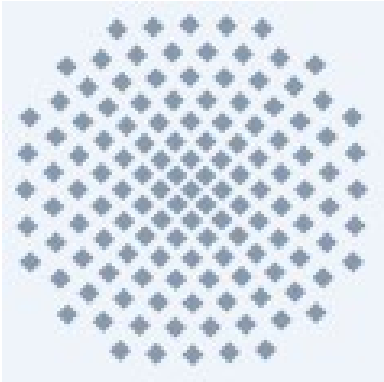


Studienarbeit Nr. 2259

JAX-WS Test Client for SWoM

Ruojin Wen

CR – Classification: H.3.5, H.5.3

Course of Study:	Informatik
Examiner:	Prof. Dr. Frank Leymann
Supervisor:	Dipl.-Phys. Dieter H. Roller
Commenced:	02. Feb 2010
Completed:	02. Aug 2010

Institute of Architecture of Application Systems

University Stuttgart

Directory

1 Introduction

As we known JAX-WS is more and more popular and make the Web Service design much easier. Web Services based on the service-oriented architecture framework serve as the foundation for modern distributed, heterogeneous application. They are perfectly suited as the function layer of the two-level programming model that is characteristic for workflow-based applications.

Workflow-based applications are composed of tow distinct pieces: a process model that describes the sequence in which the different activities making up th process model are being carried (programming in the large) and the individual components that implement the various activities (programming in the small). In the Web services environment, process models are described using the Web Services Business Process Execution Language(WS-BPEL).

The purpose of a workflow management system (WFMS) implementing the WS-BPEL specification is to manage the life cycle of business processes, to navigate through the associated process models, and to invoke the appropriate Web services. The Stuttgart Workflowmaschine (SwoM) partially implements the appropriate WS-BPEL standard.

1.1Purpose and procedure of this Studienarbeit

The purpose of this Studienarbeit is to write a JAX-WS based test client for carrying out appropriate performance tests. The input to test client is an XML file that specifies the characteristics of the test to be carried out, such as the size of the variables, the level of parallel requests, or the generation of correlation set.

1.2 Construction of this Document

In this report we will first talk about the IBM Rational Application Developer, illustrate it's advantages. Then we will see JAX – WS and compare JAX – WS and JAX -RPC. After that is XMLBeans, the new technology for developer who work with XML. At last we will see the core of this Studienarbeit.

2 RAD

IBM Rational Application Developer extends Eclipse. RAD support for Communication Enabled widgets and Service Component Architecture tools and visual development. In addition RAD supply WEB2.0 development features for visual development of responsive Rich Internet Applications with AJAX.

We can download the RAD in (www.ibm.com). Then under the information for the installation of RAD we can install RAD step by step. After that we will be to enjoy the magic of RAD. You will see that the productivity can be increased and the cost of time and development will be deceased respectively.

2.1 Advantage

As a new IDE RAD has many advantages. Here we introduce the major advantage we use in this Studienarbeit.

We see, by RAD we can access to nly the features we need, get started and installation guide, content assist.

To Web Services Development,RAD support Web Services Reliable Messaging (WS-RM), Web Services Addressing (WS-Addressing) and SOAP Message Transmission Optimization Mechanism (MTOM), JAX - RPC, JAX - WS. In addition RAD support for transforming Web Services from existing artifacts, such as Java beans, and EJB components.It provide WSDL editor and can deploy and test Web services into WebSphere Application Server.

To Web 2.0 Development, RAD provide Ajax development with Dojo Toolkit and IBM extensions, JavaScript Editor and so on.

To XML Development, we can use by RAD provided editor to simply create XML schema and generate XML instance.

As general speaking, the advantages of RAD can be summed up as four aspect:

1.Productivity Enhancement Features.

2.Web Services Development-Comprehensive Web services and SOA development tools simplify and automate the process of building WS-I compliant, interoperable Web services applications.

3.Web 2.0 Development – Features to aid the development of responsive Rich Internet Applications using the latest development and debugging tools.

4.XML Development – Comprehensive tools for creating, editing and transforming XML documents and integrating relational data and XML.

3 JAX-WS

By the development of Web service, we always improve technology. Now we get a big step, from JAX – RPC to JAX – WS . The developer benefit from JAX – WS, for example: simplify the development and deployment of Web service.

At first we take a look at Web services.

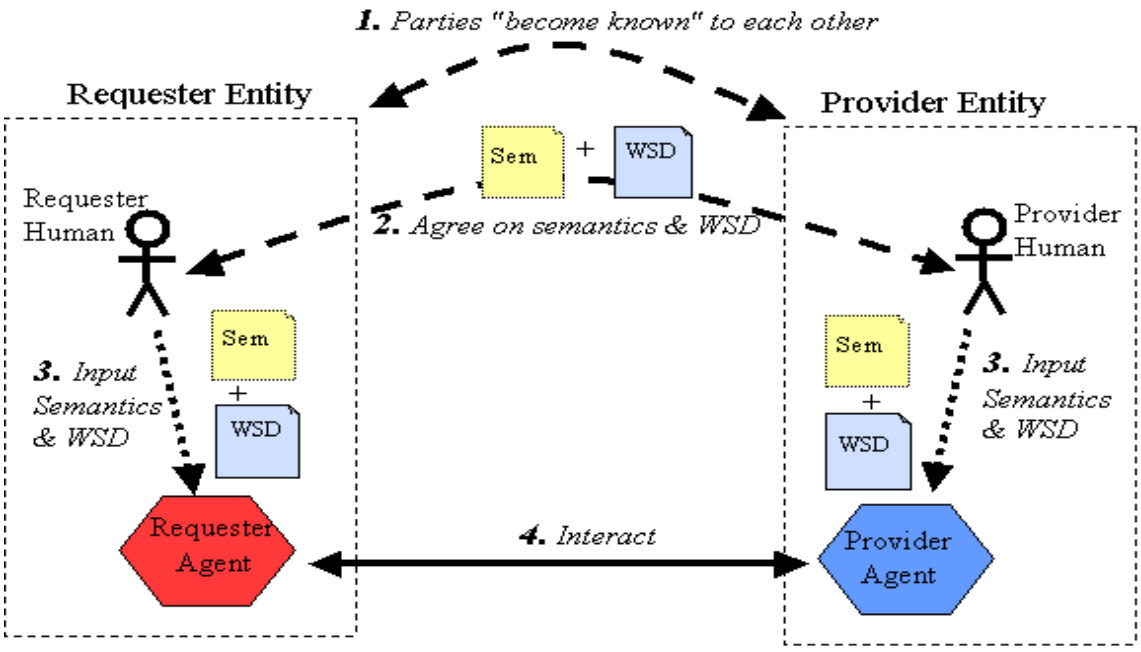
3.1 Web Services

As we know, Web Services is most popular technology that implement SOA (Service – oriented Architecture). Just as it's name implies, Web Service provide the services which is needed and must be found by the Requester. Compare to the old technology like CORBA, Web Service is platform independent and can be at run – time implemented.

