature. To create and then evaluate the templates, existing specification elements such as scope, requirements, and features were adopted. Finally, to evaluate the templates, examples were collected from the existing research results to ensure that the templates can actually be used to generate specifications for service oriented applications.

1.8. Organization

This thesis is organized in the following way. Chapter 2 discusses in detail the different applications of an SOA as well as the proposed classification along with related work in the field; Chapter 3 explains the aspects of SOA which require specification; Chapter 4 discusses the common and uncommon elements of the specification templates; Chapter 5 presents an evaluation, discussion and conclusion of sections of the templates described in Chapter 4. Following the references are appendices which show the examples that have been recreated using the templates.

2. Classification of SOA-based applications

2.1. Overview

Service oriented architecture is an architecture style where the functionality of the application is implemented by using the services. The concept of service orientation provides freedom from hard line connections and minimizes fault points. Since the underlying idea in an SOA is services, all the SOA based applications share similar specifications. By describing a classification criterion, it is possible to create a specification skeleton. The specification skeleton can then be used to make the specification process in the development cycle of an application efficient. The remaining sections of the chapter present the proposed classification criterion, discussion of the related work, and a chapter summary.

2.2. Proposed Classification of SOA-Based Applications

In an SOA environment, the resources on the network are provided as services, such that they can be utilized without the knowledge of their implementation details. SOA-based applications can be categorized into the following client-server families:

- client-server based Service Oriented Architecture;
- multi-tier based Service Oriented Architecture; and
- peer-to-peer based Service Oriented Architecture.

2.3. Client-Server Based Service Oriented Architecture

Client-server is the basic distributed computing architecture and is also known as 2-tier architecture. The client can send requests to the server. The requests are processed by the server and the requested information is sent back to the client [1]. A client-server SOA can be divided into two categories

a. *Primitive client-server service oriented architecture*. The basic client/server architecture consists of the service requestor acting as a client and the service provider acting as the server. Although this architecture does not consist of a registry, it can still be considered SOA-based because the services described here are discoverable. The services have published definition which makes them discoverable. The service requestor can invoke the service directly from the service provider. The requestor, instead of searching for a service through the registry, can store the service endpoint in its own code. The provider's access point can also be stored in a configuration file on the requestor system, so that it can be easily changed without re-compiling the entire application [11]. Figure 2.1 shows a simple diagram of a primitive client-server SOA. A client can invoke services from a service provider. Note that the diagram does not show a registry since the endpoint of the provider service is known to the requestor.

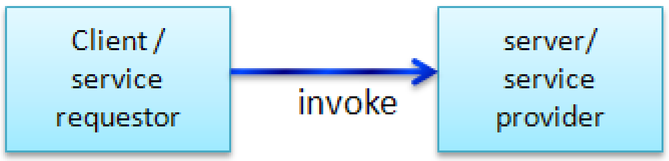


Figure 2.1 Primitive Client-server SOA

b. *Client-server SOA with Registry*. This architecture is a variant of the primitive client-server SOA with the addition of a registry. The client-server SOA is the basic service oriented architecture in the true sense since it makes use of a service registry.

Figure 2.2 shows a diagram of a client/server architecture including a service registry. As seen in the diagram, the client (service requestor) places a request with a service registry which in turn finds suitable matches for the request. The client chooses a server (service provider) and establishes contact with it in order to request its services.

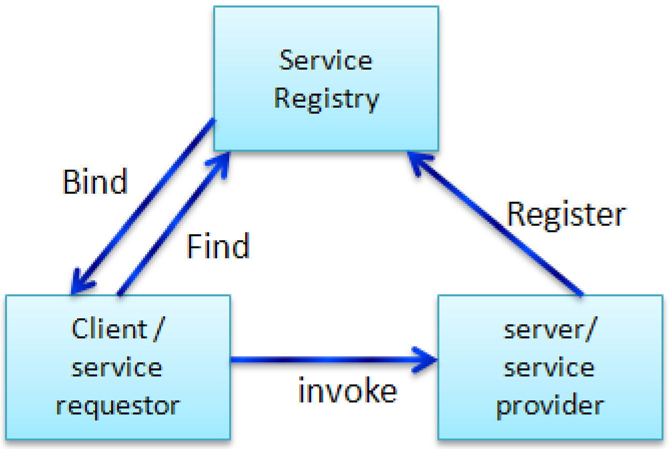


Figure 2.2 Client-server SOA with registry

2.4. Multi-Tier Based Service Oriented architecture

Multi-tier architecture refers to an architecture consisting of three or more layers [7]. At the basic level, the layers include:

a. **data layer –**the data layer is the layer where all the data is stored for the application;

b. **business layer –**business layer is the layer which consists of the processes required to realize the business rules in an application, for example, transfer money from one account into another account; and

c. **presentation layer –** presentation layer is the layer which presents an interface for the users to communicate with the application.

The layers mentioned above, together form the three-tier architecture. Multi-tier architecture can be viewed as a service oriented architecture. The presentation layer can be considered a consumer since it presents an interface for an external user to interact with the system. The business layer can be considered a registry, since it acts as a middleware such that it provides information for a consumer to connect to the publisher. The data layer can be considered a service provider since it offers access to its service, which may not be restricted to data, to be utilized by a consumer. When further separated into sub-layers the same architecture transforms into a multi-tier architecture. For example the three-tier structure mentioned above can be converted to four tiers simply by dividing the data layer into data access layer and data layer. As shown in Figure 2.3, the data layer consists of a storage layer which physically stores all the data and a data access layer which consists of functions to access and modify the data in the data layer. Separation of the data layer into data and data access layers introduces modularity in the application, which in turn makes it loosely coupled [12].

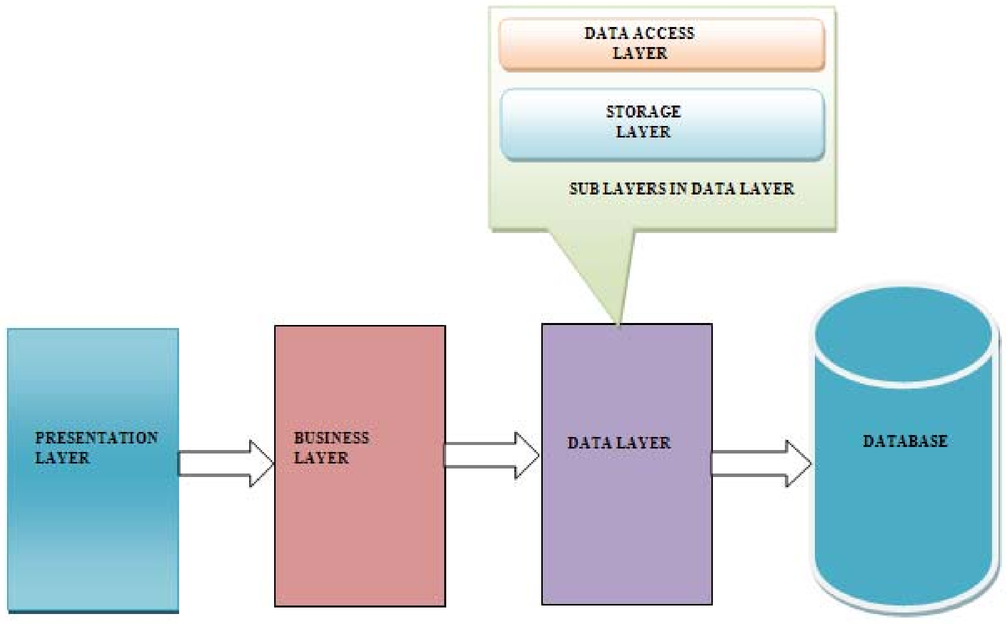


Figure 2.3 Multi-tier architecture

2.5. Peer-to-Peer Based Service Oriented Architecture

A peer-to-peer architecture is a distributed computing architecture where the notion of a client and a server is not tied to a particular system. All systems (nodes) participating in a traditional peer-to-peer architecture are known as “peers”. Every node can act as a client or a server [1]. In a peer-to-peer service oriented architecture however, nodes are essentially services. Every service can function as [13]–

a. **client** – a service can act as a client in which it queries the registry to find a service which is suitable for the required tasks. A client is a service requestor.

b. **server** – a service can act as a server when required, that is, the service can act as a suitable candidate to serve another service; and

c. **registry** – a service can also act as a registry such that other services can search for candidates for service.

In an SOA-based application, one system acts as a registry. The provider services publish their service description with the registry, so that they can be discovered by consumer services. Peer-to-peer SOA, however, is different in the sense that it lacks the presence of a central registry (see Figure 2.4 below). Every peer service has the ability to act as a registry. The peer service can maintain its own description content so that when another peer queries, the search criteria is matched against every peer's self published description and returned in case of a match [14].

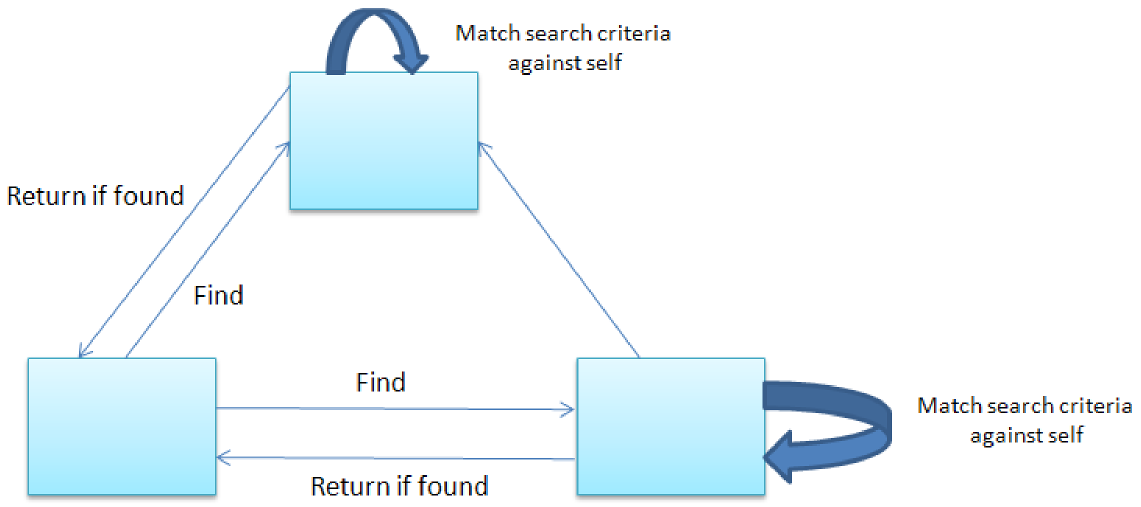

Match search criteria against self
Return if found
Find
Find
Return if found
Match search criteria against self

Figure 2.4 Peer-to-peer SOA

2.6. Related Work

SOA is one of the latest technologies being explored and has a growing potential for research in every aspect of software life cycle. Most of the existing research has been towards implementing different kinds of applications using the concept of service orientation. The research presented in [15] has described a classification criteria for service oriented applications based upon the architecture style of the applications. The paper further describes different applications in each domain. Considerable research has been done towards the service orientation of peer-to-peer architectures. Garofalakis et al. [16] Briefly introduces an idea of a peer-to-peer registry for service discovery in their paper and [17] strongly emphasizes on decentralization of the registry in a peer-to-peer network. Maheshwari et al. [18] mentions the JXTA technology which is Sun Microsystems' proposal for a peer-to-peer based application with service orientation. The closest implementation of a multi-tiered service oriented system can be seen in an e-business or an e-commerce system which uses the enterprise bus service. The enterprise bus service is implemented as middleware for communication between various services [19]. The basic functionality of an enterprise bus service includes routing, messaging and mediation. The most closely related research in this area has been provided in [20] where the researchers describe a generic conceptual model of a service oriented application. Most of the research has resulted in a new specification language/process leading to a diverse and varied specification process and less into classifying the SOA based applications such that the services implemented by one are reusable by others.

2.7. Summary

The described classification forms the basis for this thesis. Although the above mentioned classification cannot be used to generalize the specification of SOA based applications to the entirety (since applications differ in their functionality), the templates created (on the basis of the classification) in the following chapters can be used to make the specification process simpler. The next chapter describes the various aspects of SOA that need to be specified in the application in order to successfully implement the application.

3. Specification elements

3.1. Overview

This chapter introduces the various elements that have been included in the template outline in the later chapters. In order to correctly design a service oriented system, it is essential to understand the different elements that make an SOA application service oriented. The rest of the chapter describes various elements that need to be addressed when constructing a service oriented system.

