Technische Universität Wien

M A G I S T E R A R B E I T

Generic web service client for SOPA framework

Ausgeführt am Institut für Softwaretechnik und Interaktive Systeme
Der Technischen Universität Wien

Unter der Anleitung von O. Univ. Prof. A Min Tjoa

durch

Reza Rawassizadeh

Fugbach gasse 8/6
A-1020,Wien

_____ _____

Datum Unterschrift (Student)

Eidesstattliche Erklärung

Ich erkläre an Eides statt, daß ich die vorliegende Arbeit selbständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benützt und die den benutzen Quellen wörtlich oder entnommenen Stellen als solche kenntlich gemacht habe.

7-2-2007 Reza Rawassizadeh

Abstract

In service Oriented computing domains, a service needs to be requested by a client or another service, service will be executed after receiving a SOC patterned request from a client (or another service).

The Main goal of this project is to provide a generic client which can execute all given services by utilizing WSDL file.

This component is designed to be embedded in SOPA (Service Oriented Pipeline Architecture) which is a lightweight implementation of service oriented framework. SOPA aims to extend the usage of Web Services to personal computers with a simple and powerful approach, using its enterprise components. This framework help developers to build a useful gadget from existing services and share it with others. SOPA framework has been developed in Institute of Software Technology and Interactive Systems in conjunction with SemanticLIFE project.

In the first part of this thesis an introduction to web service technologies is provided then each of java web service libraries will be analyzed separately, and finally the functionality of this component in SOPA framework will be explored.

Kurzfassung

Diese Diplomarbeit ist über die Bildung eines generischen Clients zur durschführung von Service mittels WSDL.
In dieser Arbeit wird die SOPA(Service Oriented Pipeline Architecture) framework als Basis der Implementierung herangezogen.
Auf der Grundlage von sogenannten Pipelines der SOPA Architektur wird in dieser Diplomarbeit erreicht, dass der Benutzer in einfachster Weise geeignete Services für die die Zusammensetzung (auch komplexer) Aufgaben generisch auszuwählen.
Nach Angabe geeigneter Inputparameter wird Innerhalb der Herangehenweise in SOPA die entsprechenden SOPA request getätigt.
Als letzer Schrift erfolgt die Durchführung der SOPA requests durch die sogennante „Execution engine" , welche die empfängen "SOPA-reuqests" bearbeitet und letzlich in einem SOAP response oder String format der ergebnis des Web service liefert.

Die Arbeit liefert in den ersten Kapiteln eine Zusammenschau der Web-Service-Technologie, um dann in die letzen zwei Kapiteln den Beitrag welcher vor allem in der Implementierung zu sehen ist, zu beschreiben.

Acknowledgment

This work is the successive result of the SOPA framework project which has been done in the Institute for Software Technology and Interactive Systems (Vienna University of Technology). I would like to thank Professor A Min Tjoa who accepted me in his institute and helped me with this interesting project which was a remarkable experience that will lead me to another research in this field.

Also I would like to sincerely thank Mr. Amin Anjomshoaa for his help, patience and guidance.

I would like to thank my parents, for their love and support. Without their help this degree would not have been possible.

Table of Content

Chapter 1

Introduction

Information of this chapter is coming from following resources:

- The Java™ EE 5 Tutorial For Sun Java System Application Server Platform Edition 9
- WSIF: Web Service Invocation framework http://ws.apache.org/wsif
- SOAP client http://www.soapclient.com

In service Oriented computing domain, when a service is needed to be consumed, a client or service requestor should be available for that service to consume (request) it. The main steps for this approach are as follows:

1. Based on the service WSDL file, user can choose Service, port, operation and fills associated input values. The results can be either exported as input of SOPA framework as pipelines or can be used in abstract mode to call the Web service.
2. After user provides operation's essential inputs through GUI or CallWS client plug-in, user or SOPA framework will create a SOAP request for web service.
3. Execution engine gets the created SOAP request and sends it for execution; process is suspended until the result is returned from web service. Result would be a SOAP response (if service execution is available) in String format.

In rest of this chapter we will introduce some related technologies and existing SOAP clients.

1.1. Web service Clients

Following technologies do the similar thing like generic client:

Stylus studio tool, Ssoapclient, WSIF

Unfortunately the details of Stylus approach is not published, however it behaves similar to other SOAP clients. In this chapter we will introduce the other two well-known web sources clients which are SOAPClient and WSIF.

1.1.1 SOAPClient

In this project SOAPclient used as test tool to check if the web service is available or not and if my SOAP message created for exact service is right or wrong, when I was in development phase the availability of service is checked with SOAPClient, For example when Soap client can't execute the given service this means for developer this service is not available for execution. Unfortunately this tool is not free and not open source so it can't use it as pattern for this project implementation.

Soap client web site provides following concrete explanation for using soap client:
"This generic SOAP client allows you to access web services using a web browser. It performs dynamic bindings and executes methods at remote web services. Executing a SOAP service is a two-step process:

1. Enter the Web Service Description Language (WSDL) file, and click the retrieve button. The SOAP Server will build HTML forms dynamically based on the description file.
2. Enter parameters in the HTML form and click the Execute button. This triggers the execution of the remote method. A SOAP client object will be created, which performs parameter binding, message construction/delivery,

and finally response decoding. The result is then sent to user's browser as a HTTP message."

For more information you can refer to soap client web site: www.soapclient.com

1.1.2 WSIF (Web Service Invocation framework)

This is another tool provided by apache organization named WSIF, as apache defined:

"WSIF enables developers to interact with abstract representations of Web services through their WSDL descriptions instead of working directly with the Simple Object Access Protocol (SOAP) APIs, which is the usual programming model. With WSIF, developers can work with the same programming model regardless of how the Web service is implemented and accessed.

WSIF allows stub less or dynamic invocation (DII) of a Web service, based upon examination of the meta-data about the service at runtime. It also allows updated implementations of a binding to be plugged into WSIF at runtime, and it allows the calling service to defer choosing a binding until runtime.

Finally, WSIF is closely based upon WSDL, so it can invoke any service that can be described in WSDL."

It is free and open source like other Apache technologies, but the disadvantage of this tool is; that it is implemented with JAX-RPC and it is compliant with J2EE 1.4, I In the proposed approach we have used JAX-WS Which is compatible to JEE5

This component didn't call web service within stub, proxy object or any other form of object creation.

This component did it one level lower because WSDL file will be parsed, soap request will be created from file and it sends soap request in XML format to service and receive SOAP response also in XML format, we can say this solution is one step

lower level than proxy object creation. For more information about WSIF you can refer to following link: http://ws.apache.org/wsif

Web services are core component for Service oriented Architecture I tried to take SOA and Web Service apart and analyze them in different part but it is not possible to separate them, because Web service are core component of Service Oriented architecture. I just take a glance on UDDI, because it is out of scope of this project and try to explain SOAP part in detail because this project relies heavily on SOAP message creation and SOAP level service execution.

Chapter 2

Introduction to web service technology

Information for this chapter is adopted from following resources:

- Service Oriented Architecture Cocepts, technology and Design by Thomas Erl
- Service Oriented Architecture a Field guide to integrating XML and web services by Thomas Erl
- J2EE Web Services By Richard Monson-Haefel
- www.wikipedia.com
- SOAP's Two Messaging Styles By: Rickland Hollar

2.1 What is Service?

As W3C defines a Web service is a software system designed for supporting interoperable Machine to Machine interaction over a network.
Web services are mostly nothing more than application programming interfaces that can be accessed over Internet.

The W3C Web service definition cover many systems, but in common usage web service refers to those services that use SOAP XML envelope and their interfaces described by WSDL file.

The concept of services within an application has been around for a while.

Web Services, like software components, are independent blocks that collectively represent a software application. Oppose to traditional components, services have a number of unique characteristics that allow them to participate as part of a service-oriented architecture.
Each service is responsible for its own domain, which typically means limiting its scope to a specific business function or a group of logically related functions.

This approach results in the creation of units of business functionality
loosely bound in a standard communications framework. Due to the independence that services enjoy within this framework, the programming logic they encapsulate does not need to comply to any one platform or technology set.

2.2 Building Blocks of Service Oriented Architecture

There is no much effort needed to append some Web services to application. This limited integration is appropriate as supplementing current application architecture with service-based functionalities that meets a specific project requirement.

There is a distinct difference between an application that uses Web services and an application based on a service-oriented architecture

An SOA is a design model with concept of encapsulating application logic with services that interact via a specific and common communications protocol.
SOA is based on XML Web services and builds upon established XML technologies, with a focus on exposing loosely coupled services.

Following figure shows how an SOA alter the existing three-tier architecture by introducing a logical layer that establishes a common point of integration.

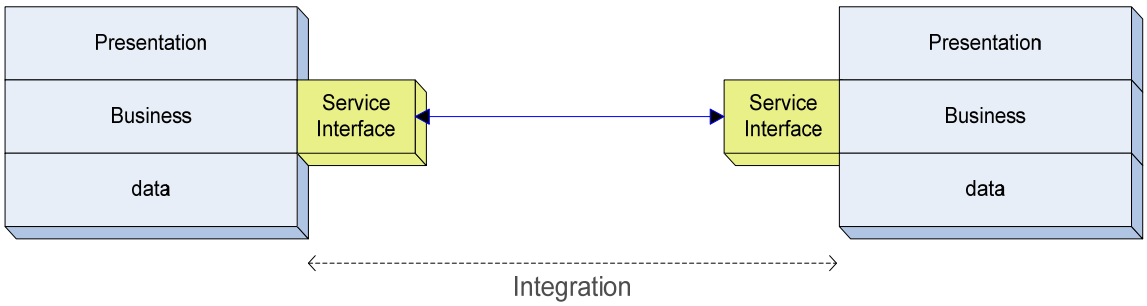


Figure 2-1 A logical representation of a service oriented architecture

When Web services are used for cross-application integration they establish themselves as part of the enterprise infrastructure.

Well-designed service-oriented environments will attempt security and scalability challenges with appropriate standard specification, rather than application specific solutions.

2.3 Web service roles

Services can have different roles based on different interaction scenarios. Depending on the context with which it's viewed, as well as the state of the currently

Service provider

When acting as a service provider, a Web service exposes a public interface through which it can be invoked by requestors of the service. In a client-server model, the service provider is mapped to the server and service consumer is mapped to client.

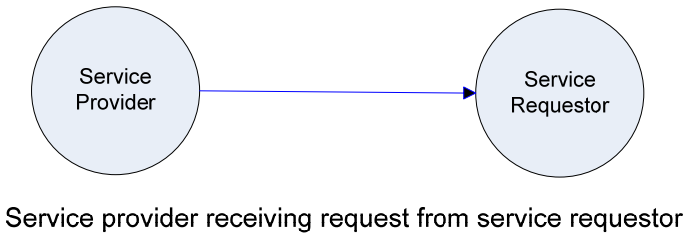


Figure 2-2 Sending request from service requestor

A service provider can also act as a service requestor.

Service requestor

A service requestor is the sender of a SOAP message or the software component (that attached to SOAP message) which requesting a specific Web service.

A service requestor can also act as a service provider.

Intermediary

It receives a message from a service requestor then forwards the message to a service provider.

Following figure explains how an intermediary processes a message; it too can act as a service provider and as a service requestor.

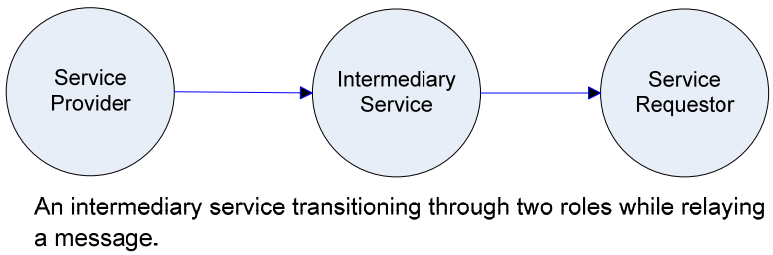


Figure 2-3 Intermediary Service

Intermediaries can exist in different forms. Some are passive, and simply relay or route messages. Some others are actively process a message before passing it on.

Intermediaries only can process and modify the message header.

Initial sender

Initial senders can be named as a service requestors (Figure 3.8) but the one who first create a request.

Ultimate receiver

This is the last Web service which receives a message. These kinds of services represent the final destination of a message.

2.4 General Web service terms and definitions

When messages are passed between two or more Web services, a differnet interaction scenarios can happened. For example initial sender can at as a service requestor, or ultimate receiver can act also as service provider or vice versa.

Message path

The route which a message travels have been named message path. It consists of one initial sender, one ultimate receiver, and can contain zero or more intermediaries.

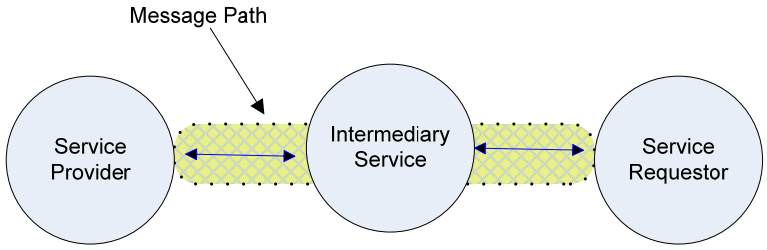


Figure 2-4 Message path

The transmission path that traveled by a message can be determined (dynamically) by routing intermediaries.
Figure 2-4 shows how a message is sent.

Message exchange patterns

Services which interact within a service-oriented environment participate in one of the predefined message exchange patterns.

Typical patterns include:

- request and response
- publish and subscribe
- fire and forget (one to one)
- fire and forget (one to many or broadcast)

The request and response pattern are common patterns especially when using in synchronous data exchange mode. The fire and forget and fire and forget patterns are used to for asynchronous data transferring.

Choreography

Rules that explains behavioral characteristics of how a group of Web services interact with each other is choreography. These rules include the sequence in which Web services can be invoked, conditions that apply to this sequence being carried out, and a usage pattern that further defines allowed interaction scenarios.

Check figure 2-5 for sequence of logically related services.

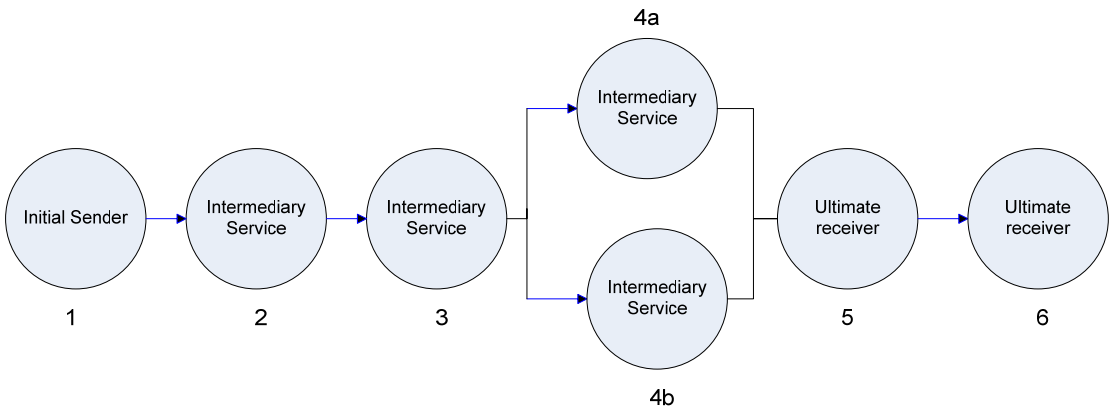


Figure 2-5 Choreography example

Activity

Message exchange patterns form the basis for service Activities (also known as tasks). An activity consists of a group of Web services that interact and collaborate to perform a function or a logical group of functions.

2.5 Service Models

Web services contain different form of standardization on different levels, including:

- application architecture
- enterprise infrastructure

- global data exchange

Despite their goal is to establishing a framework for information exchange, Web services themselves don't have standard shapes or sizes.

2.6 Web service description structure

An XML Web service is described through a stack of definition documents that constitute a service description.

This description will saved in a file named WSDL (Web service description language)

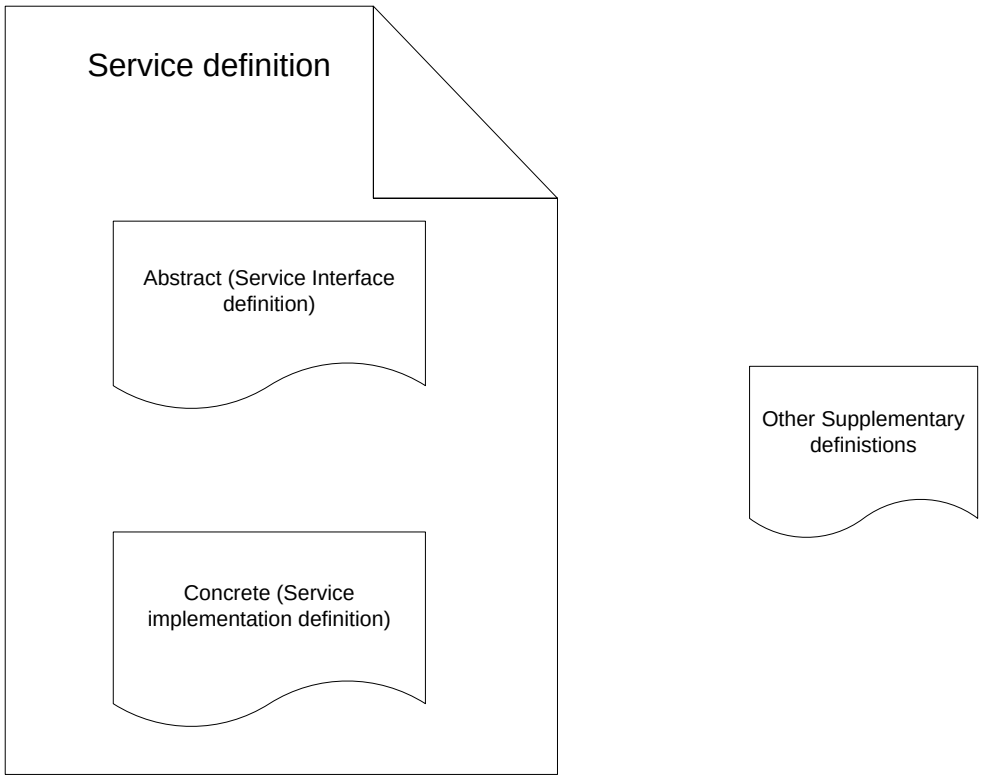


Figure 2-6 Web Service definition document

Abstract definition

Abstract definition is the logical description of a Web service interface, and it is independent from implementation details of web service.
Within a WSDL document, this abstract definition is made up of the interface and message constructs and can contains “types” construct, which is often classified separately.