W3C define the Web Services as following:

> [Definition: A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.]

Through the definition we see that Web services is implemented by an agent.



authority: http://www.w3.org/TR/ws-arch/#gengag

Figure3.1

The figure 3.1 describe how the requester and provider communicate with each other.

The above figure illustrate that usually the provider and the requester are an human or a organization. The requester want to use the services by the provider entity provided. As the requester, we only need give our requests to the Requester Agent, in the same way the provider put his service on the Provider Agent. Then Requester Agent and Provider Agent interact with each other. I think, because of the “Agent Policy”, the role played by requester and provider can be simply changed when we need. That means requester becomes provider, provider becomes requester, so the flexibility is improved.

In the above definition WSDL is mentioned. WSDL describe the Web Services. In fact WSDL is an XML file. We can distinguish WSDL from two part, one is abstract, the other is concrete.

Abstract part is composed from WSDL – Message, WSDL – Port Type, WSDL – Operation; Concrete part is composed from WSDL – Types, WSDL – Binding, WSDL – Port, WSDL – Service.

List 3.1

```
1   <?xml version="1.0" encoding="UTF-8"?>
2   <definitions name="HelloService"
3    targetNamespace="http://www.ecerami.com/wsdl/HelloService.wsdl"
4   xmlns="http://schemas.xmlsoap.org/wsdl/"
5   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
6   xmlns:tns="http://www.ecerami.com/wsdl/HelloService.wsdl"
7   xmlns:xsd="http://www.w3.org/2001/XMLSchema">

8   <message name="SayHelloRequest">
9         <part name="firstName" type="xsd:string"/>
10  </message>
11  <message name="SayHelloResponse">
12              <part name="greeting" type="xsd:string"/>
13  </message>

14  <portType name="Hello_PortType">
15              <operation name="sayHello">
16           <input message="tns:SayHelloRequest"/>
17        <output message="tns:SayHelloResponse"/>
18        </operation>
19  </portType>

20  <binding name="Hello_Binding" type="tns:Hello_PortType">
21              <soap:binding style="rpc"
22        transport="http://schemas.xmlsoap.org/soap/http"/>
23        <operation name="sayHello">
24               <soap:operation soapAction="sayHello"/>
25        <input>
26          <soap:body
27              encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
28               namespace="urn:examples:helloservice"
29              use="encoded"/>
30        </input>
31              <output>
32        <soap:body
33              encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
```

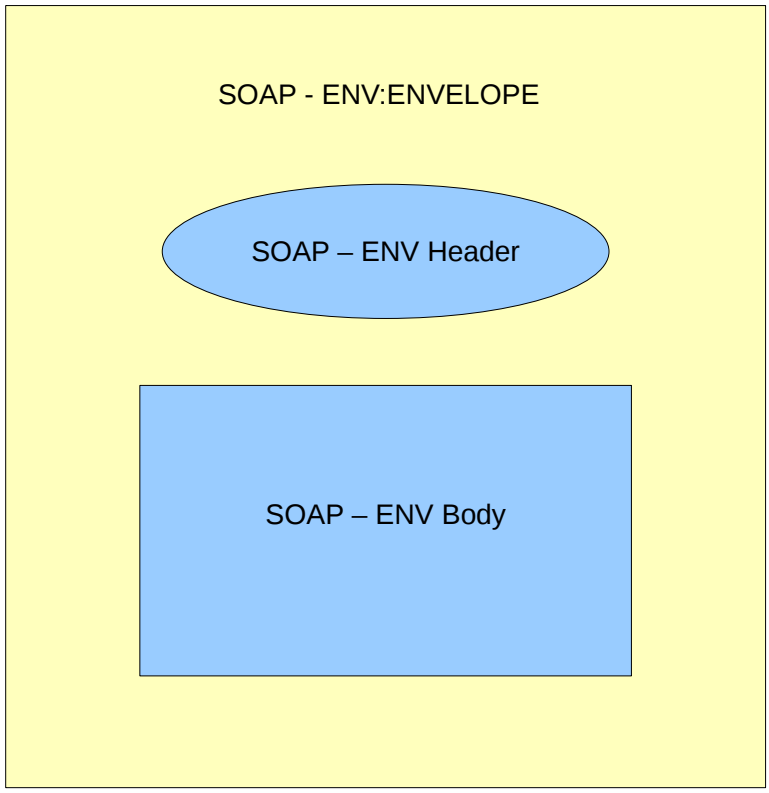
```
34                 namespace="urn:examples:helloservice"
35                 use="encoded"/>
36           </output>
37     </operation>
38  </binding>

39  <service name="Hello_Service">
40     <documentation>WSDL File for HelloService</documentation>
41     <port binding="tns:Hello_Binding" name="Hello_Port">
42        <soap:address
43           location="http://localhost:8080/soap/servlet/rpcrouter"/>
44      </port>
45     </service>
46   </definitions>
```

List 3.1 tells us how to use WSDL to implement the classical “Hello,World”.

3.2 SOAP

SOPA, Simple Object Access Protocol is a Message – Framework and use to exchange the information of Web Service. SOAP – Message relies on XML. The following figure show us the structure of SOPA:



10/7/13 1

Figure 3.2

Figure 3.2 tells us the structure of SOAP Message.

We see, There is three parts in a SOAP message, Envelope, Header, and Body.

The Envelope is the root of the SOAP message and indicates that this XML file as SOAP message.

The Header includes the application – specific information like authentication, etc.

The Body contains the actual information which we need exchange.

Following is a simple example of SOAP message:

List 3.2

```
1   <?xml version="1.0"?>
2   <soap:Envelope
3   xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
4   soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
5   <soap:Body xmlns:m="http://www.example.org/car">
6     <m:GetCarPrice>
7       <m:CarName>Porsche</m:CarName>
8     </m:GetCarPrice>
9   </soap:Body>

10  </soap:Envelope>
```

List 3.2 tells us how to create a SOAP Message. We see that SOAP Message is XML based. As figure 3.2 shows the SOAP Head and SOAP Body are inside of SOAP Envelope.

3.3 Description of JAX – WS

The Java API for XML Web Services (JAX-WS) is a Java programming language API for creating web services and is the next generation Web services programming model. JAX – WS divide form JAX – RPC. JAX – WS is a Part of the Java EE platform from Sun Microsystems. Using JAX-WS , programmer can choose RPC-oriented or Message oriented to implement Web Service. JAX-WS supports a better platform independence for Java application and take advantage of the dynamic proxy mechanism to provide a formal delegation model with a pluggable provider.