3.2. Elements of Specification Templates

In any SOA based application, there are various aspects that need to be addressed in order to create a service oriented system. The diagram below (Figure 3.1) depicts the various components of service oriented architecture which must be focused on for implementing an SOA application [21].

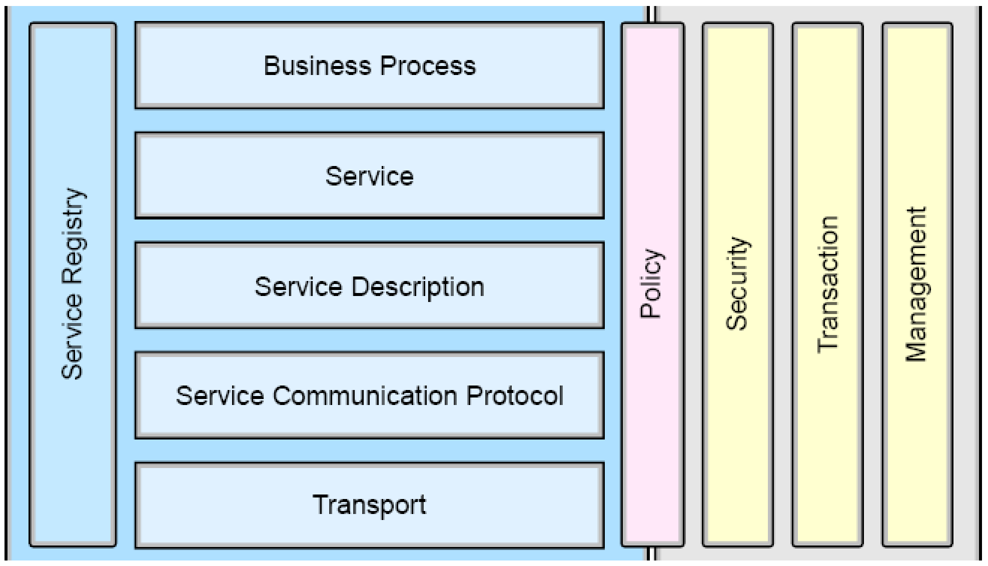


Figure 3.1 Components of an SOA [21]

The following sections provide definitions of the various components that are supported in this thesis and also depicted in Figure 3.1. The components including Business Process, Service Communication Protocol, Transport, Policy, Security, Transaction and Management and are not supported in the thesis.

Service

A service is a collection of functions in an SOA based system. In order to implement business processes in a system, services need to be implemented. Once the service parameters are known to the client, the client can bind to the server and send requests to it. The service parameters are published by the server as "Service Description".

Service Description

An SOA based application is constructed with services performing all the tasks. In order to achieve loose coupling in an SOA based application, it is important that a service is discoverable. Once discovered, the service can be used to fulfill certain tasks it promises to do. To be discoverable, a service must publish its own description. Apart from the functionality, a service description consists of various service parameters which a client might need in order to access the functions offered by the service.

The service description provides the key to establishing a connection between a service provider and consumer.

The description of a service may consist of multiple documents including–

a. description document -- The service description that describes the parameters for establishing the connection between the provider and consumer consists of the following components –

    i. operation – a list of the functions that form the interface of the service, where the functions also have corresponding input and output methods;

    ii. binding – binding in the service description describes the transport technology used by the service to communicate and binding technology may vary by operation or could be specific to the entire service; and

    iii. access point – port denotes the physical address at which the service can be accessed with a specific protocol;

b. communication policy – the policy is an optional component that consists of a document with the policies that will need to be followed in order to communicate with the service;

c. SLA (Service level Agreement) -- service level agreement consists of the following aspects [21, 22]

    i. performance – specifies the expected performance requirements of the services;

    ii. security –specifies the details of how services are secured; and

iii. reliability – specifies the expected reliability of delivery of messages.

Service Registry --

In service oriented architecture, the services need to be published and discovered with a registry in order to be utilized. The registry stores the service specification in its repository. The specification includes the service description documents (as explained above).

Registries can be of various types ranging from public registries to private registries to intranet based registries [23] –

a. *public registry:* public registries, open to any service, are available to all the services to register themselves and can be discovered by other services; and

b. *private Registry:* private registry is available to the services within an organization (to register) in order to be discovered.

Service discovery is an important aspect of SOA since the definition of SOA relies on service publication and service discovery. In order to access a service, the service consumer needs to know the location of the service. A service discovery essentially is the process of locating useful services and their endpoints, which are communication ports where the service can be contacted. Service discovery comprises of two main processes [24] –

a. ***Registration Process*** **–** For any service to present itself to the service consumers, it is important for that service to go through the registration process. A service registers itself with a registry. The registry acts like a directory and provides consumers with information regarding various services registered with it. The registration is deleted from a registry once the service is removed [24].

b. ***Look up Process*** **–** Just as a service registers itself to be of use to its consumers, a consumer also needs to be able to search the service. A consumer can query the registry for a particular search criterion. The registry finds all the services that match the criterion and returns a list back to the consumer. The consumer can then choose from the service and bind to it.

The registry is used by services to find other services hence the registry must store information about other services. The only information stored in the registry would be that required to connect to the "discoverable" service. The information must consist of at least the following parts –

a. The description of the discoverable service so that the service consumer can decide whether the service is for their use or not.

b. Once a consumer has discovered the service that they want to use, in order for them to access the provider service, they need the binding information such as a port number.

c. To enhance the searching experience, the registry is capable of returning search results by category; hence every service also has an associated category stored along with its description.

d. Along with the above mentioned parts, it is essential to specify the transport model followed by the provider’s service since the transport criteria is important while searching for a website.

The UDDI service registry specification describes the following structure to store the service information in the registry [25] -

a. tModel facet – A tModel facet consists of a pointer to the service specification document;

b. binding template – A binding template consists information about accessing a service and is associated with a tModel facet;

c. business service – A business service consists of name and description of a service. It is associated with at least one binding template;

d. business entity – A business entity has one or more business services associated with it.

Figure 3.2 depicts the relationship of the four aspects of a registry data structure as specified in UDDI, an industry adopted standard [25].

As shown in Figure 3.2, every business entity has one or more business services associated with it. Every business service, in order to be discovered, requires a binding template. One business service may have multiple binding templates associated with it. Every binding template consists of a tmodel facet [26] which consists of a pointer to the actual service specification document [27, 28]

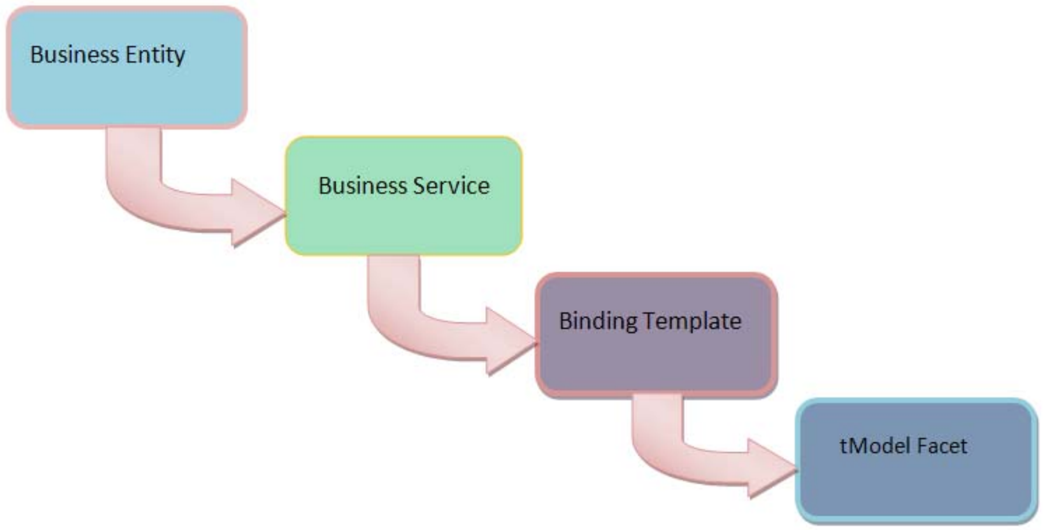


Figure 3.2 Registry structure in UDDI

3.3. Summary

The chapter described the different elements that constitute an SOA based application. The service description including protocol, interface, policy, and service level agreement; along with the registry description including data structure used, registration, service updates and executing queries addresses the basic SOA application. The following chapters use the aspects service registry and service description to create specification template for a peer-to-peer based SOA which essentially applies to each domain described in the classification criterion comprising of client-server based SOA, multi-tier based SOA and peer-to-peer based SOA.

Specification Template Outline

4.1. Overview

This chapter defines the different elements of the design specification template which can be used to realize a service oriented application. Section 4.2 describes the basic components of a service oriented application as defined in Section 3.2. Section 4.3 describes the elements of the template that apply to a specific category of applications. Section 4.4 shows the template with the elements defined below. The template also points out the various possible restrictions that may be placed on some of these elements.
The following sections describe and define the different elements of the specification template followed by a chapter summary.

4.2. Common elements

The following is an outline of the common elements involved in the proposed design specification document.

1. **Participants** – An SOA based application has the following participants –
    a. Service Provider – Service provider or a server is the participant which provides the service to the consumer. The service provider is responsible for providing the client with a service description so that the client can bind to the server in order to use the service.

The server will provide the client with a service description document which will consist of the information related to the services and the functionality provided by the services.

b. Service Consumer – The service consumer or a client is the participant which consumes the service provided by the provider/server.

2. **Service –** The service component of the template comprises of the different parameters that describe a service such that once the service parameters are known to the client, the client can bind to the server and send requests to it. The service component of the template is similar to the WSDL in the sense that it does not describe the provider node, rather it provides information about services offered by the provider node. The following are the various parameters which are described in a service description –
    a. ***Name***: Name indicates the name of the service by which the service can be identified.
    b. ***Description***: The description parameter is a natural language description of the service's functionality.
    c. ***Functions***: The role of a service in a SOA-based application is to provide the necessary functionality to realize the system. In order to achieve the desired functionality, a service, apart from the configuration related parameters, also consists of functions which allow a consumer to interact with the service. The functions defined in a service provide the functionality expected from a service. Every function in a service includes the following sub parameters –

i. *Name*: The name of the function indicates the function name by which it can be identified

ii. *Description*: The description of the function includes a brief description of the function in human readable form.

iii. *Input*: The input of the function includes a list of the all the input parameters that may be required in order to obtain the desired output. This information is especially important when a different application wants to access the service. The application can use the parameters to define the calling functions to operate easily.

iv. *Output*: The output of a function includes a list of all the output parameters that are returned by the function. This information is also important for application which wants to use the service. The application accessing the service can use this information to receive the output and store it in appropriate variables.

4.3. Uncommon elements

The following is a list of the elements of the specification template which are not used in all SOA based applications. These elements may be specific to special applications like those including a service registry or those implemented in a peer-to-peer environment. Service registry can be considered an uncommon element of the specification template. The reason is that the definition of SOA requires the services used to be discoverable. The discoverability aspect of the service is addressed by the “service” element of the template, defined in Section 4.2. The registry on the other hand can be optional in a situation where the service consumer knows the location of the service provider before hand. However, if the same service provider was put in a scenario where the registry prevails, it can be discovered because of the service component defined in Section 4.2.

1. **Registry** -- The registry component of the template consists of the information that a registry stores about a service so that it can be discovered. The registry component of the template is similar to the UDDI. It does not define a registry node but the data that a registry keeps for the services that are published with it. The following is the list of data that registry stores for a service.

Service Information:

a. *Service Name*: A descriptive name of the service.

b. *Service Description*: A description of the service that registers with the registry.

c. *Access Point:* Access information in order to access the service which could be email address, URL, or any similar address type.

d. *Category:* Category field is an optional field which helps in narrowing the search results in the registry.

e. *Reference Pointer:* A link (could be in the form of a URL) to the specification document/interface for the service (a document with technical description of the service including method signatures).

2. **Peer Node**: Peer node element is used in an SOA based peer-to-peer application. Every consumer maintains a list of all the peer nodes which provide services of interest to the consumers. The list is used when searching for a service. The query with a service description can be broadcast to all the

nodes available in the list and the nodes return the result back to the consumer peer.

a. Peer Information: Every peer stores the following information about other peer nodes that offer services useful to the peer.

   i. *Peer Id:* Id of the peer node. The id can be used to keep track of the nodes in the list of peers maintained by the consumer.

   ii. *Peer Name:* Name of the peer node in the list of peers.

   iii. *Peer Access Address:* Access address where the peer node can be contacted.

b. Peer functions:

   i. Name: AddPeer()

      Description: The AddPeer function is used to add a peer node to the list of useful peer nodes maintained by a peer.