Concrete definition

Implementation details about a Web service compose the concrete part of a WSDL document, It has the binding, service, and endpoint (or port) elements.

Service definition

Generally, Service definition is a root element of WSDL document which includes the interface (or abstract) and implementation (or concrete) definitions.

Service description

Service description consists of only a WSDL document that provides a service definition; it can include additional definition documents that they provides supplemental information.

Web Service framework

The W3C framework for Web services consists of a of three core XML components:

- Web Services Definition Language (WSDL)
- Simple Object Access Protocol (SOAP)
- Universal Description, Discovery, and Integration (UDDI)

These standard technologies, in context of service-oriented design, form XML-driven SOA.

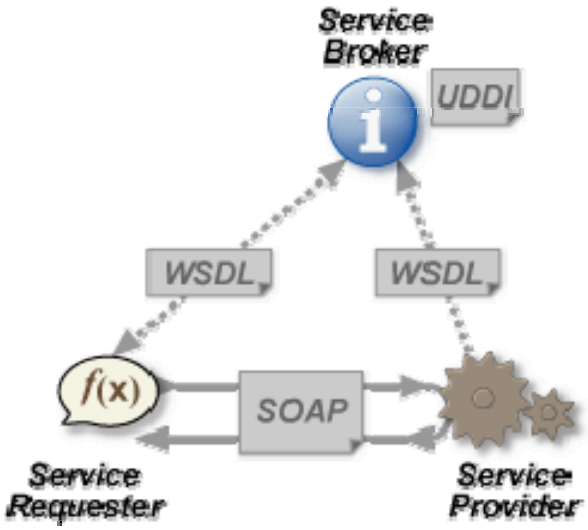


Figure 2-7 Web Service architecture

Web Services Definition Language (WSDL)

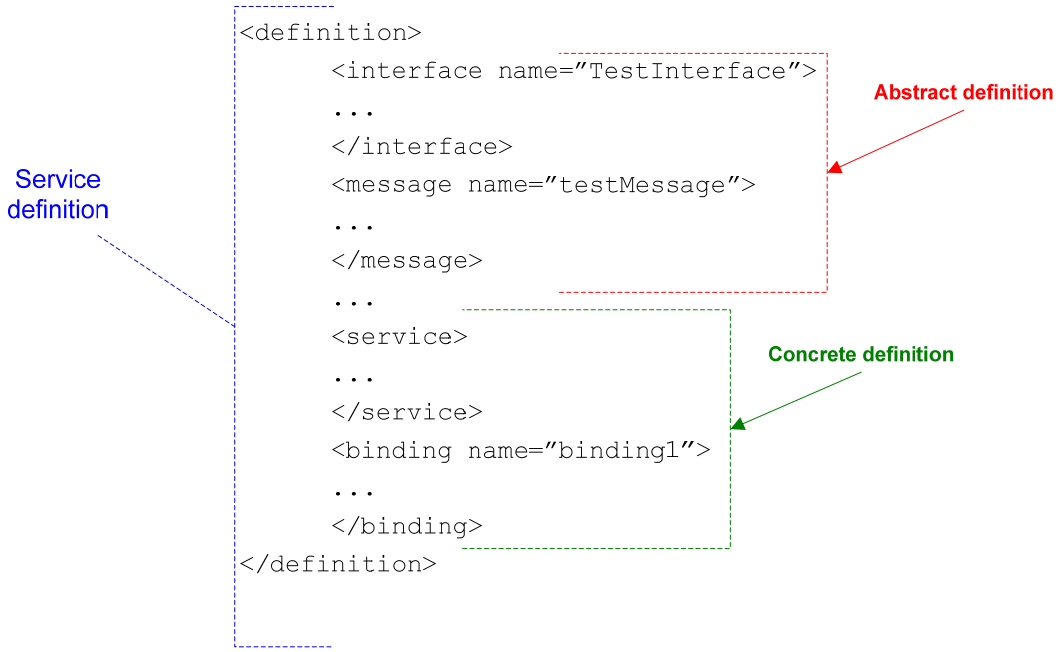
Web services need to be defined in a consistent manner so that they can be discovered by and interfaced with other services and applications. The Web Services Definition Language is a W3C specification providing the foremost language for the description of Web service definitions.

The integration layer introduced by the Web services framework establishes a standard, universally recognized and supported programmatic interface. WSDL enables communication between these layers by providing standardized endpoint descriptions.
We explain each of the elements representing WSDL.

```
<definition>
     <interface name=”TestInterface”>
     ...
     </interface>
     <message name=”testMessage”>
     ...
     </message>
     ...
     <service>
     ...
     </service>
     <binding name=”binding1”>
     ...
     </binding>
</definition>
```

A WSDL is host of primary following components:

- **portType or interface:** describes a set of messages that a service sends and/or receives.
- **message:** This element defines the data elements of an operation.
- **service:** Defines the ports supported by the Web service.
- **binding:** Describes a specific communication protocol for a portType element.



Abstract interface definition

Individual Web service interfaces are represented by WSDL interface elements. These constructs contain a group of logically related operations. In a component-based architecture**,** WSDL interface is comparable to a component interface. An operation is therefore the equivalent of a component method, as it represents a single action or function.

A Web service operation consists of a group of related input and output messages. The execution of an operation requires the transmission or exchange of these messages between the service requestor and the service provider.

```
<definition>
     <message name="BookInfo">
      ...
```

```
      </message>
      <interface name=”catalog”
            <operation name=”GetBook”>
                  <input name=”msg1” message=”BookInfo”>
            </operation>
      </interface>
</definition>
```

A message element can contain one or more input or output parameters that belong to an operation. Each part element defines one such parameter. It provides a name/value set, along with an associated data type. In a component-based architecture, a WSDL part is the equivalent of an input or output parameter (or a return value) of a component method.

```
<definitions>
      <message name="BookInfo">
         <part name=”title” type=”xs:string”>
                Field Guide
          </part>
         <part name=”author” type="xs:string”>
                Mr. T
          </part>
      </message>
</definitions>
```

- **interfaces** represent service interfaces, and can contain multiple
- **operations** represent a Web service function, and can reference multiple
- **messages** represent collections of input or output parameters, and can contain multiple parts
- **parts** represent either incoming or outgoing operation parameter data

Concrete (Implementation) definition

According to the implementation details, this part of WSDL explains concrete binding details for protocols, such as SOAP and HTTP and other implementation details.

Within a WSDL document, the service element represents one or more endpoints at which the Web service can be accessed. These endpoints consist of location and protocol information, and are stored in a collection of endpoint or port elements.

```
<definitions>
     <service name=”testservice”>
     <endpoint name=”TestEndpoint” binding=”TestBinding”>
               ...
     </endpoint>
     </service>
</definitions>
```

Now we need to define the invocation requirements of each of its operations.

```
<definitions>
     <service>
          <binding name=”” >
               <operation>
                    <input name="Msgln message="book" />
               </operation>
          </binding>
     </service>
</definitions>
```

The description of concrete information within a WSDL document can be summarized as follows:

- service elements is responsible for hosting collections of endpoints which represented by endpoint elements

- endpoint elements contain endpoint data, including physical address and protocol information

- binding elements associate themselves to web service operations

Supplementary constructs

It is an additional feature that has been used to provide data type support for Web service definitions is the "types" element. Its construct allows XSD schemas to be embedded or imported into the definition document.

```
<definitions>
     <types>
          <xsd:schema
               targetNameSpace="http://www.sopa.com/"
          ...
          </xsd:schema>
     </types>
</definitions>
```

Documentation element is optional in WSDL and used for providing information about the WSDL file.

```
<definitions>
     <documentation>
          This is for test.
     </documentation>
</definitions>
```

2.7 SOAP

SOAP's place in the web services technology stack is as a standardized packaging protocol for the messages shared by applications. The specification defines nothing more than a simple XML-based envelope for the information being transferred, and a set of rules for translating application and platform-specific data types into XML representations. SOAP's design makes it suitable for a wide variety of application messaging and integration patterns.

SOAP is XML. That is, SOAP is an application of the XML specification. It relies heavily on XML standards like XML Schema and XML Namespaces for its definition and function.

SOAP is acronym of the Service-Oriented Architecture Protocol, instead of the Simple Object Access Protocol.

SOAP facilitates synchronous (request and response) as well as asynchronous (process driven) data exchange models. With WSDL establishing a standard endpoint description format for applications, the document-centric message format is much more common.

SOAP message structure

SOAP message is in XML format and all elements gather in SOAP envelope. Let's take a brief look at the underlying structure of a SOAP message.

The root Envelope element of following sections:
body section and an optional header area.

```
<env:Envelope xmlns:env=http://www.w3.org/2003/05/soap-envelope>
     <env:Header>
          ...
     </env:Header>
     <env:Body>
          ...
     </env:Body>
</env>
```

The SOAP header is expressed using the Header construct, which can contain one or more sections or blocks.

```
<env:Envelope mlns:env="http://www.w3.org/2003/05/soap-envelope"~
     <env:Header>
          <n:shipping >
               DHL
          </n:shipping>
     </env:Header>
     <env:Body>
          ...
     </env:Body>
</env:Envelope>
```

Header blocks used commonly for:

- SOAP extensions implementation, such as those introduced by second-generation specifications.
- identification of target SOAP intermediaries
- providing some meta information about the SOAP message

While a SOAP message travel along a message path, intermediaries may process add and remove data in/from SOAP header blocks. Header is an optional part of a SOAP message, and it used for carrying header blocks.

The mandatory part of a SOAP message is the body. As represented by the Body element, this section acts as a container of the data included in the SOAP message. Data within the SOAP body is simetimes referred to as the payload or payload data.

```
<env:Envelope mlns:env=http://www.w3.org/2003/05/soap-
envelope>
     <env:Header>
          ...
     </env:Header>
     <env:Body>
          <x:Book xmlns:x="http://irmw.examples.ws/">
               <x :Title>Service-Oriented Architecture
                         A Field Guide to Integrating XML,
                         and Web services
               </x:Title>
          </x:Book>
     </env:Body>
</env:Envelope>
```

The body construct can also be used to host exception information within nested Fault elements. Fault part can locate in standard data payloads, fault is often sent separately in response messages to explaining error conditions.

The Fault structure consists of a system elements used to identify characteristics of the exception.

```
<env:Envelope mlns:env=http://www.w3.org/2003/05/soap-
envelope>
     <env:Body>
          <env:fault>
               <env:code>
                    <env:value>
                     Env: version mismatch
                    <env:value>
               </env:code>
               <env:reason>
                    <env:text xml:lang=”en”>
                         Version do not match
                    </env:text>
               </env:reason>
          </env:fault>
     </env:Body>
</env:Envelope>
```

2.8 UDDI

One of the fundamental components f a service-oriented architecture is a mechanism for Web service descriptions to be discovered by potential requestors. To establish this part of a Web services framework, a central directory to host service descriptions is required. Such a directory can become an integral part of an organization or an Internet community, so much so, it is considered an extension to infrastructure.

This is why the Universal Description, Discovery, and Integration specification has become increasingly important. A key part of UDDI is the standardization of profile records stored within such a directory, also known as a registry. Depending on who the registry is intended for, different implementations can be created.
A **public business registry** is a global directory of international business service descriptions.

Instances of this registry are hosted by large corporations (also referred to as node operators) on a series of dedicated UDDI servers. UDDI records are replicated automatically between repository instances. Some companies also act as UDDI registrars, allowing others to add and edit their Web service description profilesm. The public business registry is complemented by number service market places offering generic Web services for sale or lease.

Private registries are service description repositories hosted within an organization Those authorized to access this directory may include select external business partners. A registry restricted to internal users only can be referred to as an Internal registry.

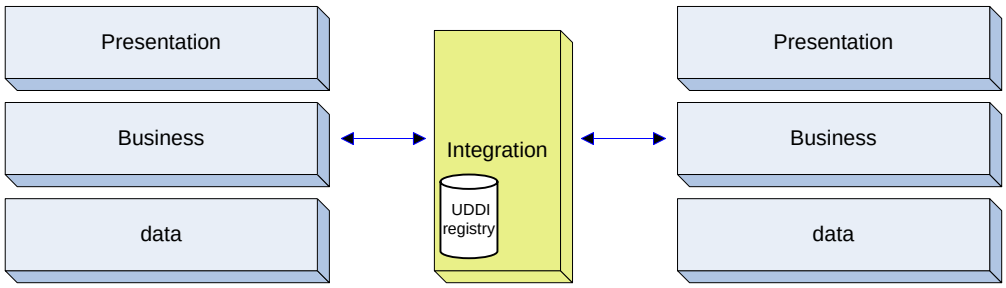


Figure 2.8: service description in UDDI registry(the private one)

UDDI registries organize registry entries using six primary types of data:

- business entities
- business services
- specification pointers
- service types
- business relationships
- subscriptions

Business entity data, as represented by the business Entity element, provides profile information about the registered business, including its name, a description, and aunique identifier.

Following shows a sample XML document which contains a businessEntity construct.

```
<businessEntity xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      mlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      businessKey="e9355d51-32ca-49cf-8eb4-lce59a£b£4a7"
      operator="Microsoft Corporation"
      authorizedName="T. E."
      xmlns="urn:uddi-org:api-v2">

<discoveryURLs>
      <discoveryURL useType="business entitiy">
            http://test.uddi.microso£t.com/discovery
            ?businesskey=e9355d51-32ca-49~£-8eb4-1ce59afb£4a7
      </discoveryURL>
</discoveryURLs>
```

```
<name xml:lang="en">
      XMLTC Consulting Inc.
</name>

<description xml:lang="en">
      XMLTC is a test eBusiness solution for corporation's agencies.
      It offers a range of design, development and integration
      services.
</description>

<businessServices>
      <businessservice
            serviceKey="leeec£al-6f99-46Oe-a392-8328d38b763a"
            businessKey="e9355d51-32ca-49cf-8eb4-lce59afbf4a7">

            <name xml:lang="en-us"> Test Home Page </name>

            <bindingTemplates>
                  <bindingTemplate
                        bindingKey="48b02d40-0312-4293-a7f5-
44449cal90984"
                        serviceKey="leeecfal-6f99-46Oe-a392-
8328d38b763a">

                  <description xml:lang="en">
                        Entry point in the XMLTC Test Web site now a
                        number of resource and sites can be accessed
                        from here.
                  </description>

      <accespoint URLType="http">http://www.xmltc.com</accesspoint>
                  <tModelInstanceDetails/>
            </bindingTemplate>

            <categoryBag>
                  <keyedrefernce tModelKey="uuid:calcf26d-9672-4404-
                  9d70-39b756e62ab4"  keyname="Namespace"
                  keyValue="namespace"/>
```

```
            </categoryBag>
      </businessservice>
</businessservices>

</businessentity>
```

Let's take this document apart to study the individual constructs.

```
<businessEntity xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                businessKey="e9355d51-32ca-49cf-8eb4-1ce59afbf4a7"
                operator="Microsoft Corporation"
                authorizedName="T. E."
                xmlns="urn:uddi-org:api-v2">
```

When the reference registered XMLTC Consulting Inc. it was given a unique identifier of e453453-32ca-49cf-8eb4-lce59afbf4a7, which was then assigned to the businessKey attribute of the businessEntity parent element.
Microsoft acted as the operator to provide an instance of the UDDI registry, you can check its name is displayed in the businessEntity element's operator attribute.

The discoveryURL element identifies the address used to locate this XML document.

The name element simply contains the official business name.

```
<name xml:lang="en">
      XMLTC Test Inc.
</name>
```

Business Service records representing the actual services offered by the registered business are nested within the businessEntity construct.

A business service is identified with a unique value assigned to the serviceKey attribute. Its parent businessEntity element is referenced by the businessKey attribute.

The only business service associated with this business entity is the business's Web site home page, as identified by the name element.

```
<name xml:lang="en-us">
      Corporate Home Page
</name>
```

Each business service provides specification pointers, known as binding templates, These records consist of addresses linking the business service to implementation information. Using service pointers, a developer can learn how and where to physically bind to a Web service.

The bindingTemplate construct displayed in the preceding example establishes the location and description of the service using the accesspoint and description elements.

Various categories can be assigned to business services. In our example, the URL we identified has been classified as a namespace using the keyedRef erence child element of the categoryBag construct.

```
<categoryBag>
       <keyedrefernce tModelKey="uuid:calcf26d-9672-4404-9d70-
      39b756e62ab4"  keyname="Namespace" keyValue="namespace"/>
</categoryBag>
```

There is no formal relationship between UDDI and WSDL. A UDDl registry provides a means of pointing to service interface definitions through the use of a Model.

You can interface programmatically with a UDDI registry. For example, we could make a SOAP message to search a company by name criteria with the following message data (payload):

```
<find_business xmlns=”urn:uddi-org:api-v3”>
      <findQualifiers>
            <findQualifier>
                  Uddi:uddi.org.findQualifier:exactMatch
            </findQualifier>
```

```
        </findQualifiers>
        <name>
              XMLTC Consulting Inc.
        </name>
</find_business>
```

2.9 Representational State Transfer

This is new form of web services because it has lots of usage it have been take in this chapter and a summary about REST web services have been explained here.

RESTful Web services attempt to emulate HTTP and similar protocols by constraining the interface to a set of well-known, standard operations (e.g., GET, PUT, DELETE). Here, the focus is on interacting with stateful resources, rather than messages or operations.
RESTful Web services can use WSDL to describe SOAP messaging over HTTP, which defines the operations, or can be implemented as an abstraction purely on top of SOAP (e.g., WS-Transfer).
REST strictly refers to a collection of architectural principles. The term is also often used in a loose sense to describe any simple interface that transmits domain-specific data over HTTP without an additional messaging layer such as SOAP or session tracking via HTTP cookies. These two meanings can conflict as well as overlap. It is possible to design any large software system in accordance with Fielding's REST architectural style without using the HTTP protocol and without interacting with the World Wide Web. It is also possible to design simple XML+HTTP interfaces that do not conform to REST principles, and instead follow a Remote Procedure Call (RPC) model. The two different uses of the term REST cause some confusion in technical discussions.

