Laboratory Manual

For

Service Oriented Computing

M. Tech. (IT)

SEM II

June 2011

Faculty of Technology
Dharmsinh Desai University
Nadiad.
www.ddu.ac.in

Table: List of Experiment.

EXPERIMENT-01

Aim: Validate XML instance document using XML schema.

Design XML Schema and XML instance document Tools/ Apparatus: GUI-IDE Tool NetBeans 6.0

Procedure:

1) Design a schema for student list. A student has information such as name, semester, roll no, email-ids, phone-nos, etc.
2) Write an XML instance document for the designed schema and validate this instance document against the schema.

MyXML.xml:

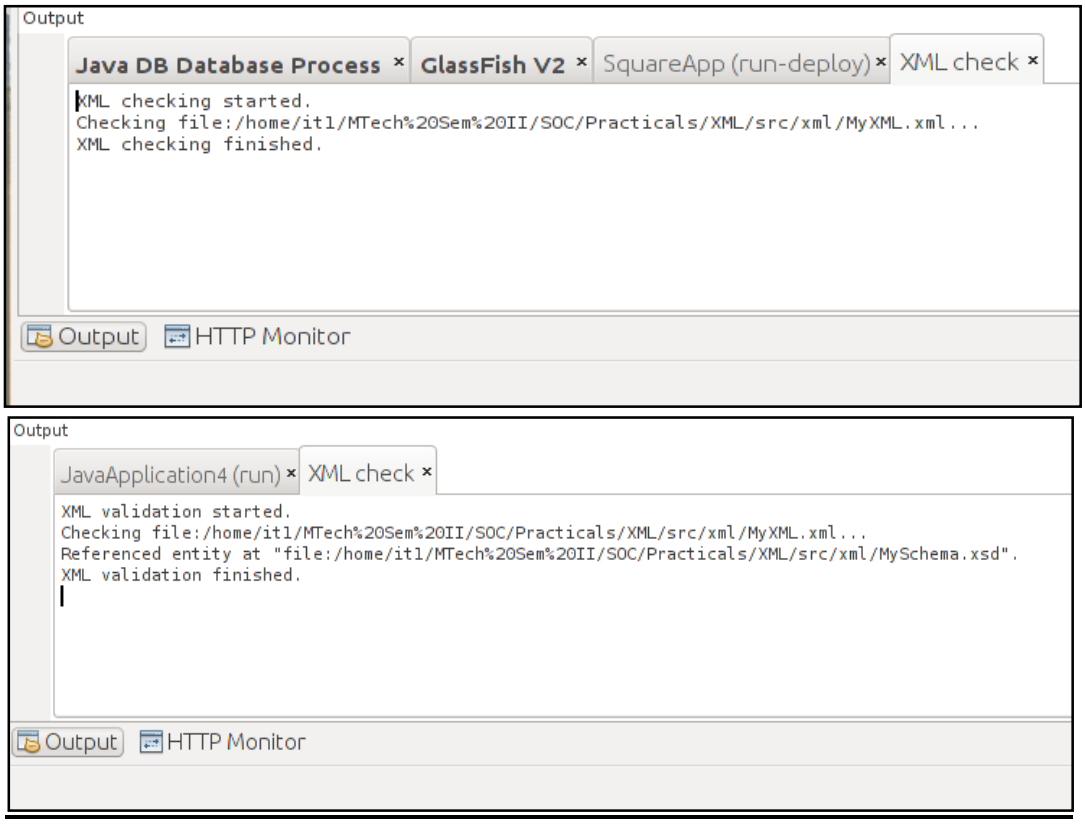
```
<?xml version="1.0" encoding="UTF-8"?>

 <xs:Device1 xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
xmlns:xs='http://xml.netbeans.org/schema/MySchema'
xsi:schemaLocation ='http://xml.netbeans.org/schema/MySchema  MySchema.xsd'>

    <xs:Mobile>
       <xs:Name> Aqua Power HD </xs:Name>
       <xs:Color>  White   </xs:Color>
       <xs:Company> Intex </xs:Company>
    </xs:Mobile>

    <xs:Dell>
       <xs:Name> Aqua Power HD </xs:Name>
       <xs:Company> Intex </xs:Company>
        <xs:Processor>  ISeven   </xs:Processor>
    </xs:Dell>

 </xs:Device1>
```

MySchema.xsd

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

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://xml.netbeans.org/schema/MySchema"
        xmlns:tns="http://xml.netbeans.org/schema/MySchema"
        elementFormDefault="qualified">

   <xsd:complexType name="MobileType">
      <xsd:sequence>
         <xsd:element name="Name"/>
```

```
            <xsd:element name="Color"/>
            <xsd:element name="Company"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="Laptop">
        <xsd:sequence>
            <xsd:element name="Name"/>
            <xsd:element name="Company"/>
            <xsd:element name="Processor"/>
        </xsd:sequence>
    </xsd:complexType>

    <xsd:element name="Device1">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="Mobile" type="tns:MobileType"></xsd:element>
                <xsd:element name="Dell" type="tns:Laptop"></xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>

</xsd:schema>
```

Output:



EXPERIMENT-02

Aim: Parse XML instance document using DOM parser.

Design XML document Tools/ Apparatus: GUI-IDE Tool NetBeans 6.0

Procedure:

1) Design an XML file.
2) Write a java program(**DOM Parser**) that reads XML file and parse it to print its contents.

XMLDoc.xml

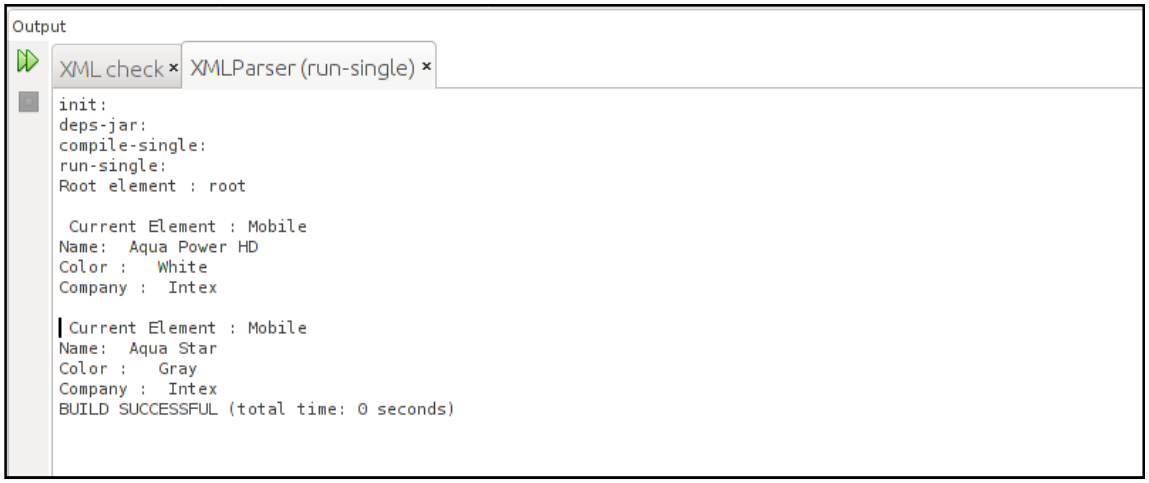
```
<?xml version="1.0" encoding="UTF-8"?>

<root>
   <Mobile>
       <Name> Aqua Power HD </Name>
       <Color>  White   </Color>
       <Company> Intex </Company>
   </Mobile>

   <Mobile>
       <Name> Aqua Star </Name>
       <Color>  Gray   </Color>
       <Company> Intex </Company>
   </Mobile>
</root>
```

Main.java

```
package xmlparser;

import java.io.File;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.DocumentBuilder;
import org.w3c.dom.Document;
import org.w3c.dom.NodeList;
import org.w3c.dom.Node;
import org.w3c.dom.Element;

public class Main
{

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

```
      try
      {
       File input = new File("/home/it1/MTech Sem
II/SOC/Practicals/XMLParser/src/XMLDoc.xml");
       DocumentBuilderFactory dbFactory
         = DocumentBuilderFactory.newInstance();
       DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
       Document doc = dBuilder.parse(input);
       doc.getDocumentElement().normalize();
       System.out.println("Root element : " + doc.getDocumentElement().getNodeName());
       NodeList nList = doc.getElementsByTagName("Mobile");

       for (int temp = 0; temp < nList.getLength(); temp++)
       {
         Node nNode = nList.item(temp);
         System.out.println("\n Current Element : "+ nNode.getNodeName());
         if (nNode.getNodeType() == Node.ELEMENT_NODE)
         {
           Element eElement = (Element) nNode;
           System.out.println("Name: "+
eElement.getElementsByTagName("Name").item(0).getTextContent());
           System.out.println("Color : " +
eElement.getElementsByTagName("Color").item(0).getTextContent());
           System.out.println("Company : " +
eElement.getElementsByTagName("Company").item(0).getTextContent());
         }
       }
     }
      catch (Exception e)
      {
       e.printStackTrace();
     }   }  }
```

Output:



EXPERIMENT-03

Aim: Create a Web Service using Code First Approach and deploy it in Application Server.

Procedure:

Steps to Create Web Service

1. Create New Java Web Application from File Menu.
2. In Created Java Web Application ,
   Create new Web Service Give some appropriate name to that web service.
3. Now Go to newly created "Arithmatic".java file. Go to design console. And Add operations which you want to provide in web service.
4. Design Console will prepare structure of all the operation you have added in web service. Then write appropriate logic in all the method you have created.
5. Once web service is created then right click on project and deploy web service.
6. Once Web service is successfully deployed.
   Go to your web service and right click on it and then Test Web Service. So Browser will open and you can test your web service.
7. When browser opens you can click on WSDL file to see it content of it for arithmatic web service is shown here.

Steps to create Web Service Client:

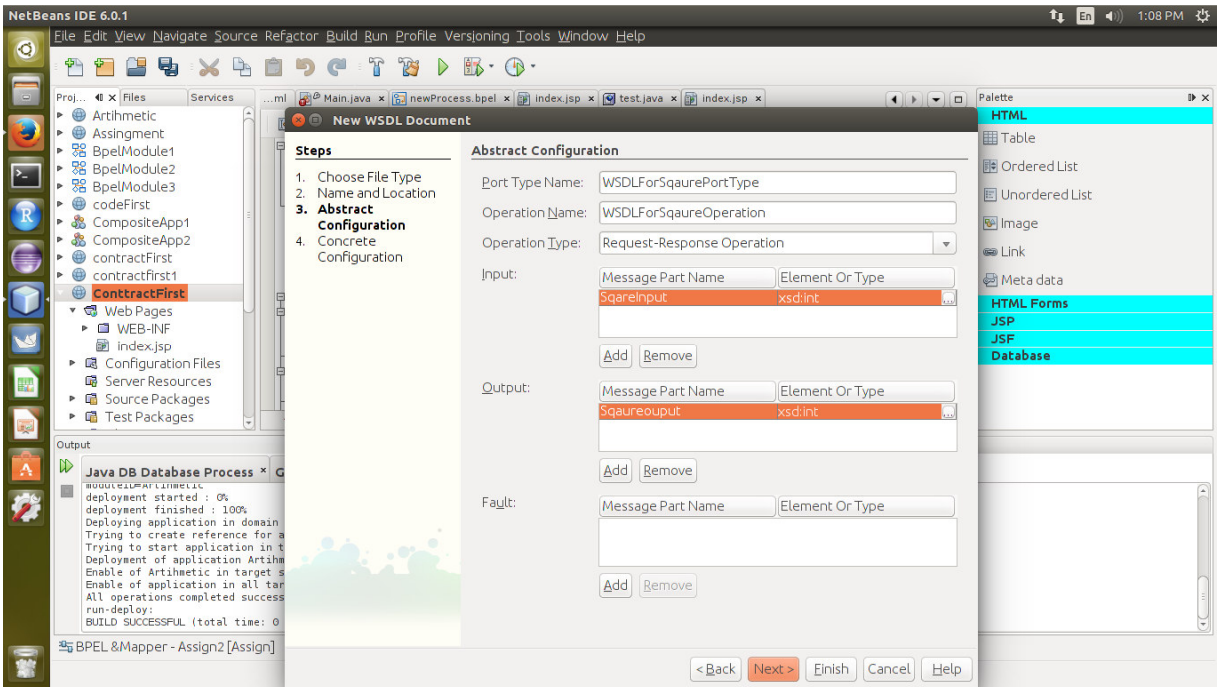
1 Create a new project of type Java application. Give Arithmatic Client.
2 Right click on project folder and select New Web service client.
3 A dialog box will appear asking location of WSDL and client. For WSDL specify
4 http://localhost:8080/Arithmetic/ArithmeticService Service?WSDL and for client specify websvcclient in package option. Make sure Style is JAX-WS.
5 Right click in source code of Main class. Select option "Web service client resource" Call web
6 service operation. A new dialog box will appear asking for selecting name of operation. Select "add" operation.

EXPERIMENT-04

Aim: Create a Web Service using Contract First Approach and deploy it in Application Server.

Procedure:

1. Create New Java Web Application from File Menu.
2. In Created Java Web Application ,
   Create new WSDL document. Create concrete WSDL document.



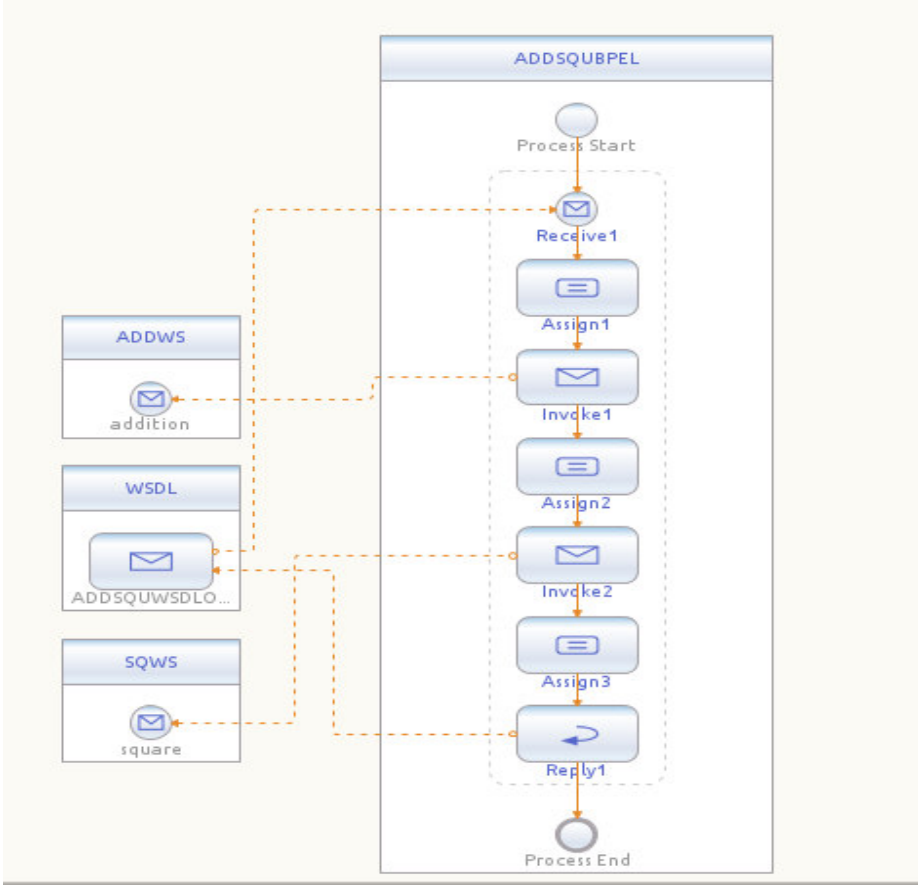
3. Now Create Web service from WSDL Document. Select WSDL file you have created while creating web service.
4. Once web service is created then right click on project and deploy web service.
5. Once Web service is successfully deployed.
   Go to your web service and right click on it and then Test Web Service. So Browser will open and you can test your web service.

EXPERIMENT-05

Aim: Create BPEL process composing two web service.

Procedure:

1. **Create two web Services 1. SquareWebService 2. SumWebService**
   As discusses in practical 2.
2. **Once Web service are created go to New Project -> SOA -> BPEL Module.**
   Create BPEL Module "MyBPELModule".
3. **Now in that Bpel module right click on "Process Files" and create new WSDL document for client.**
   Here in our practical client will send two input variable to BPEL module and BPEL module first send those two variables to First Sum Web Service and Answer of That sum Web Service given to Square Web Service. And Final Answer from square Web service given to Client.
4. **Create Following BPEL Module by Creating Partnerlinks.**



5. **Do correct mapping in every assign activity.**

6. **Once BPEL module is created then check and validate it by right clicking on it.**
7. **Then clean and Build your BPEL module so it's jar file will be created.**
8. **Then Create composite application MyCompositeBPEL**
9. **In JBI module add myBPELModule.jar.**
10. **Then create Test cases by selecting your client WSDL file and give some data shown in fig.**
11. **Clean and Build Composite module.**
12. **Run Test cases.**

Input

```
<soapenv:Envelope xsi:schemaLocation="http://schemas.xmlsoap.org/soap/envelope/
http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:myh="http://myHHMWSDL">
 <soapenv:Body>
  <myh:myHHMWSDLOperation>
   <input1>4</input1>
   <input2>6</input2>
  </myh:myHHMWSDLOperation>
 </soapenv:Body>
</soapenv:Envelope>
```

Output

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://schemas.xmlsoap.org/soap/envelope/
http://schemas.xmlsoap.org/soap/envelope/">
 <SOAP-ENV:Body>
  <m:myHHMWSDLOperationResponse xmlns:m="http://myHHMWSDL">
   <result xmlns="">100.0</result>
  </m:myHHMWSDLOperationResponse>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

EXPERIMENT-06

Aim: Create a OWL File Using Protege Editor