      Input: Peer Information

         PeerName: String

         PeerAccess: String

      Output: PeerId: string

   ii. Name: RemovePeer

      Description: The RemovePeer function is used to remove a peer node from the list of the nodes maintained by a peer.

      Input: PeerId: string

      Output: None

   iii. Name:SearchQuery

Description: The searchQuery function is used to search the peer data structure a particular service matching search criteria. The searchQuery function returns the location to the description document of the service that may be of use to the querying node.

Input: searchCriteria: String

Output: serviceDescriptionLocation:String

4.4. Template

The template below brings together the different elements of a SOA-based application, whether common or uncommon. The template fits directly with a peer-to-peer application scenario, implying that the peer-to-peer application template is essentially the superset of the templates for other architectures defined in Chapters 2 and 3.

1.**Key Participants**

A peer-to-peer SOA based application can only have one participant which can take up different roles at different times during execution. The various roles played by a peer/ node are listed below

i. Service Provider – A peer node may or may not provide services to consumers. When a peer acts as a service provider, it maintains a record of the services that it provides to the consumers. In other words a peer, when serving as a service provider, also serves as a registry.

ii. Service Consumer – The peer node may or may not be a consumer. A node which serves solely as a consumer does not need to keep a record of any services since it does not offer any. However, in case a peer acts as both a consumer and a provider, it will also serve as a registry and keep a record of all the services that are provided by the peer.

2. **Services**

< The service component of the template comprises of the different parameters that describe a service such that once the service parameters are known to the client, the client can bind to the server and send requests to it. The service component focuses on the service provider more than the service consumer The service component of the template is similar to the WSDL

<Service 1>

Name: <Name of the service>

Description: <Description of the service in human readable form>

Function(s):

<Function 1>[1]

Name: <Name of the Function.>[2]

Description: <Description of the Function in human readable form.>

Input Name:

<Input name 1> : <input name as defined under message types>

…

<Input name n>

Output Name:

[1] < > (brackets) with bold text, followed by a number denote the mutable sub headings. The sub headings are followed by a number since there could be more than one sub heading. n denotes a positive integer greater than 1.

[2] < > (brackets) with regular text denote the description of the heading it is placed next to.

<Output name> : <Output name as defined under message types>

<Function 2>

…

<Function n>

3. Registry

< The registry component of the template consists of the information that a registry stores about a service so that it can be discovered. The registry component of the template is similar to the UDDI. It does not define a registry node but the data that a registry keeps for the services that are published with it. The following is the list of data that registry stores for a service.>

Service(s) Information: *< An organization may contain provide a single service or multiple services.>*

<Service 1>

Service Name**:** <Descriptive name of the service offered by the organization mentioned above.>

Service Description**:** <Description of the service.>

Access Point: <Access information in order to access the service- could be email address, URL or any similar address type.>

Category: <Category field is an optional field which helps in narrowing the search results in the registry.>

Reference Pointer: <A link (could be in the form of a URL) to the specification document/ interface for the service (a document with technical description of the service including method signatures).>

<Service 2>

…

<Service n>

4. Peer Node

Peer Information Structure

<Every consumer maintains a list of all the peer nodes which provide services. The list is used when searching for a service. The query with a service description can be broadcasted to all the nodes available in the list.>

<Peer 1>

Peer Id**:** <Id of the peer node. The id can be used to keep a track of the nodes in the list.>

Peer Name**:** <Name of the peer node.>

Peer Access Address**:** <Access address where the peer node can be contacted.>

<Peer 2>

…

<Peer n>

Peer Functions

i. Name: AddPeer()

Description: The AddPeer function is used to add a peer node to the list of useful peer nodes maintained by a peer.

Input: Peer Information

PeerName: String

PeerAccess: String

Output: PeerId: string

ii. Name: RemovePeer

Description: The RemovePeer function is used to remove a peer node from the list of the nodes maintained by a peer.

Input: PeerId: string

Output: None

iii. Name:SearchQuery

Description: The searchQuery function is used to search the peer data structure a particular service matching search criteria. The searchQuery function returns the location to the description document of the service that may be of use to the querying node.

Input: searchCriteria: String

Output: serviceDescriptionLocation:String

4.5. Summary

The specification elements defined above apply to the various client server families described in Chapter 3. As discussed in Sarjoughian et al.[33], an SOA based application has two aspects – static and dynamic. The template outline accounts for static aspects which are service, service description and messages but does not account for the dynamic aspects like communication agreement, message framework and discovery. The next chapter shows the evaluation procedure used to assess the templates, followed by a short discussion and conclusion.

5. Evaluation and Conclusion

5.1. Overview

This chapter presents an evaluation of the structural templates constructed in the previous chapters. The evaluation section describes the methodology followed to evaluate the templates created as a part of the thesis research. Following the evaluation is a short discussion of the observations and a conclusion section which summarizes the work done in the thesis and discusses the possible extensions to the thesis research results.

5.2. Evaluation Methodology

The structural templates created in the previous chapters can be used to design SOA-based applications for distributed architectures. The template outline described in this thesis as a whole applies to a peer-to-peer architecture and can be modified to fit other types of distributed architectures. The template for peer-to-peer architecture consists of all the elements including the service descriptions and the registry and thus can be used for the evaluation purpose as opposed to evaluating the different templates for different architectures. The template constitutes of different elements of an SOA based application such as service description and registry. In order to evaluate the templates, it is important to look at the different elements mentioned above, i.e. service description and registry, individually. By comparing the elements to existing standards, we can establish the legitimacy of each aspect and eventually, the whole template. The following is the evaluation procedure used to validate the templates.

1. Map the components of the template to the existing standards and concepts.

2. Select an application example from the existing literature.
3. Using the template, create service descriptions.
4. Compare the service description specified in the literature with the one created using the template.

1. Map the components of the template – The first step in the evaluation process is mapping the components of the template with the existing standards and concepts. The template consists of the following components –
   a. Service – the service component of the template is the description of the service and includes the following pieces
      i. Name of the service
      ii. Description of the service
      iii. Functions
         a. Input to the function
         b. Output of the function

   The service component of the template comprises of the elements that can describe a service such that if published, the service can be discovered. In the SOA-based applications implemented as web-services, WSDL is used as a standard to create service descriptions. The service component defined in the template can be mapped to the abstract components of WSDL. The abstract components of WSDL are non implementation specific. Furthermore, itdoes not provide behavioral specification. The abstract components of WSDL are as follows [32] --

i. Messages: The message component of a WSDL document lists all the messages exchanged between the service providers and consumers for a particular function. Every function has an input message and an output message.

ii. Operation: An operation defines the interaction between a service provider and a service consumer. The operation also identifies different messages exchanged during the interaction. A consumer can utilize the functionality provided by a service with the help of operations defined in the service.

iii. Interface: The interface groups the messages and operations together. The interface component of WSDL is similar to the service component as a whole. Just like the "service" component, interface component encompasses the functions (operations) offered by a service and the messages that are used to interact with those functions.

The mapping of the service component of the template to WSDL is shown in Table 5.1

Service component	**Maps To**	**WSDL Component**
Input/ Output for a function	→	Messages -- Every function has an input message and an output message
Function names – the service consumer can access the service provider by using functions.	→	Operation names – the interaction between a service provider and a service consumer
Service components groups the functions and input/ outputs together	→	Interface – The interface groups the messages and operations together

Table 5.1 Mapping of the template to WSDL

d. Registry – The registry component of the template comprises of the information that a registry must store in order for services to be discovered when published. The registry consists of the following information (applicable to every service registered with the registry)

i. Service Name: name of the service

ii. Description: description of the service

iii. Access Point: access point is the service access URL

iv. Category: Category describes the category to which a service may belong in order to speed up the look-up process.

v. Reference Pointer – URL of the service description document.

UDDI is the registry standard for SOA [25]. The following form the structure of UDDI [28] --

Business Entity – business entity consists of the information about the organization publishing the service.

Business Services – business service is used to describe the information about family of services.

Binding template – binding template consists of the following required information about services.

i. Access point – The access point is used to access a web service (could be a URL or an email address).

ii. Overview URL – This element contains a reference to the service implementation document. The overview URL is an optional element that only provides access to human readable documents.

iii. tModel instance – The tModel instance comprises of the following attributes –

a. name: consists of the target namespace for the service interface document.

b. Description: Description element consists of the documentation related to the service.

c. Overview URL – overview URL points to the service interface document.

d. Category Bag – Category bag consists of keyed reference, key name, and key value. These elements indicate the intended business use for this service description.

The mapping of the registry component of the template to UDDI is as follows –

Registry component	**Maps To**	**UDDI component**
Service Name	→	Business service
Service Description	→	Business service
Access Point	→	Binding Template/ Access Point
Category	→	tModel instance/ category bag
Reference Pointer	→	tModel instance/ overview URL

Table 5.2 Mapping of registry components of the template to UDDI

e. Peer Node: The concept of peer node as described in the template is such that the peer node keeps track of all the peer nodes in its system, adding them or deleting them as necessary. The peer node maintains a list of the peers providing publications that may be useful to the peer. When searching for a service, the peer broadcasts the query to all the peers in its list. All the peers return the result of the query back to the querying peer. A similar concept of peer has been described by Hasselmeyer in [24] and Papazoglou et al. in [29]. Hasselmeyer explains that most of the Peer-to-peer systems implement "flooding search" method. The peer sends the query to all the peers providing services. Service providers act like their own registries and also allow the providers to register and unregister.

Papazoglou et al.[29] describe a federated architecture for P2P Web-Service such that there is an event notification service which keeps track of all the new peers. Every peer maintains a list of all the peers that provide publications matching its subscription query. The peer list is chosen by the event notification service.

The peer concept described in the template can thus be mapped to the peer concept described in the above mentioned published articles.

2. Select application example from the existing literature – The examples to demonstrate the templates have been chosen from the sources mentioned below. The detailed document created from the templates for each of the examples is available in Appendix A.
    a. Basic client-server architecture – The example for basic client server architecture is adopted from an article published by National E-health Transition Authority (NEHTA) [30]. The article describes the example comprising of a discharge sender service and a discharge receiver service. The discharge summary sender generates the discharge summary and

sends it to the receiver service. The receiver service returns the status of the discharge summary whenever requested. The status could be

1– Acknowledged
2 – Pending
3 – Not received

b. Client server architecture with registry – the example for the client server architecture with registry has been used from the UDDI best practices document [31]. The example consists of a stock quote service and a registry. The stock quote service returns the trade price of a stock, given its ticker symbol. The registry consists of the description of the service so that another service can query the registry to retrieve the access information for the stock quote service.

i. Multi-tier application – the example for multi-tier architecture was used from the book published by Erl [8]. The book explains an example of a timesheet application which comprises of multiple services.

The various services involved in the application together perform the following tasks

i. Employee submits the timesheet

ii. Get recorded hours for the customer

iii. Get the billed hours for the customer

iv. Compare the recorded hours and billed hours

v. Confirm authorization

vi. Get weekly hours limit

vii. Compare weekly hours limit with the hours recorded

viii. Update the employee history if the timesheet is rejected

ix. Send a notification of rejection to the employee and their manager

c. Peer-to-peer architecture – The example for peer-to-peer architecture is used from Papazoglou et al. [29]. The article describes the concept of a peer-to-peer architecture for an e-travelling system. The application works in the following way –

i. When a new peer comes in, it publishes its services on the registry

ii. The super-peer, which is an event notification service, receives a request from the new peer.

iii. The super-peer matches the published information against all the subscriptions that may find the new peer's services useful.

iv. In an event, a match is detected; the super-peer forwards the publication information of the new peer to the matched peers.

v. Super-peer also matches the subscription details of the new peer with its existing publications.