Chapter 3

Java Web service client models

Information of this chapter is coming from:

- The Java API for XML-Based Web Services (JAX-WS) 2.0 Final Release
- JSR-110: Java™ APIs for WSDL (JWSDL) Version 1.2
- SOAP's Two Messaging Styles By: Rickland Hollar
- JAX-RPC vs JAX-WS By: Russell Butek and Nicholas Gallardo
- Java Web Services Architecture by James McGovern, Sameer Tyagi, Michael Stevens and Sunil Matthew
- J2EE Web Services By Richard Monson-Haefel

JAX-RPC is technology which has been used in J2EE 1.4 for web service handling, Currently the newest Java technology that have been designed for web service and SOA operability is JAX-WS (for JEE 5 and JSE 6). In this chapter I explain the client for web service invocation of both in more detail and try to show them with example.
At the end I will make comparison between JAX-RPC and JAX-WS and explain the advantages of JAX-WS over JAX-RPC, in this project I used JAX-WS because the component should be JEE 5 compatible and JAX-RPC is not JEE5 compatible otherwise WSIF which is based on J2EE 1.4 is good candidate for implementing such a requirement.

What Is done here is to design a and implement a generic client base on JAX-WS 2.0, a SOAP message in format SOAP 1.1 is created, JAX-WS also support 1.2 but SOAP 1.1 is used.

3.1 Few notes about JAX-WS 2.0

Since the first release of JAX-RPC 1.0, some new specifications and new versions of java web service components have been released; the latest one up to now is JAX-WS 2.0.
The comparison will be provided in detail but for now note that JAX-WS 2.0 relates to these specifications and standards as follows:

JAXB
Due primarily to scheduling concerns, JAX-RPC 1.0 defined its own data binding facilities. With the release of JAXB 1.0 there is no reason to maintain two separate sets of XML mapping rules in the Java platform. JAX-WS 2.0 will delegate data binding-related tasks to the JAXB 2.0 specification that is being developed in parallel with JAX-WS 2.0.
JAXB 2.0 will add support for Java to XML mapping, additional support for less used XML
schema constructs, and provide bidirectional customization of Java , XML data binding. JAXWS 2.0 will allow full use of JAXB provided facilities including binding customization and optional schema validation.

SOAP 1.2
Whilst SOAP 1.1 is still widely deployed, it's expected that services will migrate to SOAP now that it is aW3C Recommendation. JAX-WS 2.0 will add support for SOAP 1.2 whilst requiring continued support for SOAP 1.1.
There is really not a lot of difference, from a programming model perspective, between SOAP 1.1 and SOAP 1.2.

WSDL 2.0

TheW3C is expected to progress WSDL 2.0. JAX-WS 2.0 will add support for WSDL 2.0 whilst requiring continued support for WSDL 1.1.

WS-I Basic Profile 1.1

JAX-RPC 1.1 supports WS-I's Basic Profile (BP) 1.0. BP 1.1 is newer and these new profiles clarify some minor points.

JAX-WS 2.0 supports these profiles. For the most parts, the differences is very small and do not affect the Java programming model.

The exception is attachments. WS-I not only cleared up some questions about attachments, but they also defined their own XML attachment type: wsi:swaRef.

JAX-RPC 1.1 added support for WS-I Basic Profile 1.0. WS-I Basic Profile 1.1 is expected to be use instead of 1.0.

A Metadata Facility for the Java Programming Language JAX-WS 2.0 will define the use of Java annotations to simplify the most common development scenarios for both clients and servers.

Implementing Enterprise Web Services This feature defined jaxrpc-mapping-info deployment descriptor provides deployment time Java , WSDL mapping functionality. JAX-WS 2.0 will complement this mapping functionality with development time Java annotations that control Java , WSDL mapping.

Web Services Security (JSR 183) JAX-WS 2.0 will align with and complement the security APIs

Asynchrony JAX-WS 2.0 will add support for client side asynchronous operations.

Non-HTTP Transports JAX-WS 2.0 will improve the separation between the XML message format and the underlying transport mechanism to simplify use of JAX-WS with non-HTTP transports.

3.2 Web Service Client for JAX-RPC 1.1

JAX-RPC client can be written to invoke services using one of the following three mechanisms:

- Dynamic invocation interface
- Dynamic proxies
- Stub

3.2.1 Stub

In stub form clients will locate the service endpoint by specifying a URI, and then invoke the methods on a local object, a stub represents the remote service. JAX-RPC stubs, or proxies, are different from RMI-IIOP stubs. Keep following statement in mind:

A stub is never required to be downloaded or distributed to clients.

A client is not a required artifact on the client side. The end result of the invocation is that the required SOAP envelope must be sent on the transport protocol. The client can be written in a completely different programming language, as shown later in the JAX_RPC Interoperability section.

- The stub is implemented in Java and is relevant only for a JAX-RPC client runtime.
- A stub can be dynamically generated by the client side at runtime.
- A stub is specific to the client runtime.
- A stub is specific to a protocol and transport.
- A stub must implement the javax.xml.rpc.Stub interface.

Sometime stub is referred to as static invocation; because the stub must know the remote interface about the service at compile time. Like RMI in java stub must have the class file which represents the remote interface and the implementation available for stub generation.

The client doesn't need the WSDL file at runtime. Stubs are specific to a particular runtime and are not portable across different vendor implementations.

Example of JAX-RPC Client using stub:

```
import java.util.Date;

public class StubClient {
    public static void main(String [] args) throws Exception {

   String
endpoint="http://127.0.0.1:8080/billpayservice/jaxrpc/BillPay";
    String namespace = "http://www.flutebank.com/xml";
    String wsldport = "BillPayPort";
    BillpayserviceImpl serviceproxy= new BillpayserviceImpl();
    BillPayStub stub=(BillPayStub)(serviceproxy.getBillPayPort());

stub._setProperty(javax.xml.rpc.Stub.ENDPOINT_ADDRESS_PROPERTY,endpo
int);
    PaymentConfirmation conf= stub.schedulePayment(new Date(),
                                                    "my account
as test account", 210);
      System.out.println("Payment was scheduled "+
conf.getConfirmationNum());
      PaymentDetail detail[]=stub.listScheduledPayments();
     for(int i=0;i<detail.length;i++) {
           System.out.println("Payee name "+
detail[i].getPayeeName());
           System.out.println("Account "+ detail[i].getAccount());
           System.out.println("Amount "+ detail[i].getAmt());
           System.out.println("Will be paid on "+
detail[i].getDate());
           }
     double lastpaid= stub.getLastPayment("my cable tv provider");
     System.out.println("Last payment: "+ lastpaid);
   }
}
```

Before using a stub, a client must first obtain a reference to it. The following code shows the mechanism another JAX-RPC vendor might use:

```
InitialContext ctx = new InitialContext();
Billpayservice service =
      (Billpayservice) ctx.lookup("myserver:soap:Billpayservice");
BillPay bill1 = service. getBillPayPort ();
Stub stub= (Stub) bill1;
```

The stub can be configured by passing it name and value pairs of properties. The javax.xml.rpc.Stub interface defines four standard properties to configure the stub, using the stub. _setProperty(java.lang.String name, java.lang.Object value) method:

- javax.xml.rpc.security.auth.username. Username for authentication.
- javax.xml.rpc.security.auth.username.password. Password for authentication.
- javax.xml.rpc.service.endpoint.address. Optional string for the endpoint service.

- javax.xml.rpc.session.maintain. Use java.lang.Boolean to indicate that the server needs to maintain session for the client

3.2.2 DII (Dynamic Invocation Interface)

The next way a consumer can access a service is using of dynamic invocation interface (DII).

DII is a concept that, like most other things in JAX-RPC, should be familiar to CORBA developers. Unlike static invocation, which requires that the client application include a client stub, DII enables a client application to invoke a service whose data types were unknown at the time the client was compiled.

This lets a client to discover interfaces dynamically (at runtime instead of compile time).

JAX-RPC supports DII with the javax.xml.rpc.Call interface. A Call object can be created on a javax.xml.rpc.Service using the port name and service name. Then, during runtime, the following details are set:

- Operation to invoke
- Port type for the service
- Address of the endpoint
- Name, type, and mode (in, out, inout) of the arguments
- Return type

This information is derived by looking at the WSDL file for the service. For instance, the service name=" Billpayservice "> element, the portname is the port name="BillPayPort " element, and so on.

Example: Client using DII

```
import javax.xml.namespace.QName;
import javax.xml.rpc.Call;
import javax.xml.rpc.Service;
import javax.xml.rpc.ParameterMode;
import javax.xml.rpc.ServiceFactory;

public class DIIClient_NoWSDL{
  public static void main (String[] args) throws Exception {

     String
endpoint="http://127.0.0.1:9090/billpayservice/jaxrpc/BillPay";
```

```
     String namespace = "http://www.flutebank.com/xml";
     String schemaNS = "http://www.w3.org/2001/XMLSchema";
     String serviceName = "Billpayservice";

     ServiceFactory myfactory = ServiceFactory.newInstance();
     // the Billpayservice service does not exist
     // (no stub, skeleton, or Service was generated by xrpcc)
     // but createService will return a Service object
     // that can be used to create the dynamic call

    Service service = (Service)myfactory.createService
                              (new QName(namespace,serviceName));

     QName portName = new QName(namespace," BillPayPort");
     QName operationName = new QName(namespace," getLastPayment");
     Call call = service.createCall(portName, operationName);
     call.setTargetEndpointAddress(endpoint);
     call.setProperty(Call.ENCODINGSTYLE_URI_PROPERTY,
                     "http://schemas.xmlsoap.org/soap/encoding/");

     QName paramType = new QName(schemaNS, "string");
     QName returnType = new QName(schemaNS, "double");

     call.addParameter("String_1", paramtype, ParameterMode.IN);
     call.setReturnType(returntype);

     Object[] params = {"Test for cable tv company provider"};
     Object lastpaid= (Double)call.invoke(params);
     System.out.println("Last payment: "+ lastpaid);
    }
}
```

The client wraps the DII request in a Call object. DII can be used directly, by passing these values (port, operation, location, and part information) to the Call, or indirectly, by passing the WSDL to the Call.
In indirect DII, only the port and operation names are knowm at compile time. The runtime will determine the type information about the part and location, based on the WSDL. In this case, the parameters and return types do not need to be configured using the addParameter or setReturnType method.

Example: Client using DII indirectly, where all parameters are not known (WSDL is dynamically inspected)

```
public class DIIClient_WSDL{

    public static void main(String[] args) throws Exception {

  String wsdllocation=
http://127.0.0.1:9090/billpayservice/billpayservice.wsdl";

    String namespace = "http://www.flutebank.com/xml";
    String serviceName = "Billpayservice";

    ServiceFactory myfactory = ServiceFactory.newInstance();
    Service service = (Service) myfactory.createService
```

```
                    (new URL(wsdllocation),new
QName(namespace,serviceName));

    QName portName = new QName(namespace," BillPayPort");
    QName operationName = new QName(namespace," getLastPayment");
    Call call = service.createCall(portName, operationName);
    Object[] params = {"my cable tv provider"};
    Object lastpaid= (Double)call.invoke(params);
    System.out.println("Last payment was "+ lastpaid);
   }
}
```

3.2.3 Dynamic Proxies

The JAX-RPC specification also specifies a third way for clients to access services: using the concept of dynamic proxy classes available in the standard J2SE Reflection API (the java.lang.reflect.Proxy class and the java.lang.reflect .InvocationHandler interface). A dynamic proxy class implements a list of interfaces specified at runtime. The client can use this proxy or façade as though it actually implemented these interfaces, although it actually delegates the invocation to the implementation.

Classes allowing any method on any of these interfaces can be called directly on the proxy (after casting it). Thus, a dynamic proxy class is used to create a type-safe proxy object for an interface list without requiring pregeneration of the proxy class, as you would with compile-time tools.

Example: Client using dynamic proxies

```
import javax.xml.namespace.QName;
import javax.xml.rpc.Service;
import javax.xml.rpc.ServiceFactory;
// java classes
import java.util.Date;
import java.net.URL;
// Interface class
import com.flutebank.billpayservice.BillPay;

public class DynamicProxyClient {
  public static void main( String[] argv) throws Exception{
     String namespace = "http://www.flutebank.com/xml";
     String wsldport = "BillPayPort";
     String wsdlservice = "Billpayservice";
     String wsdllocation =
```

```
"http://127.0.0.1:8080/billpayservice/billpayservice.wsdl";
     URL wsdlurl = new URL(wsdllocation);
     ServiceFactory factory = ServiceFactory.newInstance();
     Service service = factory.createService(wsldurl,
                                                new
QName(namespace, wsdlservice));
// make the call to get the stub for this service
        BillPay stub = (BillPay) service.getPort(new
QName(namespace,wsldport) , BillPay.class);
// call methods on the service
        double lastpaid= stub.getLastPayment("test cable tv co.
provider");
        System.out.println("Last payment: "+ lastpaid);
    }
}
```

Note that there is no compile-time stub generation. CORBA developers will see the similarity in the above code with its counterpart:

```
BillPay stub = (BillPay)
PortableRemoteObject.narrow(initial.lookup("Billpayservice"),BillPay
.class);
```

3.3 Web Service Client for JAX-WS 2.0

In JAX-WS there are two methods available for invoking client, one is via creating proxy object, the other one is to create SOAP message request and send it via request Dispatcher to service, then service after execution sends back SOAP response.

So we have following cases to consume Web Service as client (Service Requestor):

- SOAP XML level
- Proxy object creation

Service object is created base on the methods we choose to implement if it is going to be SOAP message service is dynamic and if it is proxy object service going to be static.

3.3.1 Dispatch (XML level Client)

XML Web Services use XML SOAP messages for communication between services and service clients. The higher level JAX-WS APIs are designed to hide the details of converting between Java method invocations and the corresponding XML messages, but in some cases operating at the XML message level is desirable. The Dispatch interface provides support for this mode of interaction, and all my JAX-WS usage is based on Dispatch

Dispatch supports two usage modes, identified by the constants javax.xml.ws.Service.Mode.MESSAGE and javax.xml.ws.Service.Mode.PAYLOAD respectively:

1. **Message**

In this mode, client applications work directly with protocol-specific message structures.

2. **Message Payload**

In this mode, client applications work with the payload of messages rather than the messages themselves. E.g., when used with a SOAP protocol binding, a client application would work with the contents of the SOAP Body rather than the SOAP message as a whole.

Dispatch is a low level API that requires clients to construct messages or message payloads as XML and requires an intimate knowledge of the desired message or payload structure. Implementations are required to support the following types of object:

javax.xml.transform.Source

Use of Source objects allows clients to use XML generating and consuming APIs directly. Source objects may be used with any protocol binding in either message or message payload mode. When used with the HTTP binding in payload mode, the HTTP request and response entity bodies must contain XML directly or a MIME wrapper with an XML root part. A null value for Source is allowed to make it possible to invoke an HTTP GET method in the HTTP Binding case.

JAXB

Objects Use of JAXB allows clients to use JAXB objects generated from an XML Schema to create and manipulate XML representations and to use these objects with JAX-WS without requiring an intermediate XML serialization. JAXB objects may be used with any protocol binding in either message or message payload mode. When used with the HTTP binding in payload mode, the HTTP request and response entity bodies must contain XML directly or a MIME wrapper with an XML root part. When used in message mode, if the message is not an XML message a WebServiceException will be thrown.

javax.xml.soap.SOAPMessage Use of SOAPMessage objects allows clients to work with SOAP messages using the convenience features provided by the java.xml.soap package. SOAPMessage objects may only be used with Dispatch instances that use the SOAP binding in message mode.

javax.xml.soap.SOAPMessage

Use of SOAPMessage objects allows clients to work with SOAP messages using the convenience features provided by the java.xml.soap package. SOAPMessage objects only have been used with Dispatch instances that use the SOAP binding in message mode.

javax.activation.DataSource

Use of DataSource objects allows clients to work withMIME-typed messages. DataSource objects may only be used with Dispatch instances that use the HTTP binding in message mode.

I used dispatch method in my project is so I don't provide example here for checking example of dispatch check chapter 4 "Execute service" class.

3.3.2 Proxy (Object level client)

Proxies provide access to service endpoint interfaces at runtime without requiring static generation of a stub class. See java.lang.reflect.Proxy for more information on dynamic proxies as supported by the JDK.

A proxy is created using the getPort methods of a Service instance:

T getPort(Class<T>) Returns a proxy for the specified SEI, the Service instance is responsible for selecting the port (protocol binding and endpoint address).

T getPort(QName port, Class<T>) Returns a proxy for the endpoint specified by port. Note that the namespace component of port is the target namespace of the WSDL definitions document.

The service Endpoint Interface parameter specifies the interface that will be implemented by the proxy. The service endpoint interface provided by the client needs to conform to the WSDL to Java mapping rules. Creation of a proxy can fail if the interface doesn't conform to the mapping or if any WSDL related metadata is missing from the Service instance.

The following example shows the use of a proxy to invoke a method (getLastTradePrice) on a service endpoint interface (com.example.StockQuoteProvider). Note that no statically generated stub class is involved.

Example: Following example shows a Static client creation for consuming service:

```
javax.xml.ws.Service service = ...;
com.example.StockQuoteProvider proxy = service.getPort(portName,
com.example.StockQuoteProvider.class)
javax.xml.ws.BindingProvider bp =
(javax.xml.ws.BindingProvider)proxy;
Map<String,Object> context = bp.getRequestContext();
context.setProperty("javax.xml.ws.session.maintain", Boolean.TRUE);
proxy.getLastTradePrice("ACME");
```

3.4 Comparison between JAX-RPC and JAX-WS

Here in the following I lists the advantages of JAX-WS over JAX-RPC base on technology or specification:

SOAP 1.2

JAX-WS and JAX-RPC both support SOAP 1.1. JAX-WS additionaly supports SOAP 1.2.

XML/HTTP

The WSDL 1.1 specification defined an HTTP binding, which is a means by which you can send XML messages over HTTP without SOAP. JAX-RPC ignored the HTTP binding. JAX-WS adds support for it.