In JAX-WS we use annotations to annotate Java class with meta data, so that indicate the Java class is a Web Service. The Web service Meta data for the Java Platform specification and annotations defined by the JAX-WS 2.1 specification. Using annotations we can let normal Java class and method to be used for Web service. For example:

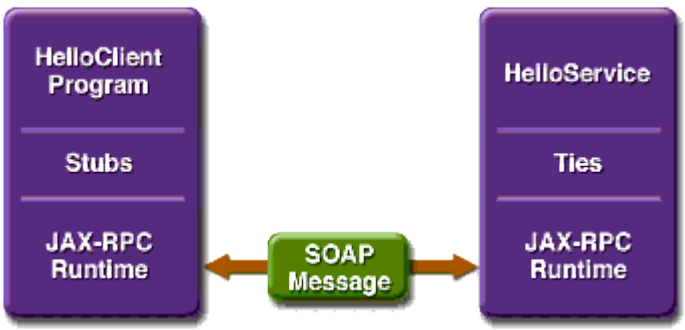
List 3.3

```
1    @WebService()
2    public class HelloWorld {
3    private String word  =  new String ("Hello, World");
4       public void HelloWorld () {
5    @WebMethod ()
6     public String sayHello (String name) {
7       return word + name + "." ;
8         }
9    }
```

List 3.3 tells us how a normal Java class and method can be transferred to a Web services and Web method.

In this example, we annotate the HelloWorld class as a web service endpoint by using the @WebService() annotation. This annotation tells the server runtime environment to expose all public methods on that bean as Web service. The Method sayHello is annotated respective with the annotation @WebMethod. By using this annotation client can call this method.

Also, using annotations make better the development of Web services within a team structure because we do not have to define every Web service in a single or common deployment descriptor as required with JAX- RPC Web service.



Authority:http://java.sun.com/webservices/docs/2.0/tutorial/doc/JAXWS3.html

Figure 3.3

Figure 3.3 illustrate how JAX-WS communicate between the web service and the client

and the structure of client and service.

3.4 Difference between JAX-WS and JAX-RPC

As we known the foundation of JAX – WS is JAX – RPC, now we will discuss about the same and difference between JAX-WS and JAX-RPC.

First is the same. Because JAX - WS still support SOAP 1.1 over HTTP 1.1 and WSDL 1.1, there is no impact to the interoperability and the knowledge about WSDL 1.1 we learned from JAX - RPC is still useful.

Now we will talk over the difference. As before said beside SOAP1.1 JAX-WS also support SOAP1.2.

JAX - WS support HTTP binding, but JAX - RPC ignored it.

Now we use XMLBeans as data mapping model of JAX - WS, XMLBeans support all schema types, and the data mapping model of JAX - RPC only can handle 90 percent of schema types.

The deployment of Java lead to two difference. One is the Java features of JAX - WS (Java 6.0) and JAX - RPC(java 1.4) is different. The other is that JAX - WS use Java 6.0 features for interface mapping model. In addition the interface mapping model of JAX - WS introduces asynchronous functionality. (notice that,General coming speaks, the difference of interface mapping model between JAX - WS and JAX - RPC is small)

In the dynamic programming model aspect we observe client and server. JAX-WS' s dynamic client model is quite different from JAX-RPC' s. Because JAX – WS provides message – oriented and dynamic asynchronous functionality. JAX – WS also has dynamic server, which JAX-RPC don't.

In addition, JAX-WS support MTOM (Message Transmission Optimization Mechanism)

So because of following reasons we should choose JAX-WS:

When we want to use the new message-oriented APIs.

When we want to use MTOM to send attachment data.

When we want to use an asynchronous programming model in your Web service client.

When we want better support for XML schema through XMLBeans.

When we have to use clients or services that can handle SOAP1.2 messages.

3.5 Principle of JAX-WS

First we should know the precondition for using JAX-WS. If we use the new APIs in the Java SE 6 platform, naming package : javax.xml.ws, then we can use JAX- WS to build web service.

Now we will see how to create Web service and client using JAX-WS.

First we need a class with a or more method that we use as web service.

We write such a class:

```
1    package Hello

2    public class HelloWorld {

3        public void sayHello () {

4            System.out.println (“ Hello, World!”);

5        }
6    }
```

Now we must import the package: javax.jws.Webservice and add the annotation: @ WebService at the beginning to let this class as a web service. The following code shows us how:

```
1    package Hello

2    import  javax.jws.Webservice;
3    @ WebService
4    public class HelloWorld {

5        public void sayHello () {

6            System.out.println (“ Hello, World!”);

7        }
8    }
```

We can also use the static publish () method of the javax.xml.ws.Endpoint to publish this class as web service in the specified context root:

```
1    import javax.xml.ws.Endpoint;

2    public static void main (String [] args) {

3                Endpoint publish (
4                    “http ://localhost: 8080/WebServiceExample/HelloWorld”,
5                        new HelloWorld());

6    }
```

Now we have know how to let a Java class be to a web service. Following we will talk about JAX-WS Client.

3.6 JAX-WS Client

We have a support of the dynamic invocation of service endpoint operations in JAX – WS .

The advance of by JAX – WS provided new dynamic Dispatch client API is more generic and offers more flexibility than JAX-RPC. Because of the Dispatch client interface, javax.xml.ws.Dispatch is an XML messaging oriented client, it is good for the advanced XML developers.

The Dispatch client API provide two mode when send data, one is PAYLOAD, the other is MESSAGE (in this Studienarbeit we use MESSAGE mode). The difference between two mode is the responsibility of Dispatch client. That means in PAYLOAD the Dispatch client is only responsible for providing the contents of the <soap :Body> and JAX-WS includes the input payload in a <soap : Envelope> element. Respectively when we using the MESSAGE mode, the Dispatch client is responsible for providing the entire SOPA envelope. In this Studienarbeit we will use the MESSAGE mode, which we will show after.

When we use the Dispatch client API, messages or payload must be construct as XML and we must get the detailed knowledge of the message or message payload. There is two kind of bindings, first is HTTP bindings while using Source objects; second named XML bindings objects, or data source objects. Four kinds of objects are supported by Dispatch client.

By using different binding models we can choose five objects.

The first is java.xml.transform.Source. When we directly use XML APIs,we choose this object. Both of SOAP and HTTP bindings support this object.

The second is JAXB object. But by the deployment of XMLBeans I think XMLBeans will completely take place of JAXB. So the third is XMLBeans object. For JAXB or XMLBeans we use SOAP and HTTP bindings.

The fourth named java.xml.SOAPMessage is used for that the clients can work with SOAP messages.

The last named javax.activation.DataSource is used when clients work Multipurpose Interne Mail Extension (MIME) message. Only HTTP binding support this object.

The environment of the Dispatch API is Java SE Runtime environment (JRE).

Following we will talk about the procedure in step by step.

First we need decide if we use dynamic client to send data in PAYLOAD or MESSAGE mode.

Second we make a service instance and add at least one port to it. The port get protocol binding and service endpoint address information.

Then we create a Dispatch<T> object with the Service.Mode.PAYLOAD method or the

Service.Mode.MESSAGE method.