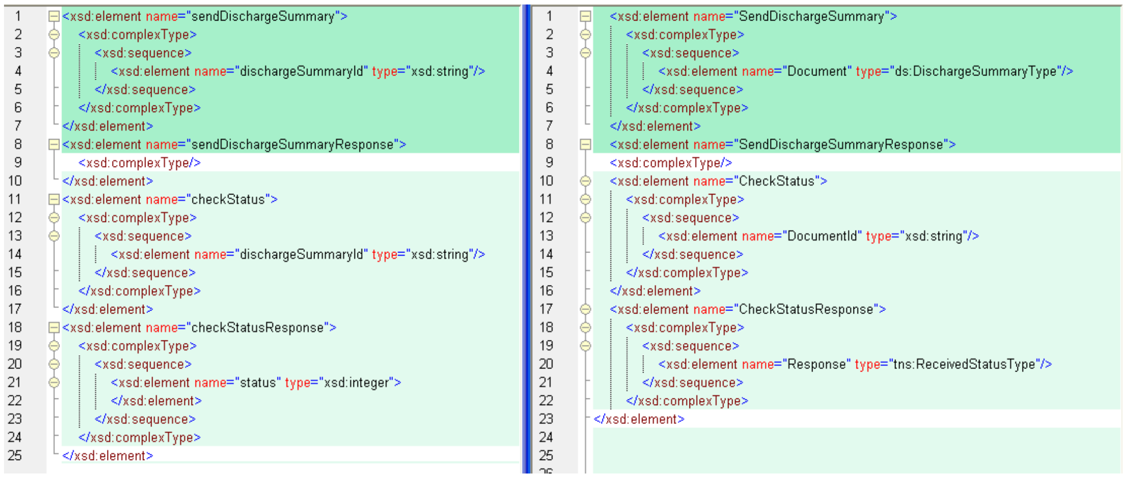
vi. Once a match is detected, the super-peer forwards the information of the all the peers offering the services matching the subscription needs to the new peer.

vii. Thus, all the peers hold the information regarding the peers that offer services that match the needs of the subscribed peers.

viii. Each peer issues a request to all the peers in its list.

ix. All the peers send the requesting peer their service description matching the request.

3. Using the template create the service descriptions – the service descriptions for the examples can be found in Appendix B.
4. Compare the service description specified in the literature with the one created using the template – Instead of comparing the entire document, different elements of the service descriptions from the template are compared against the ones existing in the literature for the same examples.
    a. Comparing the service description elements – Presented below is a comparison of the dischargeSummarySender and dischargeSummaryReceiver services from NEHTA with the ones created from the template. First the input types are compared between the two as shown below. The left screen shows the inputs from the services created using the template and the right screen shows the inputs and outputs used by the services in the NEHTA example. As seen below, there are some differences between the two. The input types and names are different in the two cases shown below. NEHTA uses a "document" type input for sending discharge summaries whereas the example created using the template used an "id" to send to the receiver.

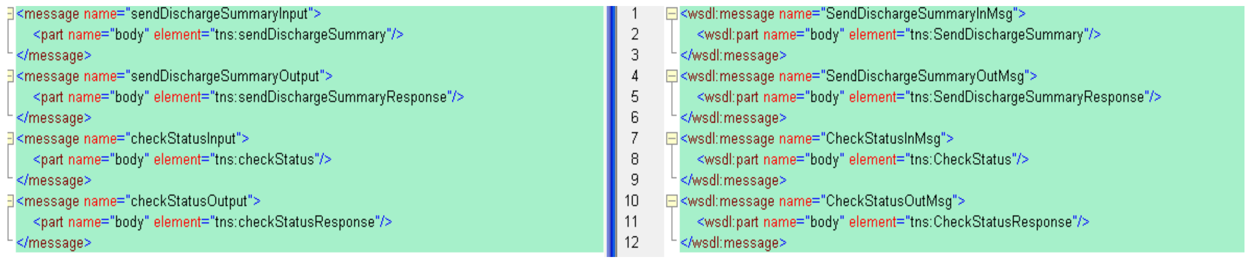


```
<xsd:element name="sendDischargeSummary">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element name="dischargeSummaryId" type="xsd:string"/>
      </xsd:sequence>
   </xsd:complexType>
</xsd:element>
<xsd:element name="sendDischargeSummaryResponse">
   <xsd:complexType/>
</xsd:element>
<xsd:element name="checkStatus">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element name="dischargeSummaryId" type="xsd:string"/>
      </xsd:sequence>
   </xsd:complexType>
</xsd:element>
<xsd:element name="checkStatusResponse">
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element name="status" type="xsd:integer">
         </xsd:element>
      </xsd:sequence>
   </xsd:complexType>
</xsd:element>
```

```
   <xsd:element name="SendDischargeSummary">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element name="Document" type="ds:DischargeSummaryType"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="SendDischargeSummaryResponse">
   <xsd:complexType/>
   <xsd:element name="CheckStatus">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element name="DocumentId" type="xsd:string"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="CheckStatusResponse">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element name="Response" type="tns:ReceivedStatusType"/>
         </xsd:sequence>
      </xsd:complexType>
</xsd:element>
```

Figure 5.1 Comparison of the input elements of the service descriptions

The following shows a comparison of the message part of the service descriptions. The left screen shows messages from the service descriptions created using the template and the right screen shows the messages from NEHTA example.



```
<message name="sendDischargeSummaryInput">
   <part name="body" element="tns:sendDischargeSummary"/>
</message>
<message name="sendDischargeSummaryOutput">
   <part name="body" element="tns:sendDischargeSummaryResponse"/>
</message>
<message name="checkStatusInput">
   <part name="body" element="tns:checkStatus"/>
</message>
<message name="checkStatusOutput">
   <part name="body" element="tns:checkStatusResponse"/>
</message>
```

```
<wsdl:message name="SendDischargeSummaryInMsg">
   <wsdl:part name="body" element="tns:SendDischargeSummary"/>
</wsdl:message>
<wsdl:message name="SendDischargeSummaryOutMsg">
   <wsdl:part name="body" element="tns:SendDischargeSummaryResponse"/>
</wsdl:message>
<wsdl:message name="CheckStatusInMsg">
   <wsdl:part name="body" element="tns:CheckStatus"/>
</wsdl:message>
<wsdl:message name="CheckStatusOutMsg">
   <wsdl:part name="body" element="tns:CheckStatusResponse"/>
</wsdl:message>
```

Figure 5.2 Comparison of the message element of the service descriptions

Similar comparisons were done between the services description documents created using the templates and the documents present in the literature. The results of the comparison were the same that is, barring element names and binding information (which was not included in the template outline), the components of template generated service descriptions matched with those of the pre-existing service descriptions.

The service descriptions for which the comparisons were done are

i. Stockquote.wsdl

ii. Employee.wsdl

iii. timesheetsubmission.wsdl

iv. Notification.wsdl

b. Comparison of the service registry components

Registry specification as shown in the UDDI example [31] –

```
<tModel authorizedName="..." operator="..." tModelKey="...">
        <name>StockQuote Service</name>
        <description xml:lang="en">
                WSDL description of a standard stock quote service interface
        </description>
        <overviewDoc>
                <description xml:lang="en">WSDL source document.
                </description>
                <overviewURL>
                        http://stockquote-definitions/stq.wsdl
                </overviewURL>
        </overviewDoc>
        <categoryBag>
                <keyedReference tModelKey="uuid:C1ACF26D-9672-4404-
                9D70-39B756E62AB4"
                keyName="uddi-org:types"
                keyValue="wsdlSpec"/>
        </categoryBag>
    </tModel>
```