WS Basic Profiles

JAX-RPC supports WS-I's Basic Profile (BP) version 1.0. JAX-WS supports BP 1.1. (WS-I is the Web services interoperability organization.)

New Java features

1. JAX-RPC maps to Java 1.4. JAX-WS maps to Java 5.0. JAX-WS relies on many of the features new in Java 5.0.
2. Java EE 5, the successor to J2EE 1.4, adds support for JAX-WS, but it also retains support for JAX-RPC, which could be confusing to today's Web services novices.

Chapter 4

Generic Client specification

Information of this chapter are coming from:

- The Java API for XML-Based Web Services (JAX-WS) 2.0 Final Release
- JSR-173: Streaming API For XML
- JSR-110: Java™ APIs for WSDL (JWSDL) Version 1.2
- Eclipse Rich Client Platform: Designing, Coding, and Packaging Java™ Applications By Jeff McAffer, Jean-Michel Lemieux
- Note on the Eclipse Plug-in Architecture, By Azad Bolur
- SWT/JFace in Action by Matthew Scarpino, Stephen Holder, Stanford Ng, Laurent Mihalkovic
- XML in Nutshell 3rd Edition By Elliotte Rusty Harold, W. Scott Means

In this chapter we will focus on explaining the implemented component in detail. We will start with Architecture and design, and then in design part we will go further inside the technical details like the used technologies etc. Afterward a user manual or tutorial for user of this component in "abstract mode" and as "pipeline client in SOPA framework" will be provided. Before startup, some explanation about the usage of this component is provided.

In Service oriented computing when an application requests to consume a service, it is necessary to prove a way to access that service. Like most message oriented technology client is responsible to handle connection to the service. Client establishes appropriate form of access to required service.

Our requirement is to create a client which is just based on WSDL. This client should establishes appropriate connection and make service call to service provider as service requestor.
This means we need a general way to create one client for any kind of web services.
Please note that client will located on service consumer part and acts as service requestor and not service provider, if it acts as service provider it is service not client.

In architecture part this component is explained very general and doesn't go into detail. Detailed information about classes and their related method will be provided in design and implementation part of this chapter.

4.1 Architecture

In this section the functional behavior and specification of generic client will be explained, it is not important if it will be used as an abstract component or if it will be used as a client for invoking pipeline in SOPA framework, in both cases the functional behavior is same.

The following figure depicts the required steps that have been mentioned in previous chapters:

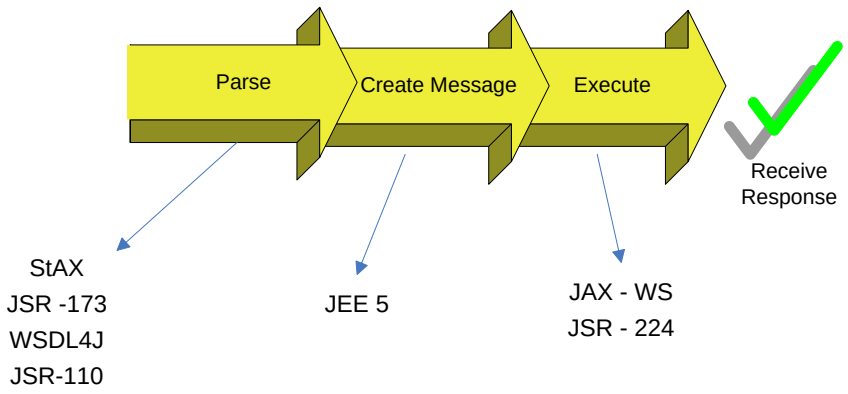


Figure 4-1 generic client architecture phases

The figure will give you an overall view of generic client (or service requestor) that will do its job in three Steps, first step is to parse a given WSDL file, parse process will be done on abstract part (not concrete part) of WSDL file. After parsing abstract part of the WSDL file, generic client tries to create a SOAP request message based on the Service Name, port name, operation and related operation arguments. As a convention from now on we will use "message creation", instead of "SOAP message creation".

As noted before, SOAP is a message format of Service oriented communication, and client must send its request in SOAP message format which has been described in detail in chapter 3. Then the client will send this SOAP request to appropriate web service based on the chosen service Name, port Name, etc and waits for results. The results will come as SOAP message response. Then result is shown to user or given to another part of SOPA framework for further administration.

If we take an overall overview of project, the following three use cases are expected from this project:

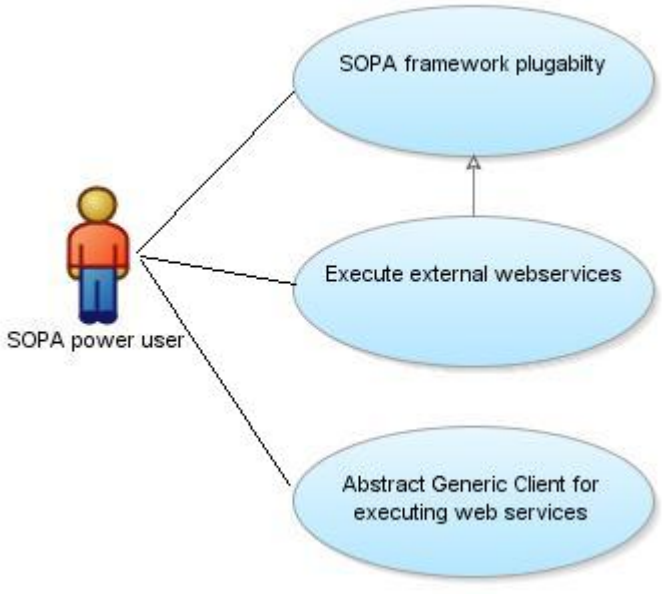


Figure 4-2 Generic client use case

Now we will go inside each step and explain them in more details.

4.1.1 WSDL Parsing Phase

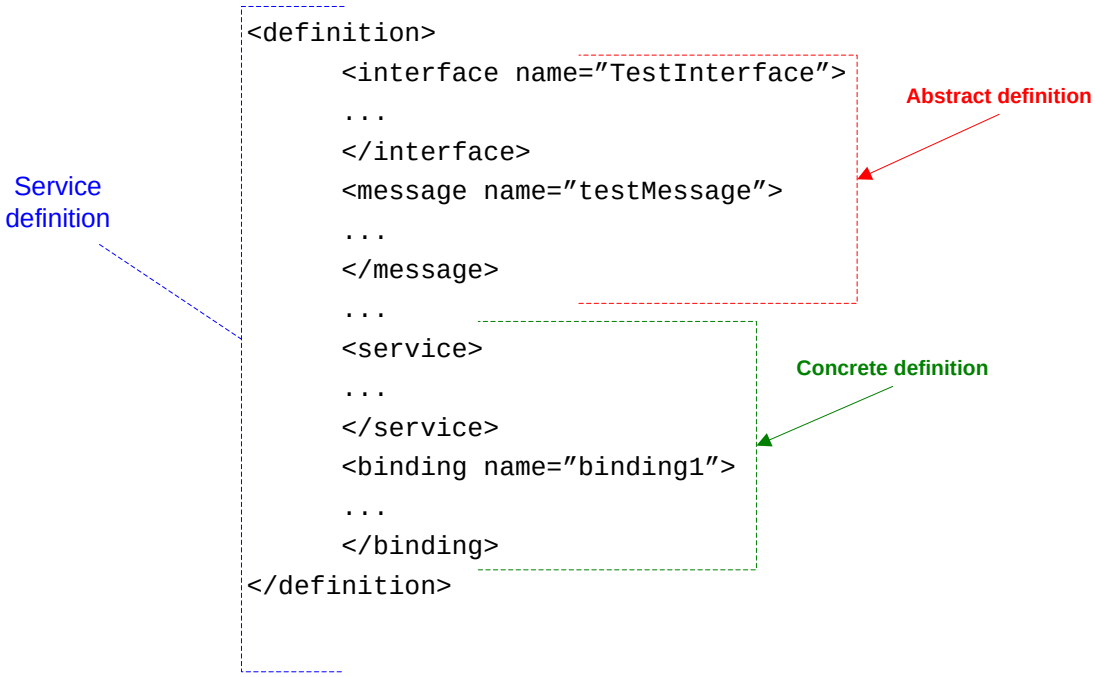
This section explains the process of parsing the given WSDL file. WSDL structure is rather complex to parse. SAX is the fastest java parser currently available but it is not useful for parsing complex XML-structured file like WSDL, DOM is good candidate for parsing complex XML structure but since it parses document tree based and not event based it is not fast. We have used WSDL4J libraries for parsing WSDL which is available from IBM. It is the current implementation of JWSDL or JSR–110.
JWSDL provides a standard set of Java APIs for representing, manipulating, reading and writing WSDL (Web Services Description Language) documents, including an extension mechanism for WSDL extensibility.

According to JSR – 110 Java APIs for WSDL [JWSDL] is an API for representing WSDL documents in Java. More specifically, JWSDL v1.2 represents WSDL v1.1 as described by a W3C Note and XML Schema, with the relaxed extensibility permitted by WS-I Basic Profile 1.1.

The WSDL4J and JWSDL provide a rich set of features, but since I need to create SOAP request message, more detailed information from WSDL file is required. For example to extract the "service target namespace" of definition element and similar information, a more useful and optimized technology like StAX is used.

According to JSR -173 StAX is bi-directional Streaming API for reading and writing XML. The Streaming API for XML gives parsing control to the programmer by exposing a simple iterator based API and an underlying stream of events. Methods such as next() and hasNext() allow an application developer to ask for the next event (pull the event) rather than handling the event in a callback. This gaves a developer more procedural control over the processing of the XML document. The Streaming API also allows the programmer to stop processing the document, skip ahead to sections of the document, and get subsections of the document. For more information about parsers you can refer to XML process methods in last chapters.

Now let's take a look at WSDL structure again



JAX-WS libraries required information (to call an operation on service) are coming from abstract part of WSDL file, because as I noted before concrete part is for implementation and not definition of services, abstract part is for logical definition of services.

Via WSDL4J library and the implemented StAX-based parser methods, application processes the WSDL in order to perform the user selected operation and fill its arguments.

User will follow the WSDL parsing process as described below:

Service Name → Port Name → Operation → Fills operation's arguments

The first thing that should have been specified in generic client is "the selected service that user wants to work with". As mentioned in the restriction section (see Appendix), it is assumed that WSDL file has only one definition element.

Then after user chooses the Service, application goes one step forward and user should choose which port to operate with. Ports are specified in WSDL files by their names, after user choosing the port he/she should choose operation which is the last step before initializing the operation arguments.

The selection of service, port and operations is required because one WSDL file could have more than one service, each service could have more than one port and each port could have different operations with same operation name but absolutely different functionality so it is not possible to directly go to operation list and choose one operation, for example we can have one WSDL file with two operation both named QueryUserInformation one is for querying user's information of application X and the other one query user's information of application Y.

For checking the WSDL parsing strategy you can check at.slife.webservice.wsClient.ParseWSDL class and its methods.

4.1.2 Message Creation Phase

After parsing is finished what is needed to be done is to create SOAP request message based on chosen Service, port and operation. In web service specification each operation has zero or more argument and can have zero one or more (opposite to java) return parameters.

What is provided to operation is message; message is xml form of gathering different arguments in logically related set.

For example this is a message:

```
<wsdl:message name="getUserDatarequest">
    <wsdl:part name="info" element="esxsd:GetUserdata"/>
    <wsdl:part name=”userName” type="string"/>
</wsdl:message>
```

Above message has one simple type and one complex type that referred to GetUserdata XSD complex type.

Each operation can have input message, output message and fault message.
Each message could be just one xml simple type like string or complex type (XSD complex type)

For example following operation contains three message:

```
<wsdl:operation name="GetEndorsingBoarder">
      <wsdl:input message="es:GetEndorsingBoarderRequest"/>
      <wsdl:output message="es:GetEndorsingBoarderResponse"/>
```

```
      <wsdl:fault message="es:GetEndorsingBoarderFault"/>
</wsdl:operation>
```

What I developed in this version supports messages with simple type only, complex types are not supported in this version, for more information about this you can refer to restriction chapter of this document.

This project uses java.xml.soap for creating SOAP messages. SOAP message creation needs operation name, input namespace and input arguments with values.

As I mentioned input arguments should contain values which should some how filled by user. In GUI user can fill input argument's values via related form and in SOPA frame work user can check pipeline and fills parameter for it(in SOAP web service client application), there are some other constants must be attended in SOAP request message creation to, these constants are mentioned in SOAP creations methods of SOAPReqCreator class.

For realizing SOAP message creation you can check at.slife.webservice.wsClient.SOAPReqCreator class, method makeSOAPmessage(String targetNS,String operationName,String inputParam,String inputNS).

4.1.3 Execution Phase

In two last phases SOAP request is made base on user chosen Operation, services ... and given parameter. Now SOAP request message is ready, required parameters also should have been filled for example in SOPA as given parameter or in GUI in related input form, now we should send this SOAP message to service.

For sending request to web service in JEE 5, JAX-WS technology is used, based on JAX-WS structure first a service object needed to be created then afterward iteration trough Service object is necessary to find required port object, now port object and service object is available when this two object is ready we can create Dispatch object from them.

XML Web Services uses XML messages for communication between services and service clients. The higher level JAX-WS APIs are designed to hide the details of converting between Java method invocations and the corresponding XML messages,

but in some cases operating at the XML message level is desirable. The Dispatch interface provides support for this mode of interaction.

Dispatch works very simple if SOAP request message is ready, when request message is available simply dispatch can call service via soap request as argument like following:
SOAPMessage response = disp.invoke(soapRequest);

For more information about calling web service via JAX-WS 2.0 libraries you can refer to at.slife.webservice.wsClient.ExecuteService class, there are different methods in this class but they all use for calling service base on given SOAP message.

4.2 Design

This phase has been done in two different ways, one for abstract generic client and the other one is for SOPA framework's generic client. I will explain each of them in separate parts first, then for technical explanation on class structures and java codes details I will explain them both together.

4.2.1 Design as Abstract component

Before I start to explain check the figure 4.3, this figure shows you how generic client can use as an abstract component via its GUI independent from SOPA framework.

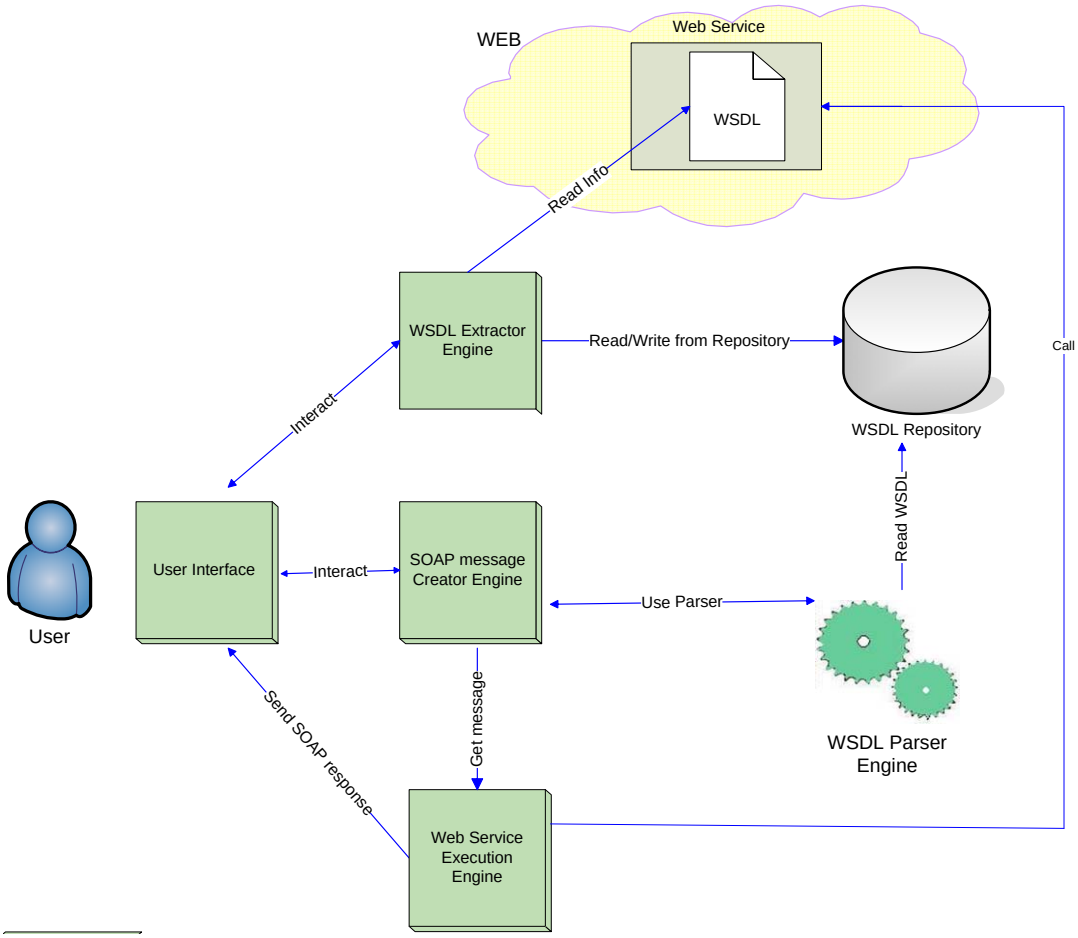


Figure 4-3 Generic Client for web service execution in abstract mode

You see in above figure user has only interaction with engines (they located in GUI) , this means via GUI user parses WSDL file and chooses appropriate information from WSDL file, GUI has lots of interaction with WSDL parser (at.slife.webservice.wsClient.parseWSDL class) after parsing and data entry is finished as third phase Web service execution engine (at.slife.webservice.wsClient.ExecuteService) receives information from GUI and call operation on service then the result of execution is shown to user.

GUI contains six forms developed based on SWT eclipse libraries:
First form is WSDL repository which save the chosen WSDL URL in text file, second one is for choosing to operates on web service's operation or create pipeline for that web service.
If you choose to operate on web service with out pipelines usage (separate from SOPA) you can choose service, port type, operation fill the value and execute service call. These forms and their usage will explained more in detail.

4.2.2 Design as SOPA plug-in

In this section the SOPA framework integration will be explored. SOPA framework is a flexible framework that provides the possibility to compose services using simpler services. It can combine Eclipse plug-ins with external web services in a simple and powerful way. The implemented component enables SOPA to interact with external services and empower the so called pipelines with Web Service communications.