After that we configure the request context properties on the javax.xml.ws. BindingProvider interface.

Last step is that we set the jaxws.response.throwExceptionIfSOAPFault property on the RequestContext option for the Dispatch API to Boolean.TRUE if you do not want the JAX-WS runtime to throw a SOAPFaultException.

The following example will show us the details of the creation of Dispatch client.

At first we create a Dispatch object:

```
1    Dispatch<OMElement> dispatch = service.createDispatch(
2    portName, OMElement.class, Mode.MESSAGE);
3    BindingProvider bp = (BindingProvider)dispatch;
4    bp.getRequestContext().put(
5    "jaxws.response.throwExceptionIfSOAPFault", Boolean.FALSE);
```

Compose the client request message for the dynamic client.

Then invoke the service endpoint with the Dispatch client either synchronously or asynchronously.

Then process the response message from the service.

We have developed a dynamic JAX-WS client using the Dispatch API.

The following code illustrates the steps to create a Dispatch client and invoke a EchoService service endpoint.

```
1    String endpointUrl = "...";

2    QName serviceName = new QName("http://com/ibm/was/wssample/echo/",
3    "EchoService");
4    QName portName = new QName("http://com/ibm/was/wssample/echo/",
5     "EchoServicePort");
```

Create a service and add at least one port to it:

```
1    Service service = Service.create(serviceName);
2    service.addPort(portName, SOAPBinding.SOAP11HTTP_BINDING, endpointUrl);
```

Create a Dispatch instance from a service:

```
1    Dispatch<SOAPMessage> dispatch = service.createDispatch(portName,
2    SOAPMessage.class, Service.Mode.MESSAGE);
```

Create SOAPMessage request. compose a request message:

```
1    MessageFactory mf = MessageFactory.newInstance(SOAPConstants.SOAP_1_1_PROTOCOL);
```

Create a message. This example works with the SOAPPART.

```
1    SOAPMessage request = mf.createMessage();
2    SOAPPart part = request.getSOAPPart();
```

Obtain the SOAPEnvelope and header and body elements:

```
1    SOAPEnvelope env = part.getEnvelope();
2    SOAPHeader header = env.getHeader();
3    SOAPBody body = env.getBody();
```

Construct the message payload:

```
1    SOAPElement operation = body.addChildElement("invoke", "ns1",
2    "http://com/ibm/was/wssample/echo/");
3    SOAPElement value = operation.addChildElement("arg0");
4    value.addTextNode("ping");
5    request.saveChanges();
```

Invoke the service endpoint:

```
1    SOAPMessage response = dispatch.invoke(request);
```

At last we process the response

3.7 Conclusion

At last we take a look at the advantage of JAX – WS. Following is the major aspect:

1. Developer use JAX – WS typically in conjunction with other technologies.

2. Although JAX – WS provides remote procedure calls or messages using XML – based protocols such as SOAP, but hides SOAP' s innate complexity behind a Java – based API.

3. Through JAX – WS runtime system API calls and matching replies are converted to and from SOAP messages.

4. Invoking Web services asynchronously.

4 Design of this Studienarbeit

In this Studienarbeit the web service is invoked by XML files which we can read from database. The XML files contain the properties that implement the web service, for example the Qname, the NameSpace and so on.

4.1 Basic Concept

The basic concept of this Studienarbeit is that we read a XML file from anywhere and then parse it. As we known all XML file come from XML schema. A XML schema file is also a XML file and it can generate relative XML files. So if we can parse a XML schema file, we can also parse the by this XML schema generated XML files. So it is important that we create a XML schema file, then this XML schema file generates XML files, which contains parameters and can invoke the web service, then we parse the XML files so that we can use the in XML files wrote parameters to invoke the web service and then we create a relative Java files so that we can change the parameters in XML files like a Java object.

To parse XML files we need a smart tool named XMLbeans. We will talk about XMLbeans in next.

4.2 XMLBeans

XMLBeans is the core tool we need in this Studienarbeit.

4.2.1 Description

XMLBeans is a technology that through XML Schema handle XML files. XMLBeans compile schema (.xsd file) to generate Java type, which will be binding to XML instance data. We use XMLBeans like using any Java class or interface. So we can access XML instance data. We get some methods like getFoo(), setFoo(), just as with Java program. While using XMLBeans we can access XML instance with strongly Java type. We don't have to write the original program again, we only need to compile the .xsd Schema that generate a new .jar and import it to our program.

XMLBeans has two characteristics, Large XML Schema support and Large XML Infoset support. For this Studienarbeit Large XML Schema support is very important, because we must work with Java.

As we know the objective of XMLBeans has been to be applicable in all non-streaming XML programming situations. So programmer should change XML Schema into a set of Java class. We will discuss the needed related APIs after.

4.2.2 History

David Bau was the chief designer for the XMLBeans 1.0 project while he was working for BEA. XMLBeans started on the grounds of XMLMaps, an XML binding tool included in previous BEA WebLogic products. XMLBeans was originally developed as part of the

proprietary BEA WebLogic Workshop Framework, but it was obvious from interviews conducted when it was first announced on January 27, 2003, that BEA wanted it to become an open standard. At that time it was not decided which organisation BEA wanted to involve in the standardisation effort. Later that year it was donated to the Apache Software Foundation.

January 27, 2003: BEA announces XMLBeans as a technology preview.
September 24, 2003: BEA donates XMLBeans to the Apache Software Foundation where it joins the Apache Incubator Project.
April 23, 2004: XMLBeans Version 1.0.2 is released. This is the first release from the incubator project.
June 25, 2004: XMLBeans graduated out of the Apache Incubator Project to become top level project.
June 30, 2005: XMLBeans Version 2.0 is released.
November 16, 2005: XMLBeans Version 2.1 is released.
June 23, 2006: XMLBeans Version 2.2 is released.
June 1, 2007: XMLBeans Version 2.3 is released.
December 14,2009 XMLBeans Version 2.5.0 is released.

4.2.3 Install XMLBeans

Following we illustrate how to Install XMLBeans. First we should download XMLBeans and deploy it.

We can get the new release Apache XMLBeans 2.5.0 (December 14, 2009) from http:// xmlbeans.apache.org/.

Note that XMLBeans 2.5.0 is still compatible with JDK 1.4. So we need xmlbeansqname.jar on classpath of our application alongside xbean.jar.

Now we can step by step install XMLBeans.

1. my computer → advance → environment.

2. add arivable XMLBEANS_HOME. When we deploy XMLBeans in C: the value is C :\ xmlbeans.2.5.0.

3. We add xmlbeans\bin into PATH.

Now we can use XMLBeans to work simply.

4.2.4 XMLBeans Command

Next we see some important command and related options of XMLBeans.

1. scomp (Schema Compiler) compiles the .xsd file into Java XMLBeans class and meta data.