Registry specification according to the template is as follows

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs=http://www.w3.org/2001/XMLSchema
elementFormDefault="qualified" attributeFormDefault="qualified">
<xs:complexType name="registry">
        <xs:annotation>
                <xs:documentation>registry service
                specification</xs:documentation>
        </xs:annotation>
        <xs:sequence>
                <xs:element name="stockquote">
                        <xs:complexType>
                                <xs:sequence>
                                         <xs:element name="Description"
                                         type="xs:string"/>
                                        <xs:element name="AccessPoint">
                                                <xs:complexType>
                                                        <xs:sequence>
                                                                <xs:element
                                                        ref="http://www.example.co
                                                        m/stockquote"/>
                                                        </xs:sequence>
                                                </xs:complexType>
                                         </xs:element>
                        <xs:element name="ReferencePointer">
                            <xs:complexType>
                                <xs:sequence>
                                    <xs:element
                                    ref="http://www.exa
                                    mple.com/stockquote.
                                    wsdl"/>
                                </xs:sequence>
                            </xs:complexType>
                        </xs:element>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:schema>
```

Upon comparing the two registry specification documents shown above, it can be seen that the documents do not match exactly. However, the elements of the two specifications are similar. For example, both the documents contain name of the service, access point of the service, and the reference point where the service description documents are available. The first document conforms to the UDDI standard whereas the second document was generated using the template.

The above mentioned steps are explained in detail along with the application of templates to the examples in appendix A.

5.3. Discussion

Evaluation of the templates show that it can be mapped to the industry followed standards for registry and service descriptions. The mapping tables show that all the components of the template can be mapped to the standards although not all the components of the standards can be mapped to the template's components. That is, for service description, all the abstract elements of the template map to the abstract elements of WSDL (the service description standard). The other elements of WSDL do not map to the template's elements because they are implementation specific. The template also differs from WSDL with respect to behavioral aspects. The template accounts for static elements but not the dynamic elements like message framework which in turn address the behavioral aspects of SOA. In case of service registry, all the elements of the registry specification from the template, map to those of the UDDI (registry standard). The services and registry section of the template have been compared against the existing standards. As shown in Section 5.3, most of the elements of the template can be understood in comparison with the abstract elements of WSDL [32], which are Messages, Operations and Interface. Dhesiaseelan [32] also discusses the implementation elements of WSDL which describe aspects of services which can be used to access them. The implementation elements of WSDL consist of the underlying transport protocol and end points like an address URI. Due to time constraints, the template described in the thesis does not include the implementation aspects of services but can be extended to include elements similar to the implementation aspects of WSDL.

The final element of the specification template is the peer node. The peer node described in the template maps to the peer node concept described in Hasselmeyer, 2005 [24] and Papazoglou et al., 2003 [29]. The peer code concept as described by Hasselmeyer and Papazoglou et al. is that of a node which holds the information of all the peers offering services matching its interest. A similar concept of the peer node is defined in the template where the node holds a structure for storing information pertaining to all the neighboring nodes of interest. The peer node in the template has the ability to update its list of nodes and conduct queries on its own service when queried.

The comparison of the template with the existing service descriptions was not a 100% success. There are various reasons for the non matching comparison –

1. Not all the service description code was available in the literature to compare the service descriptions created using the templates against hence not everything could be compared.
2. The service descriptions created using the template did not have implementation specific information and hence do not account for the dynamic aspects of SOA.
3. The peer-to-peer example did not consist of a worked out example and hence could only be evaluated by comparing the template with the existing concepts, published in technical articles like Hasselmeyer, 2005 [24] and Papzoglou, 2003 [29].

5.4. Future Work

The templates created in the thesis research can be extended to include details such as message exchange protocols, protocol used for accessing the service and the registry standards, service level agreements, and service policies. Currently the templates address the abstract elements of a service and registry, as explained in Section 5.3. In order to make the templates more comparable to the standards like WSDL, they needs to include the implementation related elements and also account for dynamic aspects including the messaging framework, communication agreement and discovery. For example, the templates can be built according to different models for exchanging messages during service interactions like publish subscribe model and request-response models. Further work could include creating software which will utilize as whole SOA based application rather than treating the service descriptions and registry as separate entities. Once the user provides information for all the components of the templates, the software could generate service description documents to implement the services.

5.5. Conclusion

Services form the basis of SOA. Researchers are constantly trying to find new ways to develop systems, which can respond dynamically to the consumer needs and provide them with the most appropriate services. This thesis essentially presents a structural specification template that can be used to design a SOA-based peer-to-peer applications. Certain elements, when removed from the template, make it usable for implementing the traditional client server architecture based applications by introducing service orientation in them.

The models used in the specification templates attempt to implement new ways of using the basic SOA by changing the roles of the standard consumer, provider and registry and implementing these models to regular day to day applications.

In conclusion, this thesis can be used as a basis for the new research into making the process of creating SOA based applications simple. Software can be created which can use the template, and can aid in building SOA based applications as a whole instead of concentrating on service description and registry as separate entities.

References

[1] Berson, Alex, *Client/Server Architecture*, 2nd ed. Mc-Graw Hill, 1996

[2] OASIS RM-CS Committee “Reference Model for Service Oriented Architecture 1.0” Aug. 2006; http://www.oasis-open.org/committees/download.php/19679/soa-rm-cs.pdf

[3] "Service Oriented Architecture." Jan. 2007; http://www.serviceoriented.org/service_oriented_architecture.html

[4] Baresi, L., Di Nitto, E. and Ghezzi, C. "Toward Open-World Software: Issue and Challenges." *Computer* 39, 2006, pp. 36-43

[5] Denaro, G., M. Pezzé, D. Tosi, and Daniela Schilling, "Towards self-adaptive service-oriented architectures", *Proceedings of the 2006 workshop on Testing, analysis, and verification of web services and applications TAV-WEB '06, 2006*, pp. 10-16.

[6] Channabasavaiah, K., Holley K. and Tuggle, E. "Migrating to a service-oriented architecture, Part 1.” Dec 2003; http://www-128.ibm.com/developerworks/library/ws-migratesoa/

[7] Ramirez, A. “Three-tier Architecture,” Jul 2000; http://www.linuxjournal.com/article/3508

[8] Erl,Thomas, *Service Oriented Architecture: concepts, technology and design*, 1 ed. Prentice Hall, 2005.

[9] W3C “Web Services Description Language (WSDL) 1.1” March 2001; http://www.w3.org/TR/wsdl

[10] OASIS “UDDI Version 3.0” Oct 2004; http://www.uddi.org/pubs/uddi_v3.htm

[11] Ryan, Alexander "What is a SOA service" Nov. 2006; http://www.digerateur.com/articles/whatIsAService.jsp

[12] Chartier, Robert “Application Architecture: An N-tier Approach – Part 1” 2008; http://www.15seconds.com/issue/011023.htm

[13] Schneider, Jeff “Convergence of Peer and Web Services” Jul 2001; http://www.openp2p.com/pub/a/p2p/2001/07/20/convergence.html

[14] Govoni, Darren "Web Services Over P2P Networks." April 2002; http://webservices.sys-con.com/read/39425.htm

[15] Chen, Yinong; Tsai, W.T. et al. **“**Architecture Classification for SOA-Based Applications” *Proceedings of the Ninth IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing*, 2006, pp. 295-302.

[16] Garofalakis, John, Panagis, Yannis et al. “Web Discovery Mechanisms: Looking for a Needle in a Haystack?” 2004

[17] Garcia, Pedro et al. “Towards a Decentralized p2pWeb Service Oriented Architecture” 2006

[18] Maheshwari, P, Kanhere S., Parameswaran, N, “Service-Oriented Middleware for Peer-to-Peer Computing” *3rd IEEE International Conference on Industrial Informatics*, 2005, pp. 98-103.

[19] Keen, Martin, Acharya, Amit et al. *Patterns: Implementing an SOA using an Enterprise Service Bus.* 1 ed. IBM, 2004.

[20] Clombo, M., Di Nitto, E., et al. “Speaking a Common Language: A ConceptualModel for Describing Service-Oriented Systems” *Service-Oriented Computing - ICSOC 2005* Springer 2005, pp.48-60.

[21] Arsanjani, A., Ang, Jenny, Endrei, Mark et al. *Patterns: Service-Oriented Architecture and Web Services*, IBM, 2004.

[22] Kodali, Raghu “What is service-oriented architecture” June 2005; http://www.javaworld.com/javaworld/jw-06-2005/jw-0613-soa.html?page=1

[23] Hagemann, S., Letz, C., Vossen, G. “Web Services Discovery – Reality Check 2.0” *Third International Conference on Next Generation Web Services Practices*, 2007 pp. 113-118.

[24] Hasselmeyer, Peer “On Discovery Process Types” *Service-Oriented Computing – ICSOC 2005* Springer-Verlag 2005 pp. 144-156.

[25] OASIS “UDDI Version 2.03 Data Structure Reference” Jul 2002; http://uddi.org/pubs/DataStructure_v2.htm

[26] OASIS “Introduction to UDDI: Important Features and Functional Concepts” Oct 2004.

[27] BEA “Publishing and Finding Web Services Using UDDI” 2007; http://e-docs.bea.com/wls/docs81/webserv/uddi.html

[28] Brittenham, P., Francisco, C., Ehnebuske, D., Graham, S. “Understanding WSDL in a UDDI registry, Part 1.” Sep 2001; http://www.ibm.com/developerworks/webservices/library/ws-wsdl/

[29] Papazoglou, M., Kramer, B., Yang, J. “Leveraging Web-Services and Peer-to-Peer Networks” *Lecture Notes in Computer Science* Springer-Verlag 2003 pp. 485-501

[30] National E-health Transition Authority Ltd. “Example technical implementation of interoperable web services” July 2007;http://www.nehta.gov.au

[31] Curbera, F. Ehnebuske, D., Rogers, D. “UDDI Best Practice” May 2002;http://www.uddi.org/pubs/wsdlbestpractices.pdf

[32] Dhesiaseelan, Arulazi “What’s new in WSDL 2.0” May 2004; http://www.xml.com/pub/a/ws/2004/05/19/wsdl2.html

[33] Sarjoughian, H., Sungung, K. et al. “A Simulation Framework For Service-Oriented Computing Systems” Proceedings of the 2008 Winter Simulation Conference.

APPENDIX A

APPLICATION OF TEMPLATES

1. Specification Template for client server architecture

Consider an e-health application that comprises of two services – discharge summary sender and discharge summary receiver. The discharge summary sender generates the discharge summary and sends it to the receiver service. The receiver service returns the status of the discharge summary whenever requested. The status could be:

1– Acknowledged
2 – Pending
3 – Not received

The following is a list of features for this application

1. Discharge summary is generated by the discharge summary sender service.
2. Sender service sends the discharge summary to the receiving service.
3. At any point the discharge service sender may request the receiving service for the status of the discharge summary. The following are the different statuses which a discharge summary may have.

   1 – acknowledged

   2 – pending

   3 – not received

Participants:

a. Discharge summary sender
b. Discharge summary receiver

Services:

Service 1: Discharge summary sender

Name: dischargeSummarySender

Description: The discharge summary sender generates the discharge summary and sends it to the receiving service. The service also sends a request to check the status of the discharge summary.

Messages: The following are the messages that are sent or received by the service

1. Sent: Discharge summary
2. Sent: Request for status
3. Sent: Ping the receiver
4. Receive: Status of the discharge upon request

Functions:

a. Name: generateDischargeSummary()

   Description: generateDischargeSummary() generates the discharge summary.

   Input: none

   Output: dischargeSummaryId: String

   Access point: www.exampleurl.com

Service 2: Discharge summary receiver

Name: dishargeSummaryReceiver

Description: The discharge summary receiver service accepts the discharge summary from the sender service and when queried, returns the most current status of a particular discharge summary

Messages: The following are the messages that are sent or received by the service

1. Receive: The discharge summary
2. Receive: The request for status
3. Sent: The status of the discharge summary

Functions:

Name: checkStatus()

Description: The discharge summary receiver checks the status of the discharge summary.

Input: dischargeSummaryId: String

Output: status: integer

Service description documents

The following is a list of the service description and specification documents that are available in Appendix B

1. dischargeSummarySender.wsdl
2. dischargeSummaryReceiver.wsdl

2. Specification document for client server with registry example

Consider a stock quote service that retrieves the trade price of a stock, given its ticker symbol. Also included is a registry which is used to register and unregister various services. Provided below is the specification outline of the application.

The following is a list of features of the application

a. The stock quote service returns the trade price of the stock, given its ticker symbol.
b. The registry registers the stock quote service.

Participants:

1. Stockquote service
2. Registry

Service:

Name: Stockquote Service

Description: The Stockquote service returns the trade price of a stock given its ticker symbol.

Functions:

Name: getTradePrice()

Description: getTradePrice function returns the trade price of a stock, given its ticker symbol.

Input: ticker symbol: String

Output: trade price: float

Registry:

Input: information: Registered service information type

i. Service name: stockquote service
ii. Description: stockquote service retrieves the trade price of the stock, given its ticker symbol.
iii. Access Point: (service access URL)
iv. Reference pointer: (URL of the wsdl file)

Service description documents

The following is a list of the service description and specification documents that are available in Appendix C

1. Stockquote.xsd
2. Stockquote.wsdl
3. Registry.xsd

3. Specification document for multi-tier architecture example

Consider a simple timesheet submission process. Once an employee submits a timesheet, it goes through a verification process. If the timesheet is verified, it is accepted otherwise the timesheet is rejected. In case of rejection, the manager and the employee are notified. The verification process is decomposed into the following steps

1. Compare the hours recorded on timesheet to hours billed.
2. Confirm that authorization was given for any recorded overtime hours.
3. Confirm that the hours recorded for any particular project do not exceed the preset limit for the project.
4. Confirm that the total hours recorded for the week do not exceed the preset limit for the employee.

Following are the features of this application

1. Employee submits the timesheet.
2. Timesheet is verified.
3. Timesheet is accepted or rejected

Participants

Presentation layer: The presentation layer consists of the timesheet submit service which accepts the timesheet from the user and sends it to the business service layer. The presentation layer comprises of the service timesheetSubmission service. The timesheet is in the form of a document.

Business layer: The business layer consists of the services that perform the following tasks

1. Get recorded hours for the customer

2. Get the billed hours for the customer
3. Compare the recorded hours and billed hours
4. Get authorization
5. Confirm authorization
6. Get weekly hours limit
7. Compare weekly hours limit with the hours recorded
8. Update the employee history

The following is a list of services comprising the business layer

1. EmployeeTimeService – This service is used to get the weekly hours limit for the employee and update the employee history.
2. TimesheetService – This service is used to get the recorded hours for a customer and the overtime. The service also gets the authorization.
3. CustomerTimesheetService – this service is used to get the customer and the billed hours for the customer.

Application Layer: The application layer consists of the services that are not specific to a certain task and can be reused. In this example, the application layer consists of the NotificationService. This service notifies the employee and the manager for approval or rejection of the timesheet.

Services:

1. Name: TimesheetSubmissionService

   Description: The timesheetSubmissionService is used to submit the timesheet filled by an employee.

   Functions:

Name: timesheetSubmit()

Input:

a. document id: integer
b. documentLocation: String

Output: message: String

2. Name: Employee

   Description: The Employee Service is used to get the weekly hours limit for the employee and update the employee history.

   Functions:

   a. Name: getEmployeeId

      Input: None

      Output: employeeId:integer

   b. Name: getWeeklyHourLimit()

      Input: employeeId: integer

      Output: weeklyHourLimit: integer

      Access Point:

   c. Name: updateEmployeeHistory()

      Input: employeeId: integer

      employeeHistoryComment: string

      Output: None

3. Name: TimesheetService

   Description: The TimesheetService is used to get the recorded hours for the customer, the overtime hours and the authorization.

Functions:

a. Name:getRecordedHours()

   Input:customerId: integer

   Output: recordedHours:float

   Access point:

b. Name: getOvertimeHours()

   Input: customerId:integer

   Output: overtimeHours()

   Access Point:

c. Name: confirmAuthorization()

   Input: employeeId:integer

   Output: authorization:Boolean

4. Name: CustomerTimesheetService

   Description: The CustomerTimesheetService is used to get the number of hours billed to the customer.

   Functions:

   a. Name: getCustomer

      Input: none

      Output: customerId:integer

   b. Name: getBilledHours()

      Input: customerId: integer

      Output: billedHours:float

5. Name: NotificationService

Description: The NotificationService is used to send notification to the employee and the manager

Functions:

a. Name: sendMessage()

   Input: employeeId: integer

   Output: responseCode:integer

Service description documents

Following is a list of the service description and specification documents that are available in Appendix D

1. Timesheetsubmitservice.xsd
2. Timesheetsubmit.wsdl
3. EmployeeService.xsd
4. Employee.wsdl
5. customerTimesheetService.xsd
6. customertimesheet.wsdl
7. timesheetservice.xsd
8. timesheetservice.wsdl
9. notificationService.xsd
10. notificationService.wsdl

4. Specification document for Peer-to-peer architecture example

Consider an e-travel application described by Papazolglou et al. [29]. The application consists of

1. Super-peer (event notification service) – the super-peer maintains a list of all the peers that have published their services on the registry. Super-peer manages a set of operations for peers including joining the network, leaving the network, publishing services and subscribing to services.
2. Registry – registry maintains the information about all the peers in the network.
3. Peers - each peer can act as a service provider and a service consumer, depending upon its requirement.

Following is a chain of events that take place in the e-travel system

1. When a new peer comes in, it publishes its services on the registry
2. The super-peer receives a request from the new peer.
3. The super-peer matches the published information against all the subscriptions that may find the new peer's services useful.
4. In an event, a match is detected; the super-peer forwards the publication information of the new peer to the matched peers.
5. Super-peer also matches the subscription details of the new peer with its existing publications.
6. Once a match is detected, the super-peer forwards the information of the all the peers offering the services matching the subscription needs to the new peer.
7. Thus, all the peers hold the information regarding the peers that offer services that match the needs of the subscribed peers.

8. Each peer issues a request to all the peers in its list.
9. All the peers send the requesting peer their service description matching the request.

Following is a list of requirements of the application system

1. The new peer publishes its services on the registry.
2. The super-peer matches the published information against all the subscriptions that may find the new peer's services useful.
3. The super-peer sends the information of the new peer to all the subscribed peers.
4. The super-peer sends the list of the peers matching the subscription request to the new peer.
5. A peer issues a request to all the peers in its network.
6. The peer accepts the response from all the peers in its network.

Participants:

1. Super-peer
2. Peer node
3. Registry

Services:

Name: superPeer

Description: The super-peer service matches the new peer's publication with the subscribers and its subscription requests with the other peers' publications.

Functions:

a. Name: addPeer()

Description: add the peer to the super-peer’s list of publishers and subscribers.

Input: None

Output: peerId: string

b. Name: deletePeer()

Description: delete the peer from the super-peer’s list of publishers and subscribers.

Input: peerId:string

Output: None

c. Name: updatePublications()

Description: Super peer update the existing publication document of the peer with a new one.

Input:

i. PeerId: String
ii. publicationDocumentLocation: String

Output: None

d. Name: updateSubscription()

Description: Super peer update the existing subscription document of the peer with a new one.

Input:

i. PeerId: String
ii. subscriptionDocumentLocation: String

Output: None

e. Name: matchPublications()

   Input: publicationDocumentLocation:String

   Output: None

f. Name: matchSubscriptions()

   Input: subscriptionDocumentLocation:String

   Output: list_peers:list

Registry:

Input: information: Registered service information type

i. Service name: name of the service
ii. Description: description of the service
iii. Access Point: (service access URL)
iv. Category:
v. Reference pointer: (URL of the wsdl file)

Peer Node:

Functions:

a. Name: addPeer()

   Description: Add a peer with matching publication in the list of peers.

   Input: peerName: String

   peerAccess: String

   Output: peerId:String

b. Name: removePeer()

   Description: Remove a peer from the list.

   Input: PeerId: String

Output: None

c. Name: searchQuery()

Description: searchQuery function is used to search the peer data structure for a service satisfying the search criteria.

Input: searchCriteria: string

Output: serviceDescriptionLocation: String

Service description documents

Following is a list of the service description and specification documents that are available in Appendix E

1. superPeer.xsd
2. registry.xsd
3. peerNode.xsd

APPENDIX B

SERVICE DESCRIPTION DOCUMENTS FOR CLIENT-SERVER ARCHITECTURE

Following are the service description documents generated using the template for the discharge summary example as described in Appendix A

1. dischargeSummarySender.wsdl – dischargeSummarySender.wsdl consists of the service description of the dischargeSummarySender service without the binding parameters since the binding parameters are implementation specific and are not included in the template.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns:s="http://www.w3.org/2001/XMLSchema"
xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
xmlns:tns="http://example.org/ns/2007/dischargeSummarySender.wsdl"
targetNamespace="http://example.org/ns/2007/dischargeSummarySender.wsdl"
>
<types>
<s:schema elementFormDefault="qualified"
targetNamespace="hhttp://example.org/ns/2007/dischargeSummarySender.wsdl
">
    <documentation></documentation>
    <xsd:element name="sendDischargeSummary">
            <xsd:complexType>
                    <xsd:sequence>
                            <xsd:element name="dischargeSummaryId"
type="xsd:string"></xsd:element>
                    </xsd:sequence>
            </xsd:complexType>
    </xsd:element>
    <xsd:element name="sendDischargeSummaryResponse">
            <xsd:complexType/>
    </xsd:element>
    </s:schema>
</types>
<message name="sendDischargeSummaryInput">
    <part name="body" element="tns:sendDischargeSummary"/>
</message>
<message name="sendDischargeSummaryOutput">
    <part name="body" element="tns:sendDischargeSummaryResponse"/>
</message>
<portType name="dischargeSummarySender">
    <operation name="sendDischargeSummary">
            <input message="sendDischargeSummaryInput"></input>
```

```
          <output message="sendDischargeSummaryOutput"></output>
     </operation>
</portType>
</definitions>
```