Core SOPA framework is based on two main components: “Service Bus” which is used for core SOPA processes and the other one is “Pipeline” which is used for pipeline administration and related tasks. The Generic client can feed the Pipeline plug-in by creating the pipeline XML files from the service definition (WSDL file), it is also possible to invoke “Service Bus” instead of pipeline but all arguments like service name, port name, etc should be given by user in this case. In Pipeline case all arguments will be read from pipeline that created before. User just needs to fill variables for input parameters and call related pipeline.
For better understanding consider the following figure and examples:

“Service Bus” call example:

```
Object[] params={"
http://www.xmethods.net/sd/CurrencyExchangeService.wsdl"
,"CurrencyExchangeService"
,"CurrencyExchangePort"
,"getRate"
,"<country2
xsi:type=\"xs:string\">Austria</country2><country1>xsi:type=\"xs:str
ing\">UK</country1>"};

Call client = new Call("at.slife.webservice");
result = client.invoke("wsCall",params);
```

Above example have been written in an eclipse plug-in we named it Client4WS. Client4WS is a client (not web service client it is an eclipse client) which makes a new call to at.slife.webservice according to above codes.
Note that wsCall is eclipse operation and it must be defined in at.slife.xmas.services as a service operation, by client plug-in developer.

```
Call client = new Call("at.slife.webservice");
```

With instantiated parameter like this:

```
Object[] params={"
http://www.xmethods.net/sd/CurrencyExchangeService.wsdl"
,"CurrencyExchangeService"
,"CurrencyExchangePort"
,"getRate"
,"<country2
xsi:type=\"xs:string\">Austria</country2><country1>xsi:type=\"xs:str
ing\">UK</country1>"};
```

And at the end appropriate operation(wsCall) on XMAS plug-in services is invoked. Same as following:

```
result = client.invoke("wsCall",params);
```

PIPELINE Call Example:

```
Object[] params =
{"CurrencyExchangeService_CurrencyExchangePort_getRate:getRate"
,"<country2>Germany</country2><country1>Austria</country1>"};
Call client = new Call("at.slife.pipeline");
result = client.invoke("invokePipe",params);
```

Above example shows the required code for calling one web service via pipeline.
Again like “Service Bus” client call, these codes should be located in an eclipse client use for calling eclipse service, which has been derived from at.slife.xmas.services but in this case Pipeline service is called not “Service Bus” service.

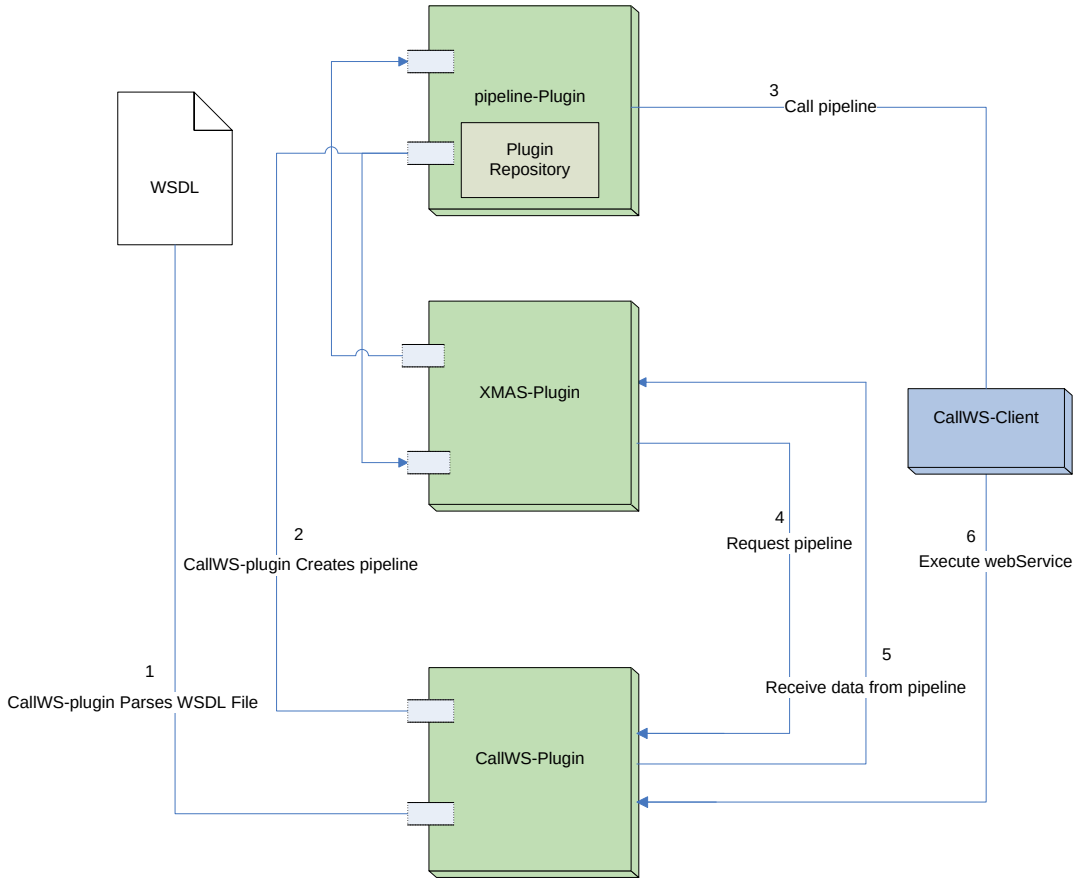


Figure 4-4 Generic client in SOPA framework

Figure 4.4 depicts the usage of generic client in SOPA framework, assume the green plug-ins as core SOPA plug-in CallWS plug-in is the one which is designed in this project to handle generic client, as you can see there, client (blue one) named CallWS – Client provided for accessing the CallWS-plugin.

Core SOPA framework is XMAS and PIPELINE, client can used via PIPELINE plug-in, and also other forms of plug-in can be plugged into SOPA framework.

CallWS is a generic client which has been designed as eclipse plug-in derived from XMAS services and provides web service access to SOPA framework.

First step is to call WSDL file, after CallWS accesses content of WSDL file and parses it via at.slife.webservice.wsClient.ParseWSDL it can creates pipeline for that WSDL file via at.slife.webservice.pipeline.Createpipeline class in second step.

After pipeline have been created (base on WSDL file) for web service, now it is possible to call it and execute operations on that web service(via pipeline).

Third step could be the first one if pipeline is available before, in simple term when we don't need pipeline creation process can jump to third step.
"callWS client" query PIPELINE plug-in for service execution (XMAS has some internal interaction with pipeline which is not in the scope of this document) XMAS plug-in requests pipeline for appropriate pipeline XML block and if there is any available pipeline plug-in provides it to callWS, then callWS can execute specified operation on specified service via user given parameter and information coming from pipeline.

The result is sent back as String but it is SOAP response message, like following:

```
String result = client.invoke("invokePipe",params);
```

String result in above in nothing than a SOAP response message in String format.

4.3 Implementation

In this part, java classes and all their associated methods in details will be explained.
This part is about technical specification and contains nothing about requirement or analysis.
Project class diagram and sequence diagram is explained after the package description.
First java packages are explained. Then we will go in details of their associated classes and for each class member variables (if necessary) and methods will be explained.

As a convention all package names containing at.slife.webservice as prefix are simplified to the last part of package name. For instance the "clientGUI" package name is at.slife.webservice.clientGUI package.

4.3.1 at.slife.webservice.clientGUI Package

This package has been used in abstract mode independent of SOPA framework. The main goal of this package is to provide users with the graphical user interface which have been used for two purposes: first to create pipelines that are very important because there is only one way to create pipeline, and the second is to let user uses GUI without SOPA framework and go trough web services for execution of operations.

ReadWSDL_1

This class is first GUI class which appears for user. It contains a list of Web services that came from web service repository (web service repository is a text file for storing WSDL URLs) this class maps to first GUI form and has following methods:

public static void main(String[] args) throws Exception
This is the main entry point to GUI application.

private void createSShell(Display disp) throws Exception
This method initializes sShell, this GUI is written by SWT libraries and sShell means an Shell member variable of SWT libraries.
This method cares about clicking add, delete button that are responsible for adding and removing WSDL to/from repository.
And a Next button which sends user to another form.

SelectBrowseOrPipleLine_2

This class appears as second GUI frame, I mentioned before that you can do two things with GUI, one is to parse WSDL and call its operation the other one is to create pipeline for each operation in chosen WSDL file.
This class has following methods:

public void initGUI() throws Exception
This method used to initialize the SWT shell and create GUI

public static void main(String[] args) throws Exception{
This method open initialized shell and display it to user.

private void createSShell(Display disp)
This method use for creating SWT shell and shows the GUI to user, it also contains some java code for handling Next and Back button. The other important thing that this method will handle is saving status of user selection (if user wants to create pipeline for this wsdl or user wants to execute operation on web service with GUI)

SelectService_3

This class have been assigned to third GUI frame, after parser parses the chosen WSDL file in this form(frame) user gets the list of available services, then he/she can choose the service and click on next button to go to next GUI form

public void initGUI() throws Exception
Same as other GUI classes.This method is using for initializing the SWT shell and creating GUI.

private void createSShell(Display disp) throws Exception
This method have been used to create SWT shell and shows GUI to user, it also contains some java code for handling “Next” and “Back” button. It saves the selected service from user selection, and then next form will show the associated port base on chosen service in this form.

SelectPortType_4

This class is 4th GUI form that will been shown to user, base on chosen service in last form(SelectService_3) user can see the port names which are related to that service in this form and select which port he/she wants to operate with.

public void initGUI() throws Exception
Same as other GUI class, this method used to initialize the SWT shell and create GUI

private void createSShell(Display disp) throws Exception
This method use to creating SWT shell and showing the GUI to user, it also contains some java code for handling Next and Back button. It saves the port name from user selection, and (next form after this one) shows the associated operation base on chosen port in this form.

SelectOperation_5

public void initGUI() throws Exception
Same as other GUI class, this method have been used to initializing the SWT shell and creating GUI

private void createSShell(Display disp) throws Exception
This method have been used to creates SWT shell and shows the GUI to user it also contains some java code for handling Next and Back button. It saves the operation name from user selection, and then next form shows the associated SOAP message which is created base on chosen operation in this form.

MessageInput_6

This class will generate a SOAP request message based on selected information in last forms, It contains "-----" for input arguments of selected operation and they should filled by user.

There are two SWT tab in this form one is to check the created SOAP request and if user like he/she can manipulate it, the other tab is some GUI form for user to enter input argument's values easily.

public void initGUI() throws Exception

Same as other GUI class, This method is using for initializing the SWT shell and creating GUI.

public void fillInputsfromSOAP(String input) throws Exception

This method gets the input part from SOAP message and updates SOAP message with user given input values on the input Data tab. It uses DOM parser for parsing given input xml and updates values in Data tab.

public String fillSOAPwithInput(String oldInputs)throws Exception

This method acts opposite to fillInputfromSOAP. In this method data from input Data tab are gathered and filled in appropriate message (in SOAP message tab), this means you can enter input arguments in both tabs SOAP message tab or Input data tab.

public String readInput4mGUI(String txtmsg) throws Exception

When input argument is read from SOAP message, they have some extra attribute like type. For sending SOAP request this extra information can remove. This method is responsible for removing extra attributes from input XML elements and returns another xml input element with out extra attribute.

private void createSShell(Display disp) throws Exception

This method have been used for SWT shell creation it also has some calculation to do what when tab changes, This calculation done in tabfolder.addSelectionListener inner methods, above methods that I explained for this class, are all used in this method.

It also has some business logic to call related class and method to execute service (with created SOAP request message) and call web service then the SOAP response message is shown SOAP message tab in text.

4.3.2 at.slife.webservice.general package

This package used for storing different kind of information about the project, for examples forms have one image on the first part of them, or state of the chosen things in GUI should stored some where, WSDL repository is container of WSDL file, its location and … needs to be stored somewhere.

In following associated classes will explained in detail

Setting

This class holds place of the created pipeline in constant member variable named PIPELINE_LOCATION.

Place of the WSDL repository file which is responsible for storing WSDL file is defined in REPOSITORY_LOCATION member variable.

Repository file name is defined in REPOSITORYFILE_NAME member variable.

Header image of each GUI form is defined in GUI_IMAGEx member variables, x range is from 1 to 6 and related to the number after "_" in form name.

StateHolder

This class contains member variables which are responsible for storing user selection when user is working with GUI.
There are predefined private and static String as member variable in this class.
Application forms fills these private Static String with their Setter methods and reads them with their associated getter methods.
Each one of this member variable has a role in creating SOAP message for requesting a service. Here is the list of these variables:
targetNameSpace, inputNameSpace, chooseWSDL, choosePortType, chooseServic, chooseOperation, chooseMessage, chFunction

chFunction here is exception. It has been used for second (SelectBrowseOrPipeLine_2) form(the one we choose to create pipeline or browse and operate on web service).

WebServiceRepository

As mentioned before WSDL files URL stored in repository, and for each execution of Application in Abstract mode user can check the list of WSDL files, adds new WSDL to repository or removes them from repository, Repository is nothing than a simple text file.
For writing and reading from repository this class is created. In simple term this class is used for repository file administration.
It contains following methods:

public void writeInWSDLRep(String wsdl) throws Exception
This method will be used to add new WSDL file in repository file.

public void del4mWSDLRep(String wsdl) throws Exception
This method will be used to remove the selected WSDL file(in GUI) from repository.

public String[] readAllWSDLs() throws Exception
This method reads the entire WSDL files from repository file and provide them to GUI.

4.3.3 at.slife.webservice.pipeline package

This package as name shown will be used for pipeline administration. When we choose a WSDL in second GUI form, it asks us if we need to create a pipeline for

that WSDL or not? if we accept to create a pipeline for each operation in that WSDL a new pipeline XML file will be created.
This package contains two classes one is for development test purposes and the other one is for creating pipeline. I explain second one in detail.

After pipeline is created it stored in xml file that has special pattern like as following: xxx_yyy_zzz.xml
xxx stands for service name, yyy stands for port name and finally zzz stands for operation name.

PipelineAdmin

This class is used for development test purposes and doesn't have any effect on project in operational phase so I don't explain methods.

CreatePipeline

This is the main class that used for pipeline creation and administration.

public void executePipelineCreation(String wsdl) throws Exception
This method gets WSDL URL as String and for each methods in WSDL creates one pipeline it doesn't contain business process, business processes happens via calling createPipeline method.

public ArrayList readAllInfo(String wsdl) throws Exception
This method reads WSDL file and puts all pipeline creation required data, in Arraylist then it returns that array list. Array list contains important data like port name, service name and …

public void createPipeline(ArrayList allInfo,String wsdl) throws Exception
This method gets the array list that comes from readAllInfo method and WSDL URL; via processing the content of these data pipeline file is created.

Please consider for each web service operation one pipeline file is created and this pipeline contains information about that specific operation.

4.3.4 at.slife.webservice.wsClient package

This package is core package for generic client business process, general package is storage for project setting, clientGUI is mapping to GUI of application and this package maps to business process of application.
It contains one class for parsing WSDL which used WSDL4J libraries and it is responsible for getting information out of WSDL file. The other important class in this package is the one that executes web service within given SOAP request message.
Here are class descriptions for this package:

ExecuteService

This class is used for Web Service execution (Here connection to web service is established and service is executed). For operating on Web service it uses JAX-WS libraries.
This class contains following methods:

public void callAsynch(String wsdlPlace,String tNS,String serviceNam ,
String portNam, String operNam,
String inputParams,String inNS) throws Exception

This method will be used for asynchronous web service execution, it is not supported in current version of this project.

public String callSynch(String wsdlPlace,String tNS,String serviceNam ,
String portNam, String operNam,

String IputParams,String inNS) throws Exception

This method gets all necessary inputs as arguments, makes SOAP message, executes the service, gets SOAP result and changes the format to String then returns it.

public String callSynch(String SOAPReq) throws Exception

This method gets the whole SOAP request message as String, makes SOAP message from SOAP message with argument(in String format), executes the Service, gets the SOAP response and returns the SOAP Response in String format.

public void execute() throws Exception

This method will be use for executing service from GUI, GUI has all the required information for web service, then this service reads them from **general.StateHolder** class and executes the service via making call to callSynch method.

public static void main(String[] args) throws Exception

This is just for internal test purposes.

ParseWSDL

This class will be used for parsing WSDL file, it uses StAX and WSDL4J libraries for parsing the WSDL file. First it reads WSDL and then parse its component.

All WSDL4J libraries are written with DOM and other nonWSDL4J dependent methods is written with StAX.

public ParseWSDL(String wsdlLocation)

This method will be used to initialize the parser. It has some WSDL4J internal usage and there is no business logic related to this method.

public String readTargetNamespace(String wsdlLocation) throws Exception

This method gets WSDL URL as input, reads it, parses it and returns the associated name Space for given WSDL file.

public String readInputNameSpace(String wsdlLocation,String operationName) throws Exception

This method gets WSDL URL and operation name as input arguments, then finds associated input name space for given operation and returns it.

private String[] readPortNAMESFromServiceName(String wsdlLoc, String serviceName) throws Exception

This method gets WSDL URL and service name as input arguments and finds associated ports which assigned to given Service name and return them in array of Strings.

public String[] readAllServices(tring wsdlLocation) throws Exception

This method gets WSDL URL as input argument and finds associated Service names which assigned to given WSDL URL and returns them in array of Strings.

public String[] readAllports(String serviceName, String wsdlLOC) throws Exception

This method gets Service name, WSDL URL as input arguments then finds associated port names which assigned to given WSDL URL and service name and at the end returns them in array of Strings.

public String[] readOperations(String currServName, String currPortName, String wsdlLoc) throws Exception

This method gets Service name, port name,WSDL URL as input arguments and finds associated operation which assigned to given WSDL URL, port name and service name and then returns all founded operation in array of Strings.

public String readInputParam(String currServName,String currPortName, String wsdlLoc,String operNam) throws Exception

This method gets all necessary information for given operation and returns a string which contains input parameters like following as example:

<country1 xsi:type=”xs:String”>- - - - - </<country1>

Please consider instead of value this method inserts “-----“and afterward user can manipulate it.

public ArrayList readinputs4Pipeline(String currServName, String currPortName, String wsdlLoc, String operNam) throws Exception

This method reads inputs for given operation but it will be used for creating pipelines.