Syntax is: scomp [options] [dirs]* [schemaFile.xsd]* [service.wsdl]* [config.xsdconfig]*

Options:

-src. Target directory for generated JAVA files.

-out [jarFileName]. The name of the output JAR that will contain the result of compilation. The default is "xmltypes.jar". jarFileName — The name for the JAR containing generated files.

-complier. Path to external Java compiler.

-d. Target directory for CLASS and XSB files.

2. scopy (Schema Copier) copies the XML schema at the specified URL to the specified file.

Syntax: schemacopy sourceurl [targetfile]

Options:

sourceurl. The URL at which the schema is located.

4.2.5 APIs of XMLBeans

To accomplish the above objectives, XMLBeans gives us three APIs.

For this Studienarbeit the XMLObject is most important, because this API generate Java class that devide from XML Schema.

The second API is XMLCursor.The developer use this API when there is only XML document but no XML Schema in our hand.

The last API named SchemaType. When we need to reflect the underlying schema meta information, this API provide a full XML Schema object model.

4.2.6 Conclusion

At last let's make a conclusion to the advantages of XMLBeans. We compare the difference between DOM, SAX, JAXB, and XMLBeans:

1. When we use DOM, an in - memory tree for the entire document is generated, so that when the content of the document is very large, our work with DOM will be very memory intensive and the efficiency is reduced. But XMLBeans is good at performance,when the developer do incremental unmarshalling, and XMLBeans give us xget methods to access built -in schema data types.

2. The advantage of SAX is that SAX is less memory intensive compared to DOM. The disadvantage is that SAX needs callback methods for event handlers, However this is not required by XMLBeans.

3. Although JAXB and Castor can work like XMLBeans by binding XML and JAVA, both can not total completely support schema . However complete schema support is exactly the one biggest advantage of XMLBeans. In addition the developer can with XMLBeans access the full XML infoset. This is good for that when the order of the elements or comments is critical for an application.

4. XMLBeans in addition support that on - time validation of the XML instance can be parsed and provide XML cursors and XQuery expressions.

4.3 Details of Studienarbeit

Now we know how we can with XMLBeans make the work within XML instance simple. Following we will take a look at the details of this Studienarbeit, for example: the core code.

As above discussed, to implement the test , we need at first an XML Schema, and then generate from this schema an XML – Instance. From the generated XML – Instance we get the related information that invoke Web services.

At first we will see the code for XML Schema:

List 4.3.1:

```
1   <?xml version="1.0" encoding="UTF-8"?>

2   <xsd:schema
3   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
4   targetNamespace="http://org/myXMLSchema"
5   xmlns:tns="http://org/myXMLSchema"
6   elementFormDefault="qualified">

7    <xsd:element name="ClientTestDescriptor" >

8          <xsd:complexType >

9             <xsd:sequence>

10               <xsd:element name="NumberOfParallelInvoke" type="xsd:integer"/>

11               <xsd:element name="MessageSize" type="xsd:integer"/>

12               <xsd:element name="URL" type="xsd:string" />

13               <xsd:element name="MessageContent" type="xsd:string"/>

14               <xsd:element name="TargetNameSpace" type="xsd:string"/>

15               <xsd:element name="SeviceName" type="xsd:string"/>

16            </xsd:sequence>

17         </xsd:complexType>
```

```
18      </xsd:element>
19    </xsd:schema>
```

List 4.3.1 give us the XML schema, which generates the XML file.

In this schema The global root element named ClientTestDescriptor and we define “ TargetNameSpace” as string type, “ ServiceName” and “URL” also as string type. These elements are required to invoke Web services.

After write schema, we use XMLBeans to compile schema into Java interface.
We use the following command to compile the schema:

scomp - out xmlSchema.jar myXMLSchema.xsd

In this command,scomp is the schema compiler, we use the – out option for the name of the output jar, xmlSchema.jar is the output jar.

The above command compiles myXMLSchema.xsd to the XMLBeans interface and classes, and then jars them into myXMLSchema.jar.

After compilation process following important Interfaces are generated:

```
1   public interface ClientTestDescriptorDocument extends
org.apache.xmlbeans.XmlObject{...}
```

This interface represents the document element. This elment is generated for global elements. In this Studienarbeit, it is generated for ClientTestDescriptor.

```
2  public interface ClientTestDescriptor extends org.apache.xmlbeans.XmlObject {…}
```

This interface describes the global root element ClientTestDescriptor.

In interface ClientTestDescriptor following methods are generated:

List 4.3.2:

```
1 /**
2       * Gets the "NumberOfParallelInvoke" element
3       */
4      java.math.BigInteger getNumberOfParallelInvoke();

5      /**
6       * Gets (as xml) the "NumberOfParallelInvoke" element
7       */
8      org.apache.xmlbeans.XmlInteger xgetNumberOfParallelInvoke();

9      /**
```

```
10      * Sets the "NumberOfParallelInvoke" element
11      */
12     void setNumberOfParallelInvoke(java.math.BigInteger numberOfParallelInvoke);

13     /**
14     * Sets (as xml) the "NumberOfParallelInvoke" element
15      */
16     void xsetNumberOfParallelInvoke(org.apache.xmlbeans.XmlInteger numberOfParallelInvoke);

17     /**
18      * Gets the "MessageSize" element
19     */
20     java.math.BigInteger getMessageSize();

21     /**
22      * Gets (as xml) the "MessageSize" element
23      */
24     org.apache.xmlbeans.XmlInteger xgetMessageSize();

25     /**
26      * Sets the "MessageSize" element
27      */
28     void setMessageSize(java.math.BigInteger messageSize);

29     /**
30      * Sets (as xml) the "MessageSize" element
31      */
32    void xsetMessageSize(org.apache.xmlbeans.XmlInteger messageSize);

33     /**
34      * Gets the "URL" element
35      */
36     java.lang.String getURL();

37     /**
38      * Gets (as xml) the "URL" element
39      */
40     org.apache.xmlbeans.XmlString xgetURL();

41     /**
42      * Sets the "URL" element
43      */
44     void setURL(java.lang.String url);

45     /**
46      * Sets (as xml) the "URL" element
47      */
48     void xsetURL(org.apache.xmlbeans.XmlString url);

49     /**
50      * Gets the "MessageContent" element
51      */
52     java.lang.String getMessageContent();

53     /**
54      * Gets (as xml) the "MessageContent" element
55      */
56     org.apache.xmlbeans.XmlString xgetMessageContent();

57     /**
58      * Sets the "MessageContent" element
59      */
60     void setMessageContent(java.lang.String messageContent);

61     /**
62      * Sets (as xml) the "MessageContent" element
63      */
```

```
64      void xsetMessageContent(org.apache.xmlbeans.XmlString messageContent);

65      /**
67       * Gets the "TargetNameSpace" element
68       */
69      java.lang.String getTargetNameSpace();

70      /**
71      * Gets (as xml) the "TargetNameSpace" element
72       */
73      org.apache.xmlbeans.XmlString xgetTargetNameSpace();

74      /**
75       * Sets the "TargetNameSpace" element
76       */
77      void setTargetNameSpace(java.lang.String targetNameSpace);

78      /**
79       * Sets (as xml) the "TargetNameSpace" element
80       */
81      void xsetTargetNameSpace(org.apache.xmlbeans.XmlString targetNameSpace);

82      /**
83       * Gets the "SeviceName" element
84       */
85      java.lang.String getSeviceName();

86      /**
87       * Gets (as xml) the "SeviceName" element
88      */
89      org.apache.xmlbeans.XmlString xgetSeviceName();

90      /**
91       * Sets the "SeviceName" element
92       */
93      void setSeviceName(java.lang.String seviceName);

94      /**
95       * Sets (as xml) the "SeviceName" element
96       */
97      void xsetSeviceName(org.apache.xmlbeans.XmlString seviceName);
```