2. dischargeSummaryReceiver.wsdl – dischargeSummaryReceiver.wsdl consists of the service description of the dischargeSummaryReceiver service without the binding parameters since the binding parameters are implementation specific and are not included in the template.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns:s="http://www.w3.org/2001/XMLSchema"
xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
xmlns:tns="http://example.org/ns/2007/test.wsdl"
targetNamespace="http://example.org/ns/2007/test.wsdl">
<types>
<s:schema elementFormDefault="qualified"
targetNamespace="hhttp://example.org/ns/2007/test.wsdl">
     <documentation></documentation>
     <xsd:element name="checkStatus">
          <xsd:complexType>
               <xsd:sequence>
                    <xsd:element name="dischargeSummaryId"
type="xsd:string"></xsd:element>
               </xsd:sequence>
          </xsd:complexType>
     </xsd:element>
     <xsd:element name="checkStatusResponse">
          <xsd:complexType>
               <xsd:sequence>
                    <xsd:element name="status"
type="xsd:integer">
                    </xsd:element>
               </xsd:sequence>
          </xsd:complexType>
     </xsd:element>
     </s:schema>
</types>
<message name="checkStatusInput">
     <part name="body" element="tns:checkStatus"/>
</message>
<message name="checkStatusOutput">
```

```
        <part name="body" element="tns:checkStatusResponse"/>
</message>
<portType name="dischargeSummaryReceiver">
        <operation name="checkStatus">
                <input message="checkStatusInput"></input>
                <output message="checkStatusOutput"></output>
        </operation>
</portType>
</definitions>
```

APPENDIX C

SERVICE DESCRIPTION DOCUMENTS FOR CLIENT-SERVER WITH REGISTRY

Following are all the documents generated using the specification template for the stockquote service example. Also included is the specification document for the registry structure and functions.

1. Stockqote.xsd – stockquote.xsd consists of the specification of the stockquote service generated using the template.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="qualified">
	<xs:complexType name="stockquote">
		<xs:annotation>
			<xs:documentation>stockquote service
specification</xs:documentation>
		</xs:annotation>
		<xs:sequence>
			<xs:element name="getTradePrice">
				<xs:complexType>
					<xs:sequence>
						<xs:element name="input">
							<xs:complexType>
								<xs:sequence>
									<xs:element
name="tickerSymbol" type="xs:string"/>
								</xs:sequence>
							</xs:complexType>
						</xs:element>
						<xs:element name="output">
							<xs:complexType>
								<xs:sequence>
									<xs:element
name="tradePrice" type="xs:integer"/>
								</xs:sequence>
							</xs:complexType>
						</xs:element>
					</xs:sequence>
				</xs:complexType>
			</xs:element>
		</xs:sequence>
	</xs:complexType>
</xs:schema>
```

2. Stockquote.wsdl -- stockquote.wsdl consists of the service description of the stockquote service without the binding parameters since binding parameters are implementation specific and are not included in the template

```
<?xml version="1.0" encoding="UTF-8"?>
xmlns:s="http://www.w3.org/2001/XMLSchema"
xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:tns="http://example.org/ns/2007/dischargeSummarySender.wsdl"
targetNamespace="http://example.org/ns/2007/stockquote.wsdl">
        <types>
                <s:schema elementFormDefault="qualified"
targetNamespace="hhttp://example.org/ns/2007/stockquote.wsdl">
                        <documentation/>
                        <xsd:element name="getTradePrice">
                                <xsd:complexType>
                                        <xsd:sequence>
                                                <xsd:element
name="tickerSymbol" type="xsd:string"/>
                                        </xsd:sequence>
                                </xsd:complexType>
                        </xsd:element>
                        <xsd:element name="getTradePriceResponse">
                                <xsd:complexType>
                                        <xsd:sequence>
                                                <xsd:element name="tradePrice"
type="xsd:float"
                                        </xsd:sequence>
                                </xsd:complexType>
                        </xsd:element>
                </s:schema>
  </types>
  <message name="getTradePriceInput">
          <part name="body" type="tns:getTradePrice"/>
  </message>
  <message name="getTradePriceOutput">
          <part name="body" type="tns:getTradePriceResponse"/>
  </message>
  <portType name="stockquote">
          <operation name="getTradePrice">
                  <input message="getTradePriceInput"/>
                <output message="getTradePriceOutput"/>
            </operation>
    </portType>
    </definitions>
```

3. Registry.xsd -- registry.xsd document consists of the registry data structure and function specification.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="qualified">
        <xs:complexType name="registryStructure">
                <xs:annotation>
                        <xs:documentation>registry service
specification</xs:documentation>
                </xs:annotation>
                <xs:sequence>
                        <xs:element name="stockquote">
                                <xs:complexType>
                                        <xs:sequence>
                                                <xs:element name="Description"
type="xs:string"/>
                                                <xs:element
name="AccessPoint">

        http://www.example.com/stockquote
                                                </xs:element>
                                                <xs:element
name="ReferencePointer">

        http://www.example.com/stockquote.wsdl
                                                </xs:element>
                                        </xs:sequence>
                                </xs:complexType>
                        </xs:element>
                </xs:sequence>
        </xs:complexType>
</xs:schema>
<xs:complexType name="registryFunctions">
        <xs:annotation>
                <xs:documentation>registry service
specification</xs:documentation>
        </xs:annotation>
        <xs:sequence>
                <xs:element name="registerService">
                <xs:complexType>
                        <xs:sequence>
                                <xs:element name="input">
                                        <xs:complexType>
                                                <xs:sequence>
                                                        <xs:element
name="service_name"  type="xs:string"/>
                                                        <xs:element
name="description" type="xs:string"/>
                                                        <xs:element
name="AccessPoint" type="xs:string"/>
                                                        <xs:element
name="Category" minOccurs="0" maxOccurs="1" type="xs:string"/>
                                                        <xs:element
name="ReferencePointer" type="xs:string"/>
                                                </xs:sequence>
                                        </xs:complexType>
                                </xs:element>
                                <xs:element name="output">
                                        <xs:complexType>
                                                <xs:sequence>
                                                        <xs:element
name="registrationId" type="xs:string"/>
                                                </xs:sequence>
                                        </xs:complexType>
                                </xs:element>
                        </xs:sequence>
                </xs:complexType>
        </xs:element>
        <xs:element name="unregisterService">
                <xs:complexType>
                        <xs:sequence>
                                <xs:element name="input">
                                        <xs:complexType>
                                                <xs:sequence>
                                                        <xs:element
name="registrationId"  type="xs:string"/>
                                                </xs:sequence>
                                        </xs:complexType>
                                </xs:element>
                                <xs:element name="output">
                                        <xs:complexType/>
                                </xs:element>
                        </xs:sequence>
                </xs:complexType>
```

```
          </xs:element>
          <xs:element name="query">
               <xs:complexType>
                    <xs:sequence>
                         <xs:element name="input">
                              <xs:complexType>
                                   <xs:sequence>
                                        <xs:element
name="queryString"  type="xs:string"/>
                                   </xs:sequence>
                              </xs:complexType>
                         </xs:element>
                         <xs:element name="output">
                              <xs:complexType>
                                   <xs:sequence>
                                        <xs:element
name="listServiceDescriptions" type="xs:list"/>
                                   </xs:sequence>
                              </xs:complexType>
                         </xs:element>
                    </xs:sequence>
               </xs:complexType>
          </xs:element>
     </xs:sequence>
</xs:complexType>
```

APPENDIX D

SERVICE DESCRIPTION DOCUMENTS FOR MULTI-TIER ARCHIECTURE

Following are all the documents generated using the specification template for the timesheet service example.

1. Timesheetsubmitservice.xsd – this document contains the specification of the service generated using the template.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="qualified">
  <xs:complexType name="timesheetSubmissionService">
      <xs:sequence>
          <xs:element name="timesheetSubmit">
            <xs:complexType>
              <xs:sequence>
                  <xs:element name="input">
                    <xs:complexType>
                        <xs:sequence>
                          <xs:element name="documentId"
type="xs:integer"/>
                          <xs:element name="timesheetDocumentLocation"
type="xs:string"/>
                        </xs:sequence>
                    </xs:complexType>
                  </xs:element>
                  <xs:element name="output">
                    <xs:complexType>
                        <xs:sequence>
                          <xs:element name="submitMessage"
type="xs:string"/>
                        </xs:sequence>
                    </xs:complexType>
                  </xs:element>
              </xs:sequence>
          </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

2. timesheetSubmission.wsdl – This document contains the service description of the timesheetsubmit service. This document is generated using the specification Timesheetsubmitservice.xsd shown above.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:s="http://www.w3.org/2001/XMLSchema"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:tns="http://example.org/ns/2007/timesheetSubmission"
targetNamespace="http://example.org/ns/2007/timesheetSubmission.wsdl">
<types>
<s:schema elementFormDefault="qualified"
targetNamespace="hhttp://example.org/ns/2007/timesheetSubmission.wsdl"
>
  <documentation>the timesheet submission service is used as a presentation
layer in the multi-tier application
  </documentation>
  <xsd:element name="timsheetSubmit">
    <xsd:complexType>
          <xsd:sequence>
                <xsd:element name="documentId" type="xsd:integer"/>
                <xsd:element name="documentLocation" type="xsd:string"/>
           </xsd:sequence>
         </xsd:complexType>
   </xsd:element>
   <xsd:element name="timesheetSubmitResponse">
         <xsd:complexType>
          <xsd:sequence>
                <xsd:element name="submitResponse" type="xsd:string" />
          </xsd:sequence>
         </xsd:complexType>
   </xsd:element>
</s:schema>
</types>
        <message name="timesheetSubmitInput">
                <part name="body" element="timesheetSubmit"/>
        </message>
        <message name="timesheetSubmitOutput">
                <part name="body" element="timesheetSubmitResponse"/\>
        </message>
```

```
<portType name="timesheetSubmission">
        <operation name="timesheetSubmit">
                <input message="timesheetSubmitInput"/>
                <output message="timesheetSubmitOutput"/>
        </operation>
</portType>
</definitions>
```