SOAPReqCreator

This class do all the SOAP message creation things

public String makeSOAPMessage(String targetNS,String operationName,String inputParam,String inputNS) throws Exception
This method gets all the required information for creating SOAP Request message, creates a SOAP request in String format and returns it.

public SOAPMessage makeSOAPmsgFromString(String msg)
This method gets the SOAP Request message in String format and makes a java SOAP message then returns the result; input of this method is return value of makeSOAPMessage.

public SOAPMessage makeSOAPmsgFromString_test()
This method is for development test purposes and doesn't need any explanations.

public static void main(String args[])
This method is for development test purposes and doesn't need any explanations.

XMLInstmaker

This class used for creating XML Instance for complex schema and doesn't supported in this version

4.3.5 at.slife.webservice Package

Connect2WS

This class located in root of at.slife.webservice and contains one methods for reading parameter from client and calling web service in SOPA framework, in another term this class is gateway to SOPA framework.

It has one method **public String wsCall(String wsdlLoc, String serviceName, String portName, String operName, String inputs)**

And this method does the execution of web service via making call to other methods in other packages.

Client code calls this method via Pipeline Plug-in because this class and its method registered as XMAS service and operation.

4.4 UML Diagrams

For better understanding the execution process and structure of this project I try to explain the class diagram and sequence diagram of this project, first I explain class diagram and then sequence diagram.

4.4.1 Class diagram

Figure 4.5 depicts UML class diagram of this project.

Classes that are using for GUI are shown in alliaceous and the other classes which are presented in blue are used for business process of generic client. StateHolder is also used for Client when GUI doesn't require, and web services have been executed (via making call to pipeline) we don't need to maintain any state because all information are coming from pipeline.

You can also mention that all alliaceous classes have association relation (has a) with ParseWSDL, because parseWSDL class will be used in both cases (SOPA or GUI) to parse WSDL file.

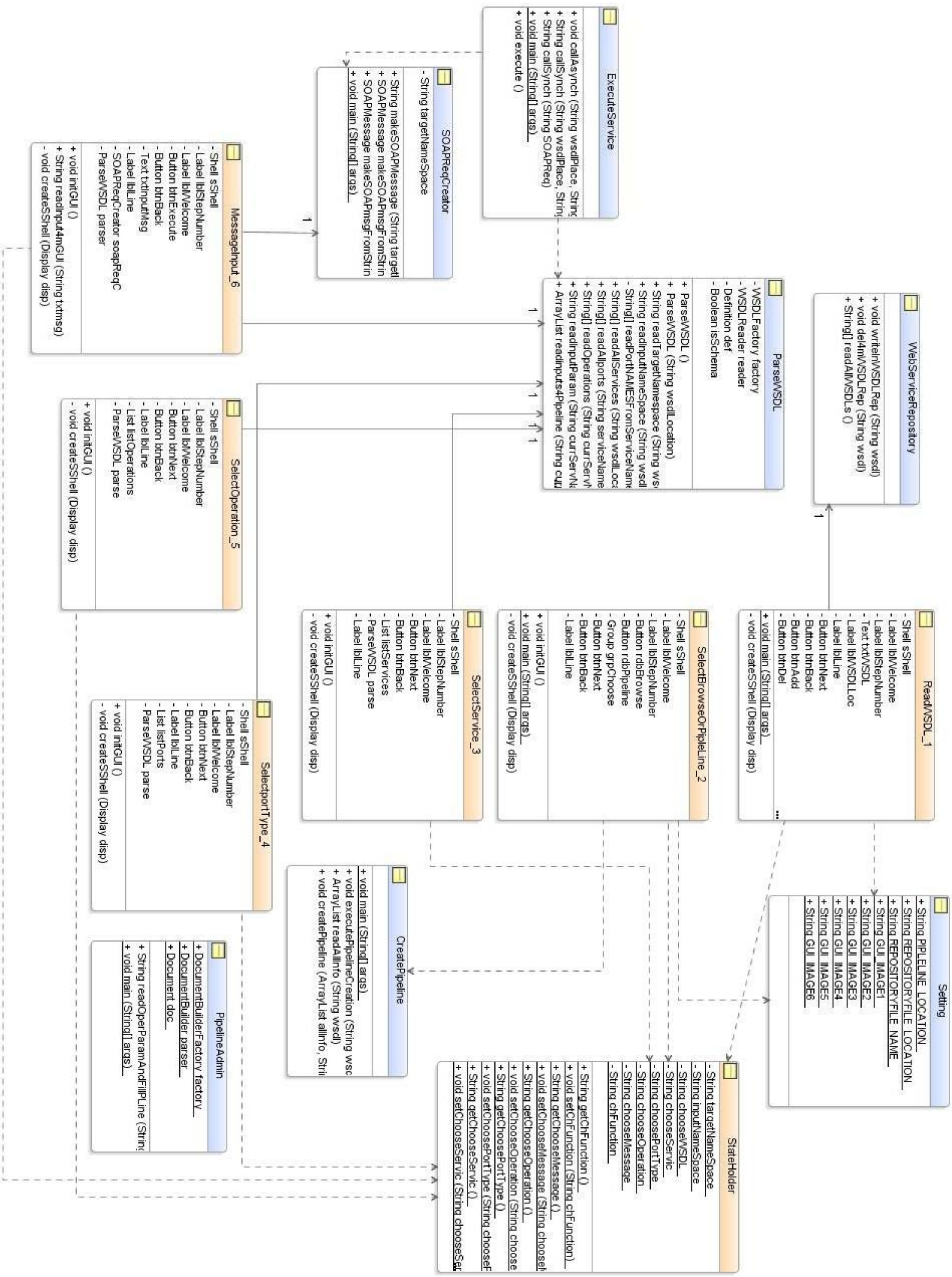
4.4.2 Sequence diagram

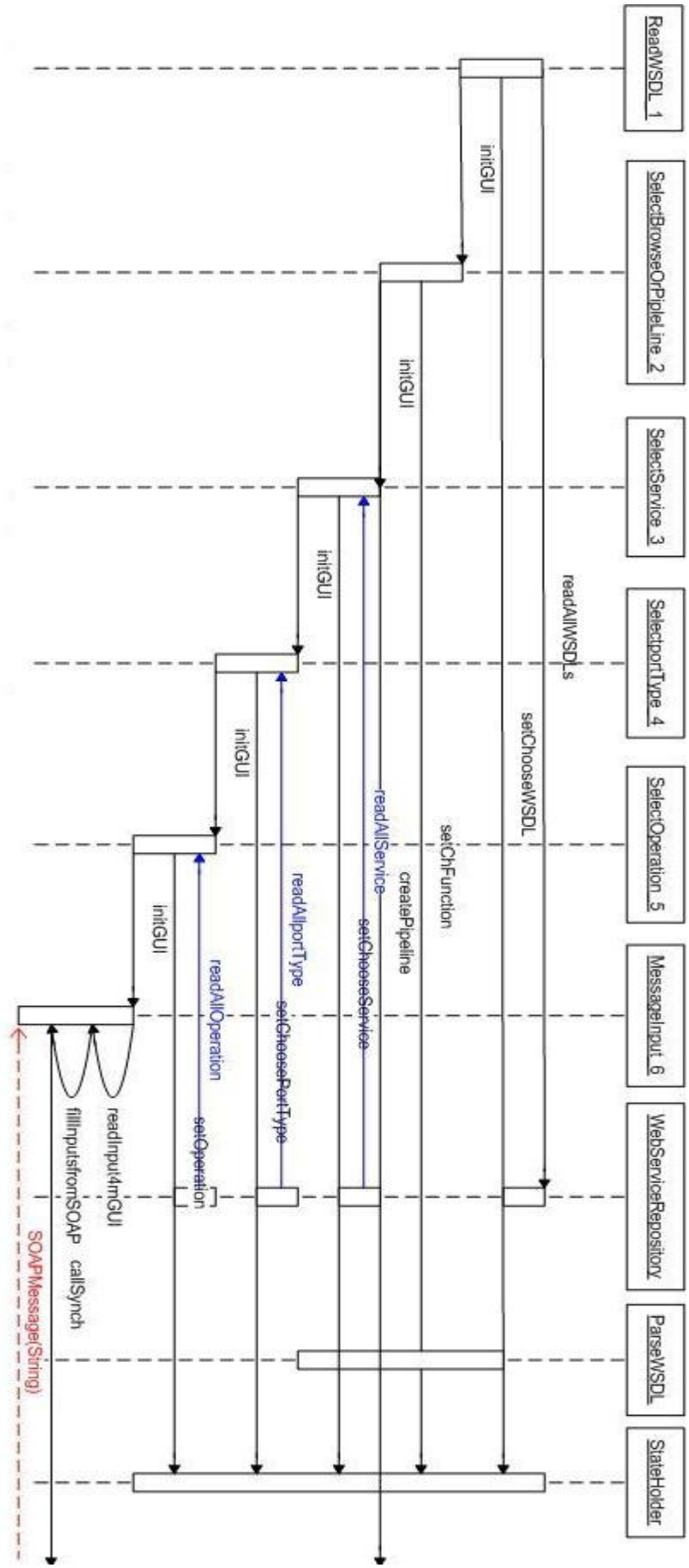
Figure 4.6 depicts UML sequence diagram of this project. In sequence diagram you can see all GUI classes have interaction with StateHolder classes, because it saves the user selection on each form, you can also see that some of them have interaction with ParseWSDL.

With checking Sequence diagram in figure 4.6 you can realize the message flow in generic client within GUI.

When we work on SOPA framework level there is no usage for GUI classes, except creating Pipeline.
In SOPA framework following classes are used:
Setting, createPipeline, ReadWSDL_1, ExecuteService, ParseWSDL, SOAPReqCreator

Figure 4-5 Generic Client class diagram




ReadWSDL_1
SelectBrowseOrPipleLine_2
SelectService_3
SelectportType_4
SelectOperation_5
MessageInput_6
WebServiceRepository
ParseWSDL
StateHolder
initGUI
initGUI
initGUI
initGUI
initGUI
readAllWSDLs
setChooseWSDL
setChFunction
createPipeline
readAllService
setChooseService
readAllportType
setChoosePortType
readAllOperation
setOperation
readInput4mGUI
fillInputsfromSOAP
callSynch
SOAPMessage(String)

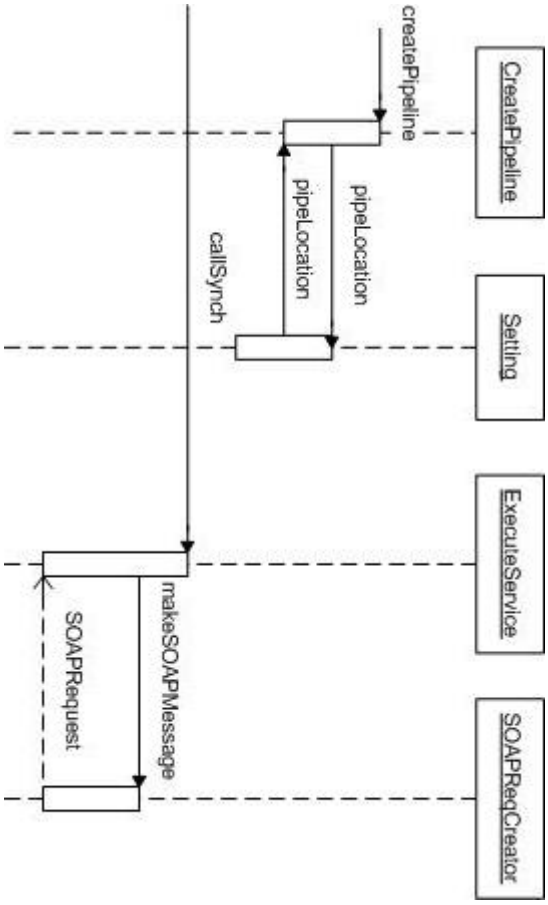


Figure 4-6 generic client Sequence Diagram

4.5 User Manual

In this part I try to explain how user can use generic client in SOPA framework or the abstract mode.

Before starting this tutorial please note that if user wants to use Generic client or use SOPA framework in both cases he/she needs to create pipeline and run GUI so I start with GUI explanation.

Imagine user runs the first GUI file ReadWSDL_1 then he/she gets the following figure (figure 4.7)

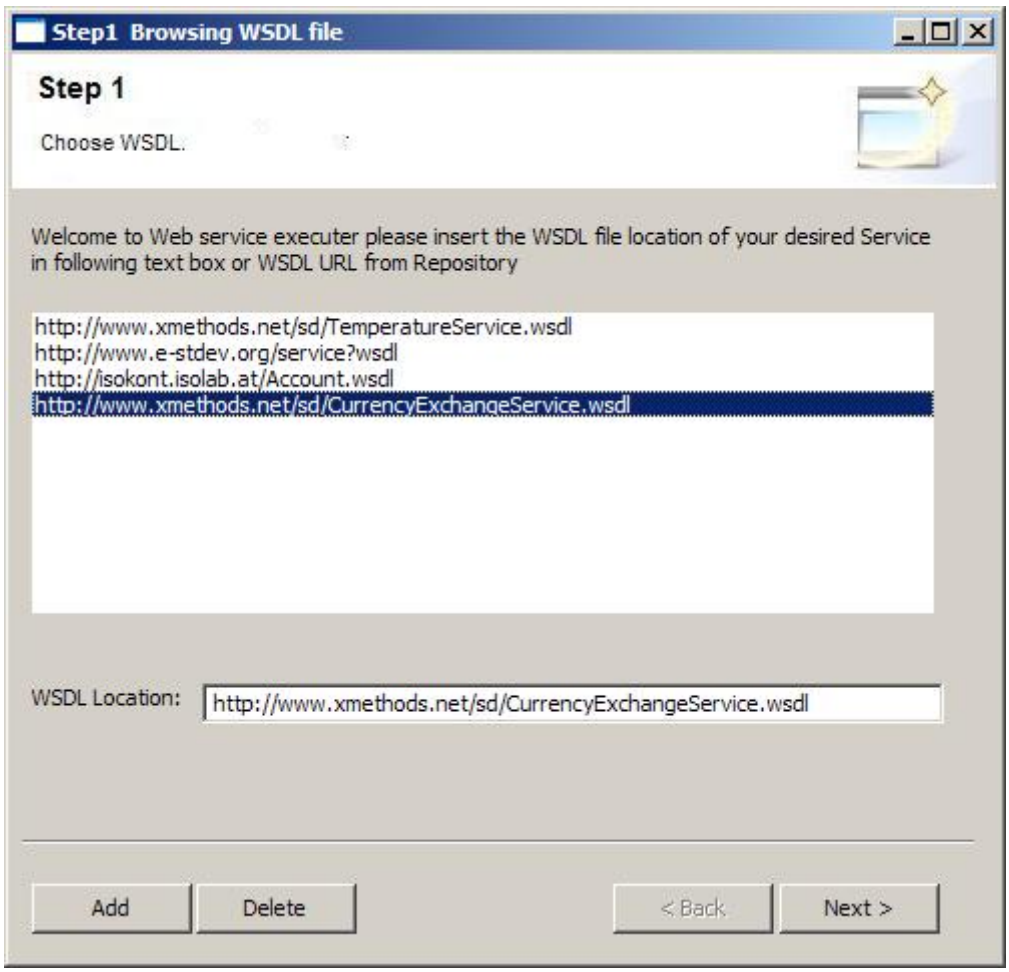


Figure 4-7 Read WSDL

This form has two functionalities, one is to add and removing web services to/from web service repository and the other one is for choosing which web service we want operate with.

If we need to add new WSDL file to web service repository we should write WSDL URL in WSDL Location text box then click on add button, if we want to remove WSDL file from web service repository we should choose the remove candidate from WSDL lists, it appears in WSDL Location then after we clicked delete Button that WSDL file Location is removed from web service repository.

After clicking next button second form will be appeared (figure 4.8)

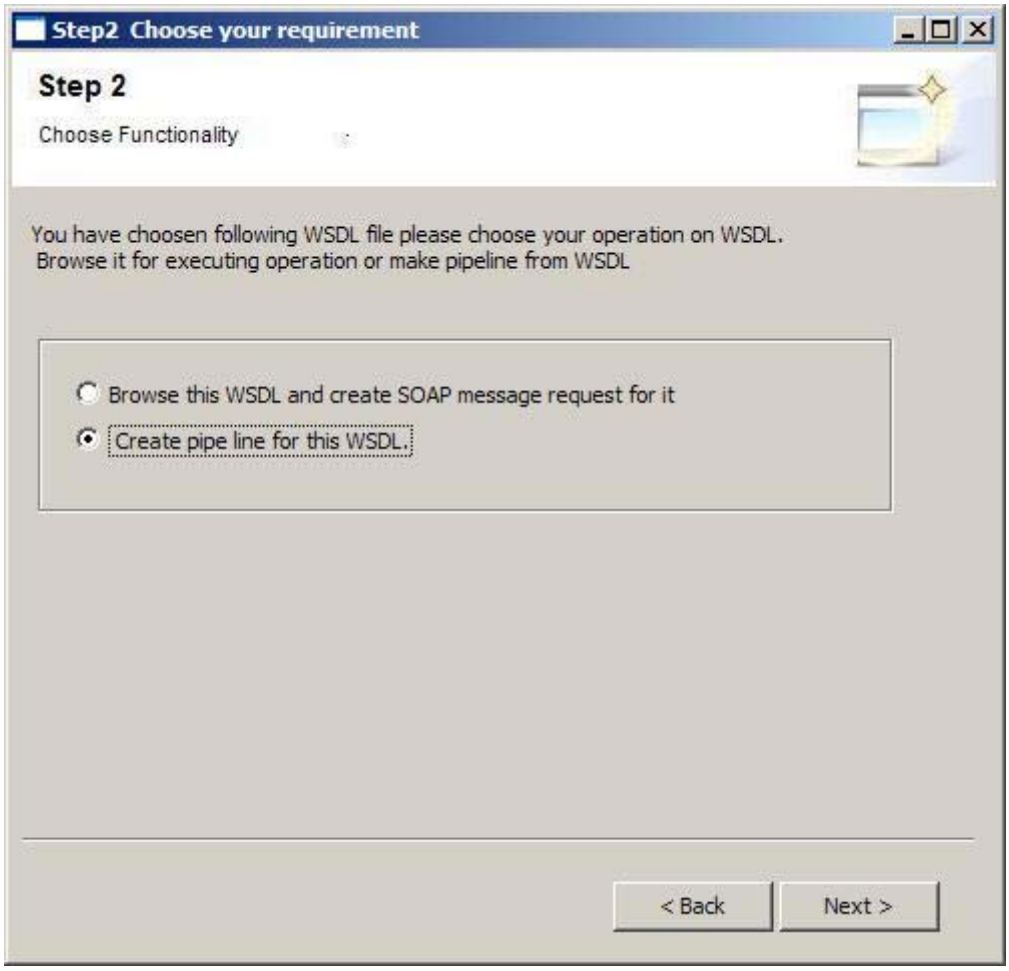


Figure 4-8 Choose functionality

In this form user chooses what he/she wants to do with the selected WSDL file, does he wants to create Pipeline for it or he/she wants to browse its services and execute its operation on GUI level.