List 4.3.2 shows us the methods, which we can use to get or set the parameter. These methods will be overridden.

The annotation above method illustrate the function of every method. These all methods are overridden in class ClientTestDescriptorDocumentImpl:

List 4.3.3

```
1   public java.math.BigInteger getNumberOfParallelInvoke()
2     {
3         synchronized (monitor())
4         {
5             check_orphaned();
6             org.apache.xmlbeans.SimpleValue target = null;
7             target =
```

```
8     (org.apache.xmlbeans.SimpleValue)get_store().find_element_user(NUMBEROFPARALLELINVOKE$0, 0);
9             if (target == null)
10            {
11                return null;
12            }
13            return target.getBigIntegerValue();
14        }
15    }

16    /**
17     * Gets (as xml) the "NumberOfParallelInvoke" element
18     */
19    public org.apache.xmlbeans.XmlInteger xgetNumberOfParallelInvoke()
20    {
21        synchronized (monitor())
22        {
23            check_orphaned();
24            org.apache.xmlbeans.XmlInteger target = null;
25            target =
26    (org.apache.xmlbeans.XmlInteger)get_store().find_element_user(NUMBEROFPARALLELINVOKE$0, 0);
27            return target;
28        }
29    }

30    /**
31     * Sets the "NumberOfParallelInvoke" element
32     */
33    public void setNumberOfParallelInvoke(java.math.BigInteger numberOfParallelInvoke)
34    {
35        synchronized (monitor())
36        {
37            check_orphaned();
38            org.apache.xmlbeans.SimpleValue target = null;
39            target =
40    (org.apache.xmlbeans.SimpleValue)get_store().find_element_user(NUMBEROFPARALLELINVOKE$0, 0);
41            if (target == null)
42            {
43                target =
44    (org.apache.xmlbeans.SimpleValue)get_store().add_element_user(NUMBEROFPARALLELINVOKE$0);
45            }
46            target.setBigIntegerValue(numberOfParallelInvoke);
47        }
45    }

49    /**
50     * Sets (as xml) the "NumberOfParallelInvoke" element
51     */
52    public void xsetNumberOfParallelInvoke(org.apache.xmlbeans.XmlInteger numberOfParallelInvoke)
53    {
54        synchronized (monitor())
55        {
56            check_orphaned();
57            org.apache.xmlbeans.XmlInteger target = null;
58            target =
59    (org.apache.xmlbeans.XmlInteger)get_store().find_element_user(NUMBEROFPARALLELINVOKE$0, 0);
60            if (target == null)
61            {
62                target =
63    (org.apache.xmlbeans.XmlInteger)get_store().add_element_user(NUMBEROFPARALLELINVOKE$0);
64            }
65            target.set(numberOfParallelInvoke);
66        }
67    }

68    /**
69     * Gets the "MessageSize" element
70    */
```

```
71      public java.math.BigInteger getMessageSize()
72      {
73          synchronized (monitor())
74          {
75              check_orphaned();
76              org.apache.xmlbeans.SimpleValue target = null;
77              target = (org.apache.xmlbeans.SimpleValue)get_store().find_element_user(MESSAGESIZE$2, 0);
78              if (target == null)
79              {
80                  return null;
81              }
82              return target.getBigIntegerValue();
83          }
84      }

85      /**
86       * Gets (as xml) the "MessageSize" element
87       */
88      public org.apache.xmlbeans.XmlInteger xgetMessageSize()
89      {
90          synchronized (monitor())
91          {
92              check_orphaned();
93              org.apache.xmlbeans.XmlInteger target = null;
94              target = (org.apache.xmlbeans.XmlInteger)get_store().find_element_user(MESSAGESIZE$2, 0);
95              return target;
96          }
97      }

98      /**
99       * Sets the "MessageSize" element
100       */
101      public void setMessageSize(java.math.BigInteger messageSize)
102      {
103          synchronized (monitor())
104          {
105              check_orphaned();
106              org.apache.xmlbeans.SimpleValue target = null;
107              target = (org.apache.xmlbeans.SimpleValue)get_store().find_element_user(MESSAGESIZE$2, 0);
108              if (target == null)
109              {
110                  target = (org.apache.xmlbeans.SimpleValue)get_store().add_element_user(MESSAGESIZE$2);
111              }
112              target.setBigIntegerValue(messageSize);
113          }
114      }

115      /**
116       * Sets (as xml) the "MessageSize" element
117       */
118      public void xsetMessageSize(org.apache.xmlbeans.XmlInteger messageSize)
119      {
120          synchronized (monitor())
121          {
122              check_orphaned();
123              org.apache.xmlbeans.XmlInteger target = null;
124              target = (org.apache.xmlbeans.XmlInteger)get_store().find_element_user(MESSAGESIZE$2, 0);
125              if (target == null)
126              {
127                  target = (org.apache.xmlbeans.XmlInteger)get_store().add_element_user(MESSAGESIZE$2);
128              }
129              target.set(messageSize);
130          }
131      }

132      /**
133       * Gets the "URL" element
```

```
134      */
135     public java.lang.String getURL()
136     {
137         synchronized (monitor())
138         {
139             check_orphaned();
140             org.apache.xmlbeans.SimpleValue target = null;
141             target = (org.apache.xmlbeans.SimpleValue)get_store().find_element_user(URL$4, 0);
142             if (target == null)
143             {
144                 return null;
145             }
146             return target.getStringValue();
147         }
148     }

149     /**
150      * Gets (as xml) the "URL" element
151      */
152     public org.apache.xmlbeans.XmlString xgetURL()
153     {
154         synchronized (monitor())
155         {
156             check_orphaned();
157             org.apache.xmlbeans.XmlString target = null;
158             target = (org.apache.xmlbeans.XmlString)get_store().find_element_user(URL$4, 0);
159             return target;
160         }
167     }

168     /**
169      * Sets the "URL" element
170     */
180     public void setURL(java.lang.String url)
190     {
191         synchronized (monitor())
192         {
193             check_orphaned();
194             org.apache.xmlbeans.SimpleValue target = null;
195             target = (org.apache.xmlbeans.SimpleValue)get_store().find_element_user(URL$4, 0);
196             if (target == null)
197             {
198                 target = (org.apache.xmlbeans.SimpleValue)get_store().add_element_user(URL$4);
199             }
200             target.setStringValue(url);
201         }
202     }

203     /**
204      * Sets (as xml) the "URL" element
205      */
206     public void xsetURL(org.apache.xmlbeans.XmlString url)
207     {
208         synchronized (monitor())
209         {
210             check_orphaned();
211             org.apache.xmlbeans.XmlString target = null;
212             target = (org.apache.xmlbeans.XmlString)get_store().find_element_user(URL$4, 0);
213             if (target == null)
214             {
215                 target = (org.apache.xmlbeans.XmlString)get_store().add_element_user(URL$4);
216             }
217             target.set(url);
218         }
219     }

220     /**
```

```
221      * Gets the "MessageContent" element
222     */
223     public java.lang.String getMessageContent()
224     {
225         synchronized (monitor())
226         {
227             check_orphaned();
228             org.apache.xmlbeans.SimpleValue target = null;
229             target = (org.apache.xmlbeans.SimpleValue)get_store().find_element_user(MESSAGECONTENT$6, 0);
230             if (target == null)
231             {
232                 return null;
233             }
234             return target.getStringValue();
235         }
236     }

237     /**
238      * Gets (as xml) the "MessageContent" element
239      */
240     public org.apache.xmlbeans.XmlString xgetMessageContent()
241     {
242         synchronized (monitor())
243         {
244             check_orphaned();
245             org.apache.xmlbeans.XmlString target = null;
246             target = (org.apache.xmlbeans.XmlString)get_store().find_element_user(MESSAGECONTENT$6, 0);
247             return target;
248         }
249     }

250     /**
251      * Sets the "MessageContent" element
252      */
253     public void setMessageContent(java.lang.String messageContent)
254     {
255         synchronized (monitor())
256         {
257             check_orphaned();
258             org.apache.xmlbeans.SimpleValue target = null;
259             target = (org.apache.xmlbeans.SimpleValue)get_store().find_element_user(MESSAGECONTENT$6, 0);
260             if (target == null)
261             {
262                 target = (org.apache.xmlbeans.SimpleValue)get_store().add_element_user(MESSAGECONTENT$6);
263             }
264             target.setStringValue(messageContent);
265         }
266     }

267     /**
268      * Sets (as xml) the "MessageContent" element
269      */
270     public void xsetMessageContent(org.apache.xmlbeans.XmlString messageContent)
271     {
272         synchronized (monitor())
273         {
274             check_orphaned();
275             org.apache.xmlbeans.XmlString target = null;
276             target = (org.apache.xmlbeans.XmlString)get_store().find_element_user(MESSAGECONTENT$6, 0);
277             if (target == null)
278             {
279                 target = (org.apache.xmlbeans.XmlString)get_store().add_element_user(MESSAGECONTENT$6);
280             }
281             target.set(messageContent);
282         }
283     }
```

```
284     /**
285      * Gets the "TargetNameSpace" element
286      */
287     public java.lang.String getTargetNameSpace()
288     {
289         synchronized (monitor())
290         {
291             check_orphaned();
292             org.apache.xmlbeans.SimpleValue target = null;
293             target = (org.apache.xmlbeans.SimpleValue)get_store().find_element_user(TARGETNAMESPACE$8, 0);
294             if (target == null)
295             {
296                 return null;
297             }
298             return target.getStringValue();
299         }
300     }

301     /**
302      * Gets (as xml) the "TargetNameSpace" element
303      */
304     public org.apache.xmlbeans.XmlString xgetTargetNameSpace()
305     {
306         synchronized (monitor())
307         {
308             check_orphaned();
309             org.apache.xmlbeans.XmlString target = null;
310             target = (org.apache.xmlbeans.XmlString)get_store().find_element_user(TARGETNAMESPACE$8, 0);
311             return target;
312         }
313     }

314     /**
315      * Sets the "TargetNameSpace" element
316      */
317     public void setTargetNameSpace(java.lang.String targetNameSpace)
318     {
319         synchronized (monitor())
320         {
321             check_orphaned();
322             org.apache.xmlbeans.SimpleValue target = null;
323             target = (org.apache.xmlbeans.SimpleValue)get_store().find_element_user(TARGETNAMESPACE$8, 0);
324             if (target == null)
325             {
326                 target = (org.apache.xmlbeans.SimpleValue)get_store().add_element_user(TARGETNAMESPACE$8);
327             }
328             target.setStringValue(targetNameSpace);
329         }
330     }

331     /**
332      * Sets (as xml) the "TargetNameSpace" element
333      */
334     public void xsetTargetNameSpace(org.apache.xmlbeans.XmlString targetNameSpace)
335     {
336         synchronized (monitor())
337         {
338             check_orphaned();
339             org.apache.xmlbeans.XmlString target = null;
340             target = (org.apache.xmlbeans.XmlString)get_store().find_element_user(TARGETNAMESPACE$8, 0);
341             if (target == null)
342             {
343                 target = (org.apache.xmlbeans.XmlString)get_store().add_element_user(TARGETNAMESPACE$8);
344             }
345           target.set(targetNameSpace);
346         }
347     }
```

```
348     /**
349      * Gets the "SeviceName" element
350      */
351     public java.lang.String getSeviceName()
352     {
353         synchronized (monitor())
354         {
355             check_orphaned();
356             org.apache.xmlbeans.SimpleValue target = null;
357             target = (org.apache.xmlbeans.SimpleValue)get_store().find_element_user(SEVICENAME$10, 0);
358             if (target == null)
359             {
360                 return null;
361             }
362             return target.getStringValue();
        }
363     }

364     /**
365      * Gets (as xml) the "SeviceName" element
366      */
367     public org.apache.xmlbeans.XmlString xgetSeviceName()
368     {
369         synchronized (monitor())
370         {
371             check_orphaned();
372             org.apache.xmlbeans.XmlString target = null;
373             target = (org.apache.xmlbeans.XmlString)get_store().find_element_user(SEVICENAME$10, 0);
374             return target;
375         }
376     }

377     /**
378      * Sets the "SeviceName" element
379      */
380     public void setSeviceName(java.lang.String seviceName)
381     {
382         synchronized (monitor())
383         {
384             check_orphaned();
385             org.apache.xmlbeans.SimpleValue target = null;
386             target = (org.apache.xmlbeans.SimpleValue)get_store().find_element_user(SEVICENAME$10, 0);
387             if (target == null)
388             {
389                 target = (org.apache.xmlbeans.SimpleValue)get_store().add_element_user(SEVICENAME$10);
390             }
391             target.setStringValue(seviceName);
392         }
393     }

394     /**
395      * Sets (as xml) the "SeviceName" element
396      */
397     public void xsetSeviceName(org.apache.xmlbeans.XmlString seviceName)
398     {
399         synchronized (monitor())
400         {
401             check_orphaned();
402             org.apache.xmlbeans.XmlString target = null;
403             target = (org.apache.xmlbeans.XmlString)get_store().find_element_user(SEVICENAME$10, 0);
404             if (target == null)
405             {
406                 target = (org.apache.xmlbeans.XmlString)get_store().add_element_user(SEVICENAME$10);
407             }
408             target.set(seviceName);
409         }
```

```
410     }
```

List 4.3.3 show us the overridden methods, which are in List 4.3.2 described.

In interface ClientTestDescriptorDocument we see an important class:

public static final class Factory {…},

In this class there are two important methods:

```
1     public static org.myXMLSchema.ClientTestDescriptorDocument newInstance(org.apache.xmlbeans.XmlOptions
2     options) {
3           return (org.myXMLSchema.ClientTestDescriptorDocument)
4     org.apache.xmlbeans.XmlBeans.getContextTypeLoader().newInstance( type, options ); }
```

newINstance () is used to create instances of this type

```
/** @param file the file from which to load an xml document */

1      public static org.myXMLSchema.ClientTestDescriptorDocument parse(java.io.File file) throws
2           org.apache.xmlbeans.XmlException, java.io.IOException {
3        return (org.myXMLSchema.ClientTestDescriptorDocument)
4     org.apache.xmlbeans.XmlBeans.getContextTypeLoader().parse( file, type, null ); }
```

parse () is used to parse the actual XML instance document.

notice that: there is many methods named parse, but the parameter within it difference. We use the parse with parameter : java.io.File because we need load an XML instance.

Now we use RAD to simply generate a XML instance from schema:

List 4.3.4

```
1     <?xml version="1.0" encoding="UTF-8"?>
2       <tns:ClientTestDescriptor xmlns:tns="http://org/myXMLSchema"
3        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4          xsi:schemaLocation="http://org/myXMLSchema testXMLSchema2.xsd ">
5        <tns:NumberOfParallelInvoke>0</tns:NumberOfParallelInvoke>
6          <tns:MessageSize>0</tns:MessageSize>
7          <tns:URL>tns:URL</tns:URL>
8           <tns:MessageContent>tns:MessageContent</tns:MessageContent>
9            <tns:TargetNameSpace>tns:TargetNameSpace</tns:TargetNameSpace>
10          <tns:SeviceName>tns:SeviceName</tns:SeviceName>
11           </tns:ClientTestDescriptor>
```

List 4.3.4 describes the XML file, which is generated from the schema in list 4.3.1. This XML file includes the information, which we need to invoke the Web service.

Now we need to bind an XML instance to the above classes. Here is the code from ClientTest.java that use the generated interfaces to handle an XML file (We choose core code):

```
1    import org.apache.xmlbeans.XmlException;
2    import org.myXMLSchema.*;
3    import org.myXMLSchema.ClientTestDescriptorDocument.ClientTestDescriptor;

4    //import com.TestClient.ClientTestScriptDocumentHandler;

5    //import com.TestClient.ClientTestScriptDocumentHandler;
6    import org.myXMLSchema.ClientTestDescriptorDocument;
7    import java.lang.Exception;
8    import java.math.BigInteger;
9    import javax.xml.namespace.QName;
10   import java.io.StringReader;
11   import java.net.URL;
12   import java.util.Scanner;

13   import javax.xml.soap.MessageFactory;
14   import javax.xml.soap.SOAPException;
15   import javax.xml.soap.SOAPMessage;
16   import javax.xml.transform.Source;
17   import javax.xml.transform.stream.StreamSource;
18   import javax.xml.ws.Dispatch;
19   import javax.xml.ws.Service;
```

Notice that: we must add xmlSchema.jar and xmlpublic.jar into JRE library, or we can not use following package:

org.apache.xmlbeans.XmlEccetion and,
org.myxmlschema.ClientTestDescriptorDocument.ClientTestDescriptor

and there will be many failures in the from schema generated interface.

Here is the code for loading and parsing XML file:

Load:

```
1    public static void dosomething(){
2            System.out.println("please type the Filepath!");
3        }
4        static Scanner scanner=new Scanner(System.in);
5    static String Filepath=scanner.next();  //get filepath
```

Parse:

```
1    File inputXMLFile = new File(Filepath);
2                ClientTestDescriptorDocument  clientTestDoc =
3    ClientTestDescriptorDocument.Factory.parse(inputXMLFile);
4            ClientTestDescriptor ctd = clientTestDoc.getClientTestDescriptor();
```

Here is the code for JAX – WS client:

```
1    // create a client and invoke web service
2                    Service service = Service.create(new URL(ctd.getURL()), new QName(ctd.getTargetNameSpace(),
3    ctd.getSeviceName()));
4                    System.out.println("Service created");

5                    Dispatch<SOAPMessage> dispatchMsg;
6                dispatchMsg = service.createDispatch(new QName(ctd.getTargetNameSpace(),ctd.getSeviceName()),
7                        SOAPMessage.class,javax.xml.ws.Service.Mode.MESSAGE);
```

So above is the core of this Studienarbeit. Using method: dosomething we input the path of the XML file that contains the information invoke the Web service, and then parse this XML file and get the information we want, at last we use these information to invoke the web service and control it.

Using XMLBeans the developer can more simply work with the XML based project. When we emulate some process, for a instance, the biological systems, the information is transmit by XML file, then we can read the information and handle it and send it to next Node.

5 References:

1. http://www-01.ibm.com/software/awdtools/developer/application/features/index.html?S_CMP=wspace

2. http://java.sun.com/webservices/docs/2.0/tutorial/doc/JAXWS3.html

3. http://www.ibm.com/developerworks/webservices/library/ws-tip-jaxwsrpc.html

4. http://java.sun.com/developer/technicalArticles/J2SE/jax_ws_2/

5. http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp?topic=/com.ibm.websphere.express.doc/info/exp/ae/twbs_jaxwsdynclient.html

6. http://webservice.bea.com/examplesWebApp/docs/core/index.html

7. http://xmlbeans.apache.org/docs/2.0.0/guide/tools.html

8. http://en.wikipedia.org/wiki/XMLBeans

9. http://www.ibm.com/developerworks/xml/library/x-beans1/

10. http://java.boot.by/scdjws5-guide/ch04.html (http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp?topic=/com.ibm.websphere.express.doc/info/exp/ae/cwbs_jaxws.html)

11. www.metadesignsolutions.com/node/28

12. www.w3.org/TR/ws/arch#gengag

13. www.developer.com/services/aricle.php/1602051/WSDL-Essential.htm

14. http://www.w3schools.com/SOAP