3. EmployeeService.xsd – This document consists of the employee service specification, generated using the template.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xs:complexType name="EmployeeService">
        <xs:annotation>
         <xs:documentation>Comment describing your root
element</xs:documentation>
         </xs:annotation>
         <xs:sequence>
           <xs:element name="getWeeklyHourLimit">
                <xs:complexType>
                  <xs:sequence>
                        <xs:element name="input">
                          <xs:complexType>
                                <xs:sequence>
                                  <xs:element name="employeeId"
type="xs:integer"/>
                                 </xs:sequence>
                           </xs:complexType>
                         </xs:element>
                         <xs:element name="output">
                          <xs:complexType>
                            <xs:sequence>
                                <xs:element name="weeklyHoursLimit"
type="xs:integer"/>
                            </xs:sequence>
                          </xs:complexType>
                         </xs:element>
                        </xs:sequence>
                  </xs:complexType>
                </xs:element>
                <xs:element name="updateEmployeeHistory">
                  <xs:complexType>
                        <xs:sequence>
```

```
                                <xs:element name="input">
                                  <xs:complexType>
                                         <xs:sequence>
                                           <xs:element name="employeeId"
type="xs:integer"/>
                                         </xs:sequence>
                                  </xs:complexType>
                                </xs:element>
                                <xs:element name="output">
                                  <xs:complexType>
                                         <xs:sequence>
                                           <xs:element name="responseCode"
type="xs:integer"/>
                                    </xs:sequence>
                                   </xs:complexType>
                                 </xs:element>
                              </xs:sequence>
                       </xs:complexType>
                     </xs:element>
                   </xs:sequence>
          </xs:complexType>
</xs:schema>
```

4. Employee.wsdl – This document consists of the employee service description as specified in the document “EmployeeService.xsd” above.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns:s="http://www.w3.org/2001/XMLSchema"
xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
xmlns:tns="http://example.org/ns/2007/Employee"
targetNamespace="http://example.org/ns/2007/Employee.wsdl">
<types>
<s:schema elementFormDefault="qualified"
targetNamespace="hhttp://example.org/ns/2007/Employee.wsdl">
  <xsd:element name="getEmployeeId">
    <xsd:complexType/>
  </xsd:element>
  <xsd:element name="getEmployeeIdResponse">
    <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="employeeId" type="xsd:integer">
            </xsd:element>
           </xsd:sequence>
     </xsd:complexType>
   </xsd:element>
   <xsd:element name="getWeeklyHoursLimit">
    <xsd:complexType>
        <xsd:sequence>
         <xsd:element name="employeeId" type="xsd:integer">
         </xsd:element>
        </xsd:sequence>
       </xsd:complexType>
   </xsd:element>
   <xsd:element name="getWeeklyHoursLimitResponse">
       <xsd:complexType>
         <xsd:sequence>
              <xsd:element name="weeklyHoursLimit"
 type="xsd:integer">
              </xsd:element>
         </xsd:sequence>
        </xsd:complexType>
       </xsd:element>
       <xsd:element name="updateEmployeeHistory">
        <xsd:complexType>
         <xsd:sequence>
              <xsd:element name="employeeId" type="xsd:integer">
              </xsd:element>
              <xsd:element name="comment" type="xsd:string">
              </xsd:element>
         </xsd:sequence>
         </xsd:complexType>
       </xsd:element>
       <xsd:element name="updateEmployeeHistoryResponse"/>
 </s:schema>
</types>
 <message name="getEmployeeIdInput"/>
 <message name="getEmployeeIdOutput">
       <part name="body" element="tns:getEmployeeIdResponse"/>
 </message>
 <message name="getWeeklyHoursLimitInput">
       <part name="body" element="tns:getWeeklyHoursLimit"/>
 </message>
 <message name="getWeeklyHoursLimitOutput">
       <part name="body"
 element="tns:getWeeklyHoursLimitResponse"/>
 </message>
 <message name="updateEmployeeHistoryInput">
       <part name="body" element="tns:updateEmployeeHistory"/>
 </message>
 <message name="updateEmployeeHistoryOutput">
```

```
        <part name="body"
element="tns:updateEmployeeHistoryResponse"/>
</message>
<portType name="Employee">
        <operation name="getEmployeeId">
                <input message="getEmployeeIdInput"/>
                <output message="getEmployeeOutput"/>
        </operation>
        <operation name="getWeeklyHoursLimit">
                <input message="getWeeklyHoursLimitInput"/>
                <output message="getWeeklyHoursLimitOutput">
        </operation>
        <operation name="updateEmployeeHistory">
                <input message="updateEmployeeHistoryInput"/>
                <output message="updateEmployeeHistoryOuput"/>
        </operation>
</portType>
</definitions>
```

5. customerTimesheetService.xsd – This document consists of the customertimesheet service specification, generated using the template.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified">
<xs:element name="customerTimesheetService">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="getCustomer">
                <xs:complexType>
                  <xs:sequence>
                        <xs:element name="input" />
                         <xs:element name="output"
                                  <xs:complexType>
                                <xs:sequence>
                                  <xs:element name="customerId"
type="xs:integer"/>
                                 </xs:sequence>
                                </xs:complexType>
                          </xs:element>
                        </xs:sequence>
                   </xs:complexType>
```

```
                    </xs:element>
                    <xs:element name="getBilledHours">
                     <xs:complexType>
                            <xs:sequence>
                              <xs:element name="input">
                               <xs:complexType>
                                     <xs:sequence>
                                       <xs:element name="customerId"
type="xs:integer"/>
                       </xs:sequence>
                              </xs:complexType>
                            </xs:element>
                            <xs:element name="output">
                              <xs:complexType>
                                     <xs:sequence>
                                       <xs:element name="billedHours"
type="xs:float"/>
                                      </xs:sequence>
                                </xs:complexType>
                              </xs:element>
                            </xs:sequence>
                      </xs:complexType>
                     </xs:element>
                    </xs:sequence>
                  </xs:complexType>
         </xs:element>
</xs:schema>
```

6. customertimesheet.wsdl – This document contains the customertimesheet service description.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns:s="http://www.w3.org/2001/XMLSchema"
xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
xmlns:tns="http://example.org/ns/2007/customertimesheet.wsdl"
targetNamespace="http://example.org/ns/2007/customertimesheet.wsdl">
<types>
<s:schema elementFormDefault="qualified"
targetNamespace="hhttp://example.org/ns/2007/customertimesheet.wsdl">
  <xsd:element name="getCustomer">
        <xsd:complexType/>
  </xsd:element>
  <xsd:element name="getCustomerResponse">
```

```
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name="customerId" type="xsd:integer"/>
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
            <xsd:element name="getBilledHours">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name="customerId" type="xsd:inetger"/>
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
            <xsd:element name="getBilledHoursResponse">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name="billedHours" type="xsd:float"/>
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
        </s:schema>
    </types>
        <message name="getCustomerIdInput">
            <part name="body" type="getCustomerId"/>
        </message>
        <message name="getCustomerIdOutput">
            <part name="body" type="getCustomerIdResponse"/>
        </message>
        <message name="getBilledHoursInput">
            <part name="body" type="getBilledHours"/>
        </message>
        <message name="getBilledHoursOutput">
            <part name="body" type="getBilledHoursResponse"/>
        </message>
        <portType name="customertimesheet">
            <operation name="getCustomerId">
                <input message="getCustomerIdInput"/>
                <output message="getCustomerIdOutput"/>
            </operation>
            <operation name="getBilledHours">
                <input message="getBilledHoursInput"/>
                <output message="getBilledHoursOutput"/>
            </operation>
        </portType>
```

7. timesheetservice.xsd – This document consists of the timesheetservice specification generated using the template.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified">
<xs:element name="TimesheetService">
<xs:complexType>
<xs:sequence>
  <xs:element name="getRecordedHours">
    <xs:complexType>
            <xs:sequence>
              <xs:element name="input">
                <xs:complexType>
                        <xs:sequence>
                          <xs:element name="customerId" type="xs:integer" />

                                    </xs:sequence>
                        </xs:complexType>
                </xs:element>
                <xs:element name="output">
                        <xs:complexType>
                          <xs:sequence>
                            <xs:element name="recordedHours" type="xs:float" />

                                      </xs:sequence>
                          </xs:complexType>
                        </xs:element>
                </xs:sequence>
              </xs:complexType>
  </xs:element>
  <xs:element name="getOvertimeHours">
    <xs:complexType>
             <xs:sequence>
               <xs:element name="input">
                        <xs:complexType>
                          <xs:sequence>
                            <xs:element name="customerId" type="xs:integer" />
                          </xs:sequence>
                         </xs:complexType>
                        </xs:element>
                        <xs:element name="output">
                         <xs:complexType>
                           <xs:sequence>
                                   <xs:element name="overtimeHours" type="xs:float" />
```

```
                    </xs:sequence>
                   </xs:complexType>
                  </xs:element>
                 </xs:sequence>
              </xs:complexType>
      </xs:element>
         <xs:element name="confirmAuthorization">
          <xs:complexType>
           <xs:sequence>
               <xs:element name="input">
                <xs:complexType>

                          <xs:sequence>
                     <xs:element name="employeeId" type="xs:integer">
                     </xs:element>
                    </xs:sequence>
                 </xs:complexType>
                </xs:element>
                <xs:element name="output">
                 <xs:complexType>
                     <xs:sequence>
                      <xs:element name="authorization"
type="xs:boolean">
                      </xs:element>
                     </xs:sequence>
                  </xs:complexType>
                 </xs:element>
               </xs:sequence>
              </xs:complexType>
          </xs:element>
         </xs:sequence>
        </xs:complexType>
   </xs:element>
</xs:schema>
```

8. Timesheetservice.wsdl – This document consists of the service description for timesheetservice.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns:s="http://www.w3.org/2001/XMLSchema"
xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
xmlns:tns="http://example.org/ns/2007/timesheetService"
targetNamespace="http://example.org/ns/2007/timesheetService.wsdl">
<types>
```

```
<s:schema elementFormDefault="qualified"
targetNamespace="hhttp://example.org/ns/2007/timesheetService.wsdl">
  <xsd:element name="getRecodedHours">
         <xsd:complexType>
           <xsd:sequence>
                  <xsd:element name="customerId" type="xsd:string"/>
            </xsd:sequence>
         </xsd:complexType>
   </xsd:element>
   <xsd:element name="getRecordedHoursResponse">
         <xsd:complexType>
           <xsd:sequence>
                  <xsd:element name="recordedHours" type="xsd:float"/>
            </xsd:sequence>
          </xsd:complexType>
         </xsd:element>
         <xsd:element name="getOvertimeHours">
           <xsd:complexType>
                  <xsd:sequence>
                    <xsd:element name="customerId" type="xsd:integer"/>
                  </xsd:sequence>
           </xsd:complexType>
          </xsd:element>
          <xsd:element name="getOvertimeHoursResponse">
           <xsd:complexType>
            <xsd:sequence>
                  <xsd:element name="overtimeHours" type="xsd:float"/>
            </xsd:sequence>
           </xsd:complexType>
          </xsd:element>
          <xsd:element name="confirmAuthorization">
           <xsd:complexType>
             <xsd:sequence>
                   <xsd:element name="employeeId" type="xsd:integer"/>
                  </xsd:sequence>
            </xsd:complexType>
           </xsd:element>
           <xsd:element name="confirmAuthorizationResponse">
            <xsd:complexType>
                   <xsd:sequence>
                     <xsd:element name="authorization" type="xs:boolean"/>
                   </xsd:sequence>
            </xsd:complexType>
           </xsd:element>
         </s:schema>
```

```
</types>
<message name="getRecordedHoursInput">
        <part name="body" type="getRecordedHours"/>
</message>
<message name="getRecodedHoursOutput">
        <part name="body" type="getRecordedHoursResponse"/>
</message>
<message name="getOvertimeHoursInput">
        <part name="body" type="getOvertimeHours"/>
</message>
<message name="getOvertimeHoursOutput">
        <part name="body" type="getOvertimeHoursResponse"/>
</message>
<message name="confirmAuthorizationInput">
        <part name="body" type="confirmAuthorization"/>
</message>
<message name="confirmAuthorizationOutput">
        <part name="body" type="confirmAuthorizationResponse"/>
</message>
<portType name="timesheetservice">
        <operation name="getRecordedHours">
                <input message="getRecordedHoursInput"/>
                <output message="getRecordedHoursOutput"/>
        </operation>
        <operation name="getOvertimeHours">
                <input message="getOvertimeHoursInput"/>
                <output message="getOvertimeHoursOutput"/>
        </operation>
        <operation name="confirmAuthorization">
                <input message="confirmAuthorizationInput"/>
                <output message="confirmAuthorizationOutput"/>
        </operation>
</portType>
</definitions>
```