If user selects “Create pipeline for this WSDL” when he clicks next he gets message box that informs him pipeline for your WSDL file is created and shows him the path of created pipeline in Operating system.

If he chooses “Browse this WSDL and creates SOAP Message request for it” he will go to next form shown in figure 4.9

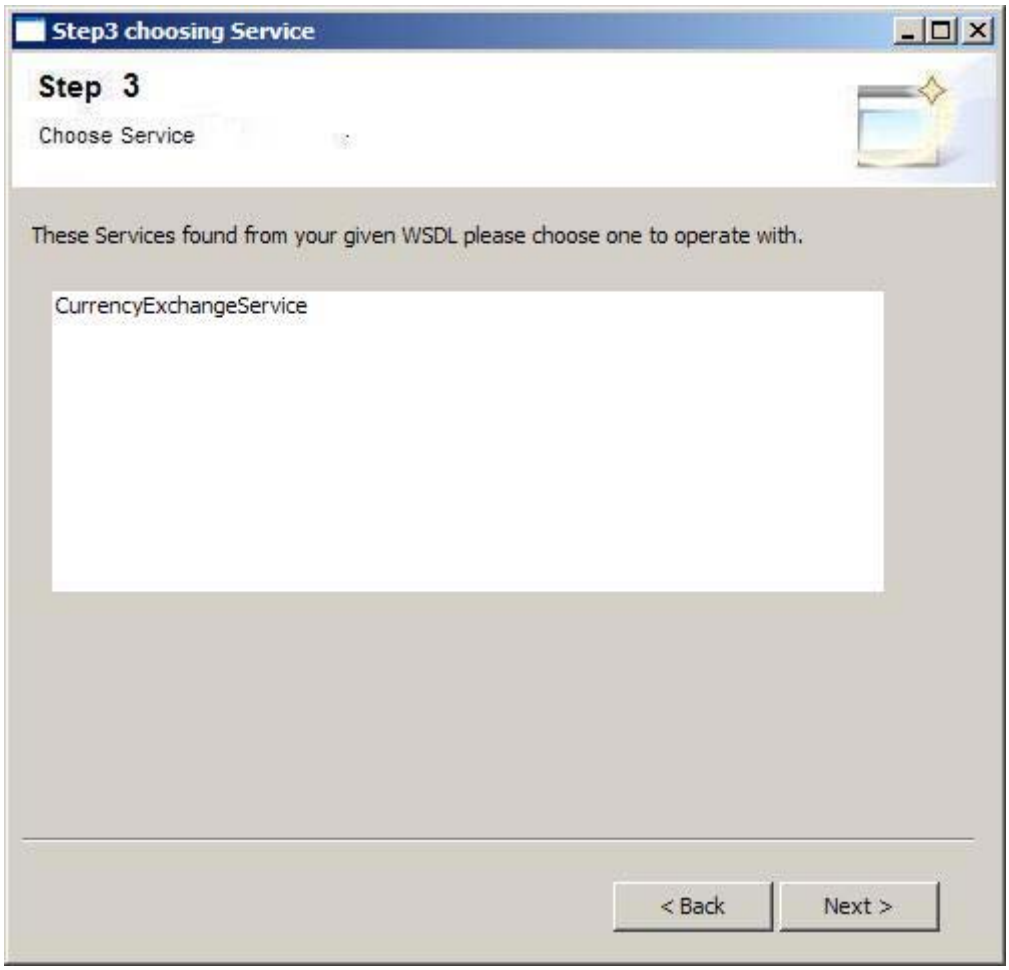


Figure 4-9 choose Service

Figure 4.9 shows you the list of services of a chosen WSDL file, for example this one has one services and it is CurrencyExchangeService here.
After user selecting the service he can click the Next button and see the next form (figure 4.10), if he doesn’t select any service and clicks next he will get an alert to select one service.

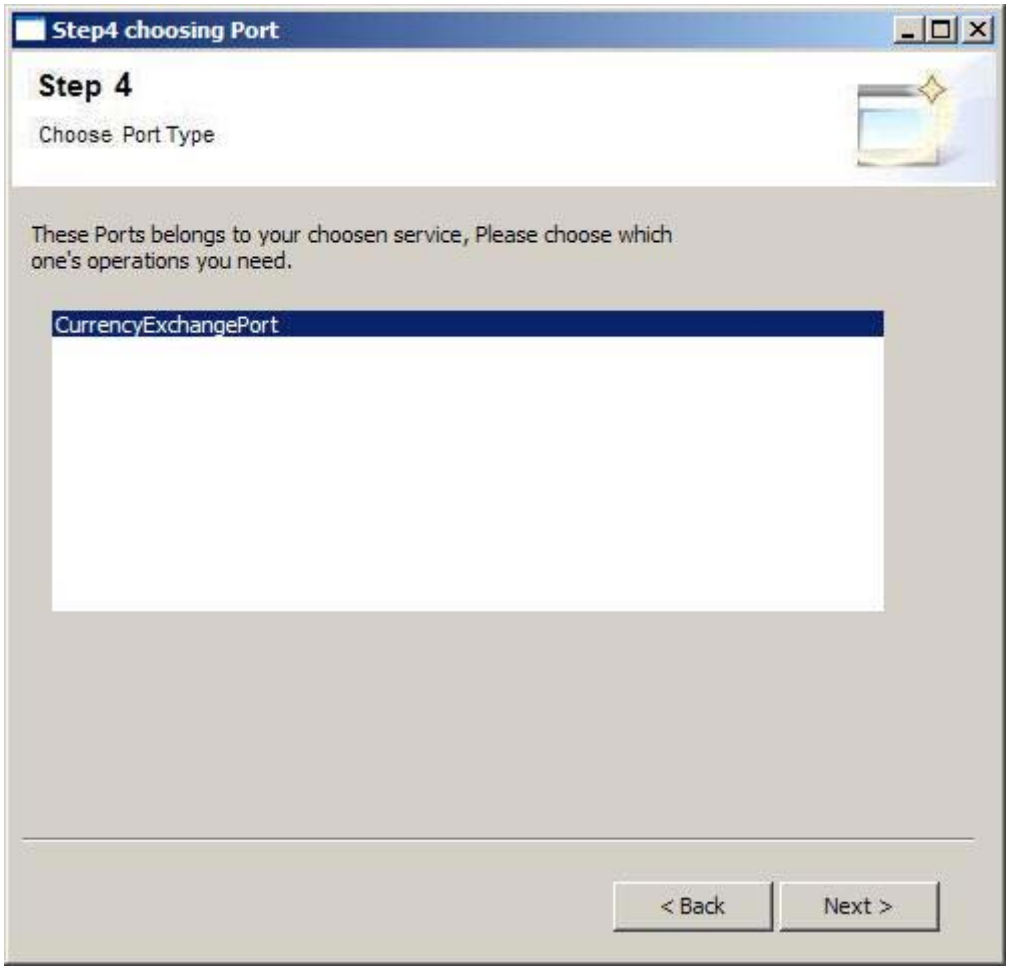


Figure 4-10 choose Port

Above form shows the ports that coming from chosen service in last form. Now user should choose the port he/she want to operate with, and click Next button.

After user clicks next button another form appears and this form shows the operations which are available for chosen port in last form.

Figure 4-11 choose web service operation

Figure 4.11 shows the form that has been used for selecting operations.

After Operation is selected by user, user gets the last form that shows SOAP request message created for the chosen operation, note that places for input variables are empty and shown with "-----"in figure 4.12. User can go to input Data tab figure 4.13 and enters values for input variables. It is possible to enter data for input variables for both tabs (SOAP message tab and Data tab).

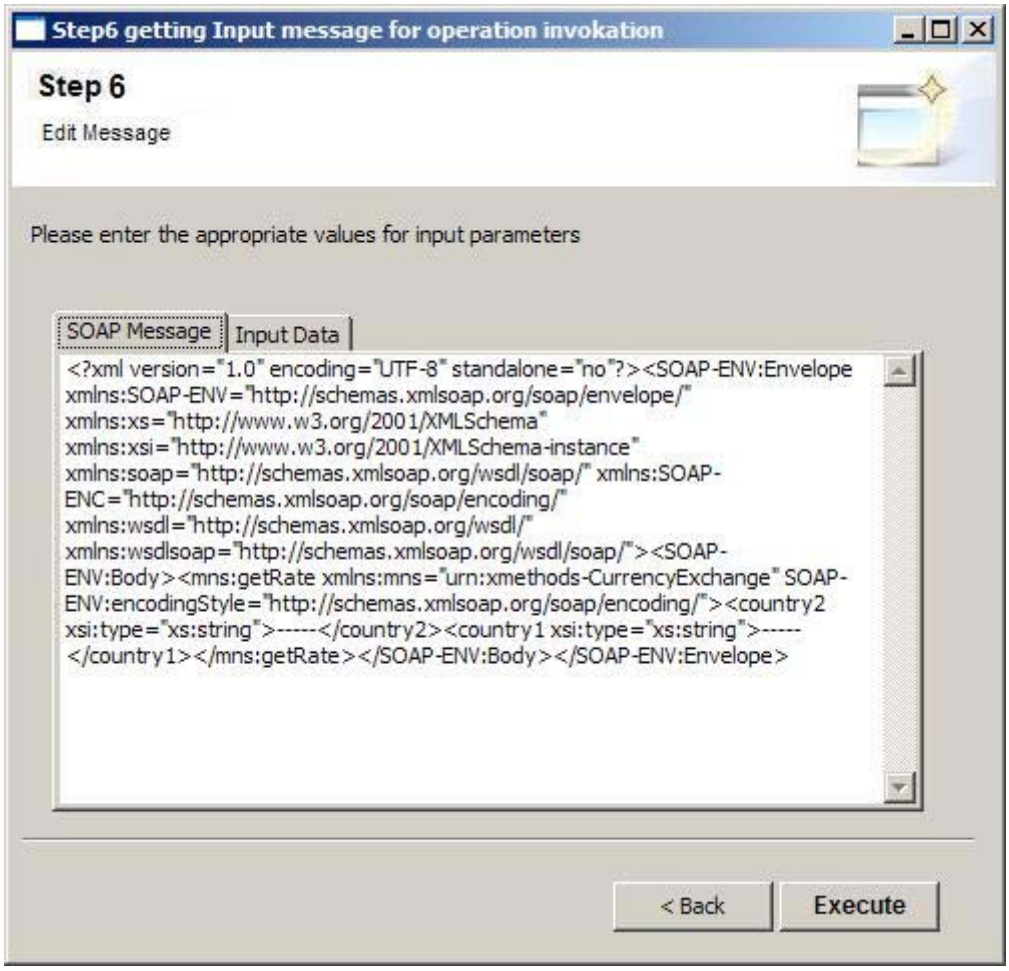


Figure 4-12 Message Input, SOAP message tab

After entering input variable data by user is finished, user can click Execute button and checks the SOAP response result in SOAP Message tab.

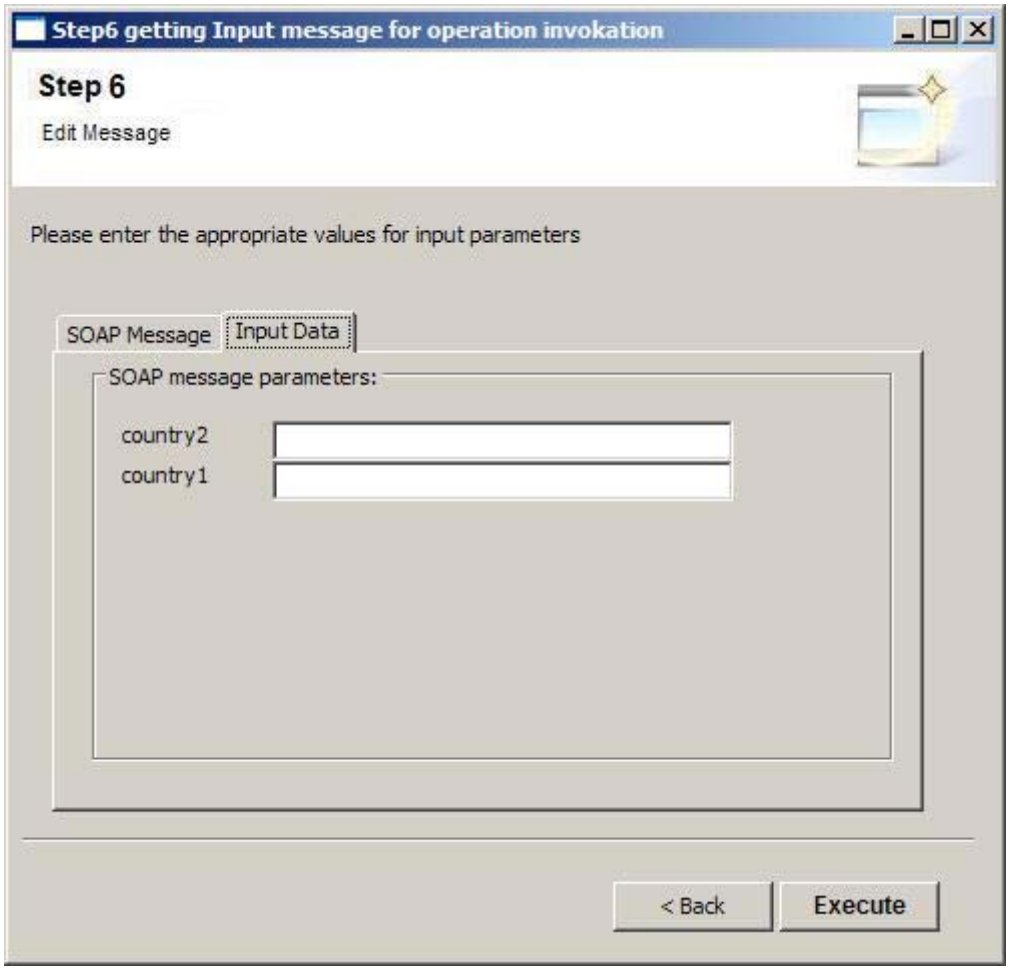


Figure 4-13 Message Input, Input data tab

Chapter 5

Integration with SOPA

Information of this chapter coming from following references:

- Oracle BPEL Tutorial http://otn.oracle.com
- Semantic Enrichment of Search Result: the Coupling of Semantic Strore and Google Services. by Khabib Mustafa, Amin Andjomshoaa, A Min Tjoa Institue for Software Technology and Interactive Systems TU Viennsa, Austria
- Service Oriented Architecture Cocepts, technology and Design by Thomas Erl
- Service Oriented Architecture a Field guide to integrating XML and web services by Thomas Erl

5.1 General Information

For understanding Business process (BPEL or SOPA) following terms should have been explained in detail: Coordination, orchestration, business Activity, Atomic transaction.
First these terms will be explained then as second step BPEL and SOPA will be explained, at the end a comparison between each of them will be provided.

5.1.1 Coordination

Every activity introduces a level of context into an application runtime environment. Something that is happening or executing has meaning during its lifetime, and the description of its meaning can be classified as context information.
The complexity of an activity can relate to a number of factors, including:

- The amount of services that participate in the activity
- The duration of the activity
- The frequency with which the functionality of the activity changes
- Whether or not multiple instances of the activity can concurrently exist

A coordinator based context management framework, as provided by WS-Coordination and its supporting coordination types, introduces a layer of composition control to SOA. It standardizes the management and interchange of context information within a variety of key business protocols.
Coordination also reduces the need for service to retain state. Statelessness is a key service orientation principle applied to services for use within SOA, coordination forces statelessness by assuming responsibility for the management of context information.

5.1.2 Transaction

Atomic transaction

Atomic transaction implements the familiar commit and rollback features to enable cross service transaction support. Most salient adjective of Atomic Transaction is that it can't be broken in smaller logical part.

ACID transaction

The protocols provided by the WS-AutomicTransaction specification enable cross service transaction functionality comparable to the ACID compliant transaction features found in most distributed application platforms.

The term ACID is acronym representing following characteristics:
Atomic: Either all of the changes within the scope of the transaction succeed or none of them succeed. This characteristic introduces the need for the rollback feature that is responsible for restoring any changes completed as part of failed transaction to their original state.
Consistent: None of the data changes made as a result of the transaction can violate the validity of any associated data models. Any violations result in a rollback of the transaction.
Isolated: If multiple transactions occur concurrently, they may not interfere with each other. Each transaction must be guaranteed an isolated execution environment.
Durable: Upon the completion of a successful transaction, changes made as result of the transaction can survive subsequent failures.

Much of the transactional functionality implemented in service oriented solutions is done so among the components that executes an activity on behalf of a single service. However, as more services emerge within an organization and as service compositions become more commonplace, the need to move transaction boundaries into cross-service interaction scenarios increases. Being able to guarantee an outcome of an activity is a key part of enterprise-level computing, and atomic transactions therefore play an important role in ensuring quality of service.

Not only do atomic transactional capabilities lead to a robust execution environment for SOA activities, they also make interoperability when used into integrated environments.
This allows the scope of an activity to span different solutions built with different vendor platforms, while still being assured a guaranteed all-or-nothing outcome.
If WS-Atomic Transaction used by different application this of course broadens the option of two phase commit protocol.

5.1.3 Business Activity

Business activities govern long-running complex service activities. Lots of effort need to be done to cause business activity has enough ability to complete. During this period, the activity can perform numerous tasks that involve many participants.

What distinguish business activity from a regular complex activity is that its participants are required to follow specific rules defined by protocols. Business activities primarily differ from the also protocol-based atomic transaction in how they deal with exceptions and in the nature of the constraints introduced by the protocol rules.

For instance, business activity protocols don't offer rollback capabilities. Given the potential for business activities to be long-running, it would not be realistic to expect ACID-type transaction functionality. Instead, business activities provide an optional compensation process that can be invoked when exception conditions are encountered.

Business Activity is obvious reason of SOA's compos-able nature. Service autonomy and statelessness are preserved by permitting services to participate within an activity for only the duration they are absolutely required to. This also allows for the design of highly adaptive business activities wherein the participants can augment activity or process logic to accommodate changes in the business tasks being automated. Through the use of the compensation process, business activities increase SOA's quality of services by providing built in fault handling logic.

Please consider that the use of business activity doesn't exclude the use of atomic transactions. In fact, it is likely that a long-running business activity will encompass the execution of several atomic transactions during its lifetime.

5.1.4 Orchestration

The role of orchestration broadens in service oriented environments. Through the use of extensions that allow for business process logic to be expressed via services, orchestration can represent and express business logic in a standardized, service base context.
When building service oriented solutions, this provides an attractive means of housing and controlling the logic representing the process being automated.

Orchestration further leverages the interoperability in service designs by providing potential integration endpoints into processes. A key aspect to how orchestration is positioned within SOA is the fact that orchestrations themselves exist as services. Therefore building base on orchestration logic standardizes process representation across organization, while addressing the goal of enterprise orientation.

A primary industry specification that standardizes orchestration is the Web services Business Process Execution languages (WS-BPEL or formerly known BPEL4WS), which I explain in more detail and compare it with SOPA. SOPA is also another form of business orchestration.

5.1.5 Choreography

In ideal world all organizations would agree on how internal processes should be structured, so that should they ever have to interoperate, they would already have their automation solutions in perfect alignment.

But this not reality just imagination, the requirement of organizations to interoperate via services is becoming increasingly real and increasingly complex. This is especially true when interoperation requirements extend into the realm of

collaboration, where multiple services from different organizations need to work together to achieve a common goal.

The Web services choreography description language (WS-CDL) is one of several specification that attempts to organize information exchange between multiple organizations (or even multiple application within organizations), with an emphasis on public collaboration.

An important characteristic of choreographies is that they are intended for public message exchanges. The goal is to establish a kind of organized collaboration between services representing different service entities, none of the entities necessarily controls the collaboration logic.

The fundamentals concept of exposing business logic through autonomous services can be applied to just any implementation scope. Two services within a single organization, each exposing a simple function, can interact via a basic MEP (Message exchange protocol) to complete a simple task. Two services belonging to different organizations, each exposing functionality form entire enterprise business solutions, can interact via a basic choreography to complete task. Both scenarios involve two services, and both scenarios support SOA implementations.

Therefore choreography can assist in the realization of SOA across organization boundaries while it supports compos-ability, reusability and extensibility. Choreography can also increase organizational agility and discovery. Organizations are able to join into multiple online collaborations, which can dynamically extend or even alter related business processes that integrate with the choreographies.

5.2 Introduction to BPEL4WS or WS-BPEL

In computing, Business Process Execution Language (or BPEL), is a business process language that grew out of WSFL (web service flow language) and XLANG. It is serialized in XML and aims to enable programming in the large. The concepts of programming in the large and programming in the small distinguish between two aspects of writing the type of long-running asynchronous processes that one typically sees in business processes.
In another word BPEL uses for process orchestration.

In BPEL Business process logic is centralized in one location, as opposed to being distributed across and embedded within multiple services.

Following XML shows you BPEL structure, explanation of each part is out of scope of this document:

```
<process>
     <partnerLinks>
      ...
     </partnerLinks>
     <variables>
      ...
     </variables>
     <faultHandlres>
      ...
     </faultHandlres>
     <sequence>
          <receive …>
          <invoke…>
          <reply…>
          ...
     </sequence>
</process>
```

Example:

Following is Hello world example provides from Oracle BEPL tutorial.
In following example we have partner link as client for providing service:

```
<process name="HelloWorld" targetNamespace="http://sopa/example" xmlns:tns="http:acm.org">
     <!—List of services participating in this BPEL process-->
     <partnerLinks>
           <partnerLink name="client" partnerLinkType="tns:HelloWorld" myRole="HelloWorldProvider"/>
     </partnerLinks>
     <!— List of messages and XML documents used with this BPEL process-->
     <variables>
           <!— References to the message passed as input during initiation -->
           <variable name="input" messageType="tns:HelloWorldRequestMessage"/>

           <!—Reference to the message that will be returned to the requestor -->
           <variable name="output" messageType:"tns:HelloWorldresponseMessage"/>
     </variables>

     <!—ORCHESTRATION LOGIC -->
     <sequence name="main">
           <!-- Receive input from requestor -->
           <receive name="receiveInput" partnerLink="client"
                 portType="tns:HelloWorld" operation="process"
                 variable="input" createInstance="yes"/>
           <!— Generate reply to synchronous request-->
           <assign name="assign-1">
             <copy>
                 <from expression="contact(&quot;Hello &quot;,
                                  bpws:getVariableData(&quot;input&quot;,
                                  &quot;payload&quot;,
                                  &quot;tns:HelloWorldRequest/tns:input&quot;))">
                 </from>
                 <to variable="output" part="payload" query="/tns:HelloWorldResponse/tns:result" />
             </copy>
           </assign>
             <reply name="replyOutput" partnerLink="client" portType="tns:Helloworld"
                   operation="process" variable="output"/>
     </sequence>
</process>
```

5.3 Introduction to SOPA

As explained before, Orchestration service layer provides a powerful means by which contemporary service-oriented solutions can realize some key benefits.
One of the most significant advantages of service oriented computing is separation of Logic from other application layers.

By Abstracting business process we get following advantages:

- Application and business services can be freely designed to be process agnostic and reusable.

- The process service has better degree of statefulness, thus changing on services in process flow has no effect on other services.

SOPA stays for *service oriented pipeline architecture*; it is a service oriented framework which has been developed to implement Service composition and service orchestration (business process layer of SOA) , base on creating pipelines and calling services (both eclipse and web services) from pipelines.

5.3.1 What is pipeline?

Pipeline term which has been used in SOPA defines a set of related service calls and intermediate transformations. In another word it provides some mechanism to orchestrate set of logically related services. Pipeline has XML structure and makes a call to service located in pipeline in XML structure format.

Following example shows you a pipeline calling “getRate” operation of “currency exchange” web service belongs to xmethods:

```
<?xml version="1.0" encoding="UTF-8"?>
<pipelines><pipeline name="getRate">
  <parameters>
    <parameter name="country2" type="string"/>
    <parameter name="country1" type="string"/>
  </parameters>
      <call service="at.slife.webservice" operation="wsCall"
            returns="string">
             <parameter name="wsdlLoc" type="string">
                  http://www.xmethods.net/sd/CurrencyExchangeService
            .wsdl
            </parameter>
```

```
            <parameter name="serviceName" type="string">
                  CurrencyExchangeService
            </parameter>
            <parameter name="portName" type="string">
                  CurrencyExchangePort
            </parameter>
            <parameter name="operName" type="string">
                  getRate
            </parameter>
            <parameter type="string"> {country2} </parameter>
            <parameter type="string"> {country1} </parameter>
   </call>
  <serialize type="xml"/>
</pipeline></pipelines>
```

5.3.2 SOPA plug-in model

As noted before generic client component can be used separately from SOPA framework and also it can integrate in SOPA framework as plug-in, before we go further in integration part I prefer to explain about SOPA framework and make an overview about BPEL then at the end we will make some few notes about their differences.

SOPA is type of architecture which is based on eclipse platform and provides a service bus, but SOPA services are not restricted to web services there are three different categories of services (two are eclipse services and one is web services, second one is the main goal of this project) which can integrate in SOPA framework. Here in following an concrete explanation provided.

5.3.2.1 Eclipse services

1. **GUI Plug-ins**

GUI services which have been used as visual eclipse plug-in. They can be use in SOPA framework as SOPA services.

2. **Business Services**

There is a possibility for integrating all types of non GUI eclipse plug-ins, they will be located in SOPA framework as external service.

5.3.2.2 Non Eclipse services

Web Services

Standard web services which are distributed on the web can also bring inside SOPA framework via generic client plug-in. This plug-in is main focus of this document and is core responsible component of executing external web services in SOPA framework, within this feature SOPA framework can act as Service Bus.

Following figure shows you how can SOPA act as Service Bus for three different kind of services

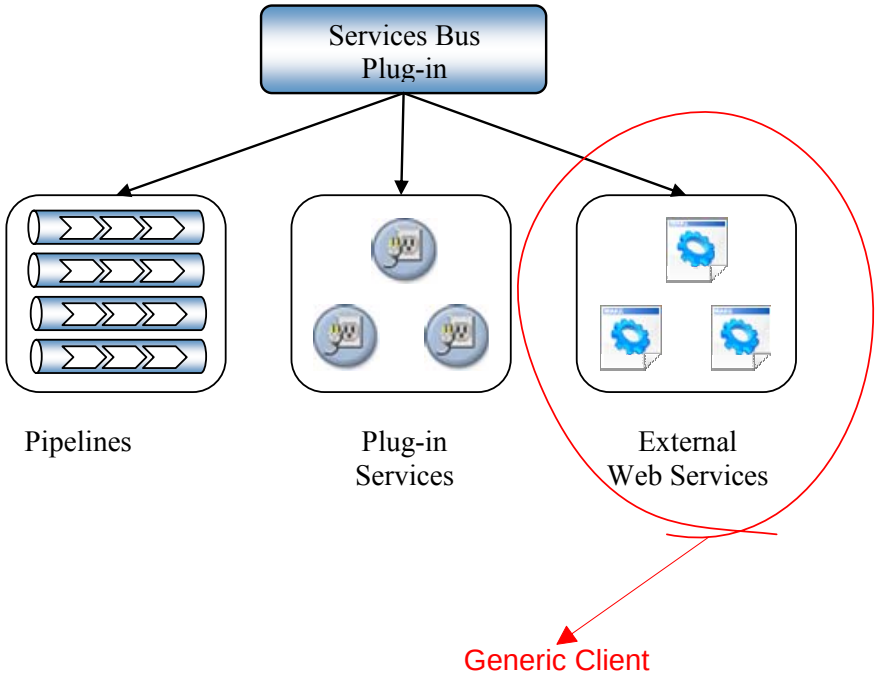


Figure 5-1 SOPA service Bus Plug-in and the location of generic client

5.3.3 SOPA features

As I mentioned before SOPA provided a Service composition and service orchestration like BPEL, I will explained BPEL in this chapter in detail.
SOPA implementation policy is to make integration and service consumption easy to use, in another word I can say SOPA is light weight ESB (Enterprise Service Bus) which is easy to extend. Every kind of web services can be add to SOPA as extension plug-in and there is no or less overhead in service extension and service integration.
The idea of SOPA is because current computer are strong enough to handle different kind of web service operation's call, we can bring interface to desired service on local machine and via that interface (pipeline) in first step fill arguments and in second step call operation.

One another Interesting feature in SOPA is inside SOPA you can add a pipeline to call another pipeline like BPEL that from one partner link you can call another complete BPEL process.

For example following shows you how via one pipeline another pipeline has been called:

```
<pipeline name="internalCall">
          <parameters>
               <parameter name="a" type="int"/>
               <parameter name="b" type="int"/>
          </parameters>
          <call id="node" service="at.slife.test" operation="multiply"
                returns="int">
               <parameter type="int">{a}</parameter>
               <parameter type="int">{b}</parameter>
          </call>
          <call id="node" service="at.slife.test" operation="add"
                returns="int">
               <parameter type="int">{b}</parameter>
               <parameter type="int">
          </call>
          <call service="at.slife.pipeline" operation="complex:square">
                     <parameter type="int">{b}</parameter>
          </call>
          <call id="node" service="at.slife.test@localhost" operation="add"
                returns="int">
                <parameter type="int">
                  {xpath:/html/body/number[1]/text()}
                </parameter>
                <parameter type="int"
                  {xpath:/html/body/number[2]/text()}
                </parameter>
          </call>
          <transform xsl="transformer.xsl"/>
          <serialize type="xml"/>
    </pipeline>
```

In above example the bold part shows you how a pipeline can be called as parameter of another pipeline.

5.3.4 SOPA Service monitoring framework

What is important in Service composition is quality of Service, traditional service quality measurement forms are not sufficient for Service oriented architecture context.

Service Oriented system's quality relies on the quality of each service that it has been composed of. In Service oriented architecture components are not managed centrally, because they are distributed over the network.

Important monitoring factor for service in Service oriented architecture could be **availability**, **correctness** and **response time**.

SOPA monitoring component can be configured to check specific services at required intervals and log the results. This process measure Availability, response time and correctness.

5.3.5 SOPA Alternative services (service backup)

In SOPA each service may be backed up with a set of alternative services, an alternative service may be composition of two or more services.
At system design time the alternative service backup sets can be defined for each service. When original service is not available SOPA will automatically switch to one of the alternative service based on administrator defined criteria.

For example imagine a service which used for currency exchange if this service is not available it should be replaced with a similar service that provide us the same result.

At system design time the alternative service backup sets can be defined for each service. At runtime if the original service is not available or fails, SOPA will automatically switch to one of the alternative services based on administrator defined criteria.
This feature will increase the failure tolerance and level of service reliability.

5.3.6 SOPA implementation example (SemanticLIFE)

SemanticLIFE is a personal information management supporting personal data like emails, contacts web browsing, sessions, chat sessions and fileupload.
Explaining semantic life is out of scope of this document but for more information you can refer to this link: http://storm.ifs.tuwien.ac.at/

The whole SemanticLIFE system is designed as a set of interactive plug-ins that fit into the main application and this guarantees flexibility and extensibility of the SemanticLIFE platform. Communication inside system is Service oriented and via SOPA it is possible to compose complex solutions and scenarios from atomic services of SemanticLIFE plug-ins.

5.3.3 WS-BPEL (BPEL4WS) comparison with SOPA

Both BPEL and SOPA are used for service composition, BPEL4WS is current standard language used for service orchestration and composition and lots vendors

implement tools for it. But there are differences between SOPA and BPEL I try to explain advantage of SOPA over BEPL in following:

- SOPA has potential to include a built-in monitoring frame work, this could be an advantage over BPEL4WS because BPEL lack the unit test tooling support and this is a drawback for BPEL in terms of quality and efficiency. There are test methods available for BPEL but they are platform specific and there is no standard specification available for BPEL4WS monitoring yet, but vendors provide monitoring tools for example oracle BAM(Business Activity monitoring) provides process monitoring and Sensor technology to monitor quality of services. SOPA has a monitoring framework which have been used to measure quality and quantity of web service characteristics in order to decrease failure.

- Another difference between SOPA and BPEL4WS is: BPEL4WS language currently does not support the explicit definition of business process fragments that can be invoked from another (or the same) business process. The only way to approximate similar behavior today is by defining a complete business process as an independent service and invoking it using an <invoke> activity. The fact that the invoked activity is really implemented as another process is completely hidden from the parent process, in other words, there is no chance to establish any coupling of process instance lifecycles. There are some new extension for subprocesses that aim to overcome this problem however it doesn't cover empirical aspects of Service oriented architectures such as quality assurance, statistical based decision making, unit test and monitoring system status, but SOPA has this feature.

Appendix

Restrictions

Please consider following restrictions according to the usage of generic client.

General Restrictions

- This component is designed for JEE5 and it uses JAX-WS library as execution engine so it is not designed for JAX-RPC and of course there is no possibility to use JAX-RPC library for further code manipulation.

- GUI is designed with SWT libraries, Parser is designed with WSDL4J library and Call-WS plug-in part is SOPA framework dependent.

- This component is designed and implemented on windows XP operating system, there should be no restriction to run it on other platforms but I didn't test that.

- There is no port needed to be occupied for executing of this component but because it calls services via HTTP, standard HTTP port will establishes connection to service.

WS Client invocation restrictions

- Most of WSDL files have only one definition element, parser part of generic client component is designed in a way to assume each WSDL has one and only one definition element, this means any WSDL which has more than one definition element inside, is not supported here in this work.

- Complex data types as type definition in WSDL file is not supported in this version so any web service's operation which contains complex XSD type as input is not executable or callable via this component.

- In client invocation of web service we have different ways to call an operation on service. This component is designed to invoke synchronously (one way). Asynchronous invocation or any other form of service invocation is not supported in this version.

Reference

- The Java API for XML-Based Web Services (JAX-WS) 2.0 Final Release

- JSR-110: Java™ APIs for WSDL (JWSDL) Version 1.2

- JSR-173: Streaming API For XML

- Service Oriented Architecture a Field guide to integrating XML and web services by Thomas Erl

- Service Oriented Architecture Cocepts, technology and Design by Thomas Erl

- Semantic Enrichment of Search Result: the Coupling of Semantic Strore and Google Services
by Khabib Mustafa, Amin Andjomshoaa, A Min Tjoa Institue for Software Technology and Interactive Systems TU Viennsa, Austria

- Java Web Services Architecture by James McGovern, Sameer Tyagi, Michael Stevens and Sunil Matthew

- The Java™ EE 5 Tutorial For Sun Java System Application Server Platform Edition 9

- Eclipse Rich Client Platform: Designing, Coding, and Packaging Java™ Applications By Jeff McAffer, Jean-Michel Lemieux

- Note on the Eclipse Plug-in Architecture, By Azad Bolur

- SOAP's Two Messaging Styles By: Rickland Hollar

- JAX-RPC vs JAX-WS By: Russell Butek and Nicholas Gallardo (http://www-128.ibm.com/developerworks/webservices/library/ws-tip-jaxwsrpc.html)

- J2EE Web Services By Richard Monson-Haefel

- XML in Nutshell 3rd Edition By Elliotte Rusty Harold, W. Scott Means

- Oracle BPEL Tutorial http://otn.oracle.com

- SWT/JFace in Action by Matthew Scarpino, Stephen Holder, Stanford Ng, Laurent Mihalkovic