9. notificationService.xsd – notificationService.xsd consists of the

   notificationService specification generated using the template.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xs:element name="NotificationService">
    <xs:complexType>
         <xs:sequence>
```

```
            <xs:element name="sendNotification">
                  <xs:complexType>
                    <xs:sequence>
                          <xs:element name="input">
                            <xs:complexType>
                                  <xs:sequence>
                                    <xs:element name="employeeId"
type="xs:integer"/>
                                    </xs:sequence>
                                  </xs:complexType>
                            </xs:element>
                            <xs:element name="output">
                              <xs:complexType/>
                            </xs:element>
                          </xs:sequence>
                      </xs:complexType>
                    </xs:element>
                  </xs:sequence>
            </xs:complexType>
          </xs:element>
</xs:schema>
```

10. notificationService.wsdl – The notificationService.wsdl consists of the service description for the notification service.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns:s="http://www.w3.org/2001/XMLSchema"
xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
xmlns:tns="http://example.org/ns/2007/notificationService"
targetNamespace="http://example.org/ns/2007/notificationService.wsdl">
<types>
  <s:schema elementFormDefault="qualified"
targetNamespace="hhttp://example.org/ns/2007/notificationService.wsdl">
    <xsd:element name="sendNotificaiton">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="employeeId" type="xsd:integer"/>
             </xsd:sequence>
           </xsd:complexType>
          </xsd:element>
          <xsd:element name="sendNotificationResponse">
            <xsd:complexType/>
          </xsd:element>
   </s:schema>
```

```
</types>
<message name="sendNotificationInput">
        <part name="body" type="sendNotification"/>
</message>
<message name="sendNotificationOutput">
        <part name="body" type="sendNotificationResponse"/>
</message>
<portType name="notficationService">
        <operation name="sendNotification">
                <input message="sendNotificationInput"/>
                <output message="sendNotificationOutput"/>
        </operation>
</portType>
</definitions>
```

APPENDIX E

SERVICE DESCRIPTION DOCUMENTS FOR PEER-TO-PEER ARCHIECTURE

Following are the service description documents generated using the template for the peer-to-peer example as described in Appendix A

1. superPeer.xsd – The superPeer.xsd consists of super peer specification in the xml format, generated from the template.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="qualified">
  <xs:complexType name="superPeer">
    <xs:sequence>
      <xs:element name="addPeer">
              <xs:complexType>
                <xs:sequence>
                        <xs:element name="input">
                          <xs:complexType/>
                         </xs:element>
                         <xs:element name="output">
                           <xs:complexType>
                             <xs:sequence>
                                     <xs:element name="peerId" type="xs:string"/>
                                    </xs:sequence>
                             </xs:complexType>
                           </xs:element>
                          </xs:sequence>
                         </xs:complexType>
                        </xs:element>
                        <xs:element name="deletePeer">
                          <xs:complexType>
                            <xs:sequence>
                                    <xs:element name="input">
                                      <xs:complexType>
                                        <xs:sequence>
                                                <xs:element name="peerId" type="xs:string"/>
                                         </xs:sequence>
                                       </xs:complexType>
                                      </xs:element>
                                      <xs:element name="output">
                                        <xs:complexType/>
                                       </xs:element>
                                      </xs:sequence>
                                    </xs:complexType>
                             </xs:element>
```

```
                    <xs:element name="updatePublications">
                          <xs:complexType>
                            <xs:sequence>
                                  <xs:element name="input">
                                    <xs:complexType>
                                          <xs:sequence>
                                            <xs:element name="peerId"
type="xs:string"/>
                                            <xs:element
name="publicationDocumentLocation" type="xs:string"/>
                                          </xs:sequence>
                                         </xs:complexType>
                                        </xs:element>
                                        <xs:element name="output">
                                          <xs:complexType/>
                                        </xs:element>
                                    </xs:sequence>
                                  </xs:complexType>
                             </xs:element>
                                 <xs:element name="updateSunbscription">
                                  <xs:complexType>
                                    <xs:sequence>
                                        <xs:element name="input">
                                          <xs:complexType>
                                                <xs:sequence>
                                                  <xs:element name="peerId"
type="xs:string"/>
                                                  <xs:element
name="subscriptionDocumentLocation" type="xs:string"/>
                                                  </xs:sequence>
                                                </xs:complexType>
                                           </xs:element>
                                           <xs:element name="output">
                                                 <xs:complexType/>
                                           </xs:element>
                                          </xs:sequence>
                                     </xs:complexType>
                                        </xs:element>
                                        <xs:element name="matchPublications">
                                          <xs:complexType>
                                            <xs:sequence>
                                                  <xs:element name="input">
                                                    <xs:complexType>
                                                          <xs:sequence>
```

```
                                                            <xs:element
name="publicationDocumentLocation" type="xs:string"/>
                                                          </xs:sequence>
                                                     </xs:complexType>
                                                  </xs:element>
                                                  <xs:element name="output">
                                                    <xs:complexType/>
                                                  </xs:element>
                                            </xs:sequence>
                                          </xs:complexType>
                                   </xs:element>
                                   <xs:element name="matchSubscriptions">
                                     <xs:complexType>
                                          <xs:sequence>
                                            <xs:element name="input">
                                                  <xs:complexType>
                                                    <xs:sequence>
                                                          <xs:element
name="subscriptionDocumentLocation" type="xs:string"/>
                                                     </xs:sequence>
                                                   </xs:complexType>
                                                  </xs:element>
                                                  <xs:element name="output">
                                                    <xs:complexType>
                                                          <xs:sequence>
                                                            <xs:element
name="list_peers" type="xs:list"/>
                                                          </xs:sequence>
                                                    </xs:complexType>
                                                  </xs:element>
                                          </xs:sequence>
                                  </xs:complexType>
                          </xs:element>
                  </xs:sequence>
          </xs:complexType>
</xs:schema>
```

2. Registry.xsd – the registry.xsd consists of the registry specification used as a part of this application.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="qualified">
	<xs:complexType name="registryStructure">
		<xs:annotation>
			<xs:documentation>registry service
specification</xs:documentation>
		</xs:annotation>
		<xs:sequence>
			<xs:element name="stockquote">
				<xs:complexType>
					<xs:sequence>
						<xs:element name="Description"
type="xs:string"/>
						<xs:element
name="AccessPoint">

	http://www.example.com/stockquote
						</xs:element>
						<xs:element
name="ReferencePointer">

	http://www.example.com/stockquote.wsdl
						</xs:element>
					</xs:sequence>
				</xs:complexType>
			</xs:element>
		</xs:sequence>
	</xs:complexType>
</xs:schema>
<xs:complexType name="registryFunctions">
	<xs:annotation>
		<xs:documentation>registry service
specification</xs:documentation>
	</xs:annotation>
	<xs:sequence>
		<xs:element name="registerService">
			<xs:complexType>
				<xs:sequence>
					<xs:element name="input">
						<xs:complexType>
```

```
                                                    <xs:sequence>
                                                         <xs:element
name="service_name"  type="xs:string"/>
                                                         <xs:element
name="description" type="xs:string"/>
                                                         <xs:element
name="AccessPoint" type="xs:string"/>
                                                         <xs:element
name="Category" minOccurs="0" maxOccurs="1" type="xs:string"/>
                                                         <xs:element
name="ReferencePointer" type="xs:string"/>
                                                    </xs:sequence>
                                               </xs:complexType>
                                          </xs:element>
                                          <xs:element name="output">
                                               <xs:complexType>
                                                    <xs:sequence>
                                                         <xs:element
name="registrationId" type="xs:string"/>
                                                    </xs:sequence>
                                               </xs:complexType>
                                          </xs:element>
                                     </xs:sequence>
                                </xs:complexType>
                           </xs:element>
                           <xs:element name="unregisterService">
                                <xs:complexType>
                                     <xs:sequence>
                                          <xs:element name="input">
                                               <xs:complexType>
                                                    <xs:sequence>
                                                         <xs:element
name="registrationId"  type="xs:string"/>
                                                    </xs:sequence>
                                               </xs:complexType>
                                          </xs:element>
                                          <xs:element name="output">
                                               <xs:complexType/>
                                          </xs:element>
                                     </xs:sequence>
                                </xs:complexType>
                           </xs:element>
                           <xs:element name="query">
                                <xs:complexType>
                                     <xs:sequence>
```

```
                                        <xs:element name="input">
                                                <xs:complexType>
                                                        <xs:sequence>
                                                                <xs:element
name="queryString"  type="xs:string"/>
                                                        </xs:sequence>
                                                </xs:complexType>
                                        </xs:element>
                                        <xs:element name="output">
                                                <xs:complexType>
                                                        <xs:sequence>
                                                                <xs:element
name="listServiceDescriptions" type="xs:list"/>
                                                        </xs:sequence>
                                                </xs:complexType>
                                        </xs:element>
                                </xs:sequence>
                        </xs:complexType>
                </xs:element>
        </xs:sequence>
</xs:complexType>
```

3. peerNode.xsd – this document consists of the specification elements of a

   peerNode, generated using the template.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="qualified">
  <xs:complexType name="peerNode">
        <xs:sequence>
          <xs:element name="addPeer">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="input">
                          <xs:complexType>
                            <xs:sequence>
                              <xs:element name="peerName" type="xs:string"/>
                              <xs:element name="peerAccess" type="xs:string"/>
                             </xs:sequence>
                            </xs:complexType>
                           </xs:element>
                           <xs:element name="output">
                            <xs:complexType>
                              <xs:sequence>
                                    <xs:element name="peerId" type="xs:string"/>
                              </xs:sequence>
                             </xs:complexType>
                            </xs:element>
                          </xs:sequence>
                     </xs:complexType>
                    </xs:element>
                    <xs:element name="removePeer">
                          <xs:complexType>
                            <xs:sequence>
                              <xs:element name="input">
                                    <xs:complexType>
                                      <xs:sequence>
                                        <xs:element name="peerId"
type="xs:string"/>
                                       </xs:sequence>
                                      </xs:complexType>
                                     </xs:element>
                                    <xs:element name="output">
                                 <xs:complexType/>
                             </xs:element>
                            </xs:sequence>
                    </xs:complexType>
                </xs:element>
                <xs:element name="searchQuery">
                    <xs:complexType>
                     <xs:sequence>
                          <xs:element name="input">
                           <xs:complexType>
                                <xs:sequence>
                                 <xs:element name="searchCriteria"
type="xs:string"/>
                                </xs:sequence>
                                </xs:complexType>
                            </xs:element>
                            <xs:element name="output">
                                 <xs:complexType>
                                   <xs:sequence>
                                          <xs:element
name="serviceDescriptionLocation" type="xs:string"/>
                                   </xs:sequence>
                                  </xs:complexType>
                                </xs:element>
                               </xs:sequence>
                           </xs:complexType>
                        </xs:element>
                    </xs:sequence>
             </xs:complexType>
</xs:schema>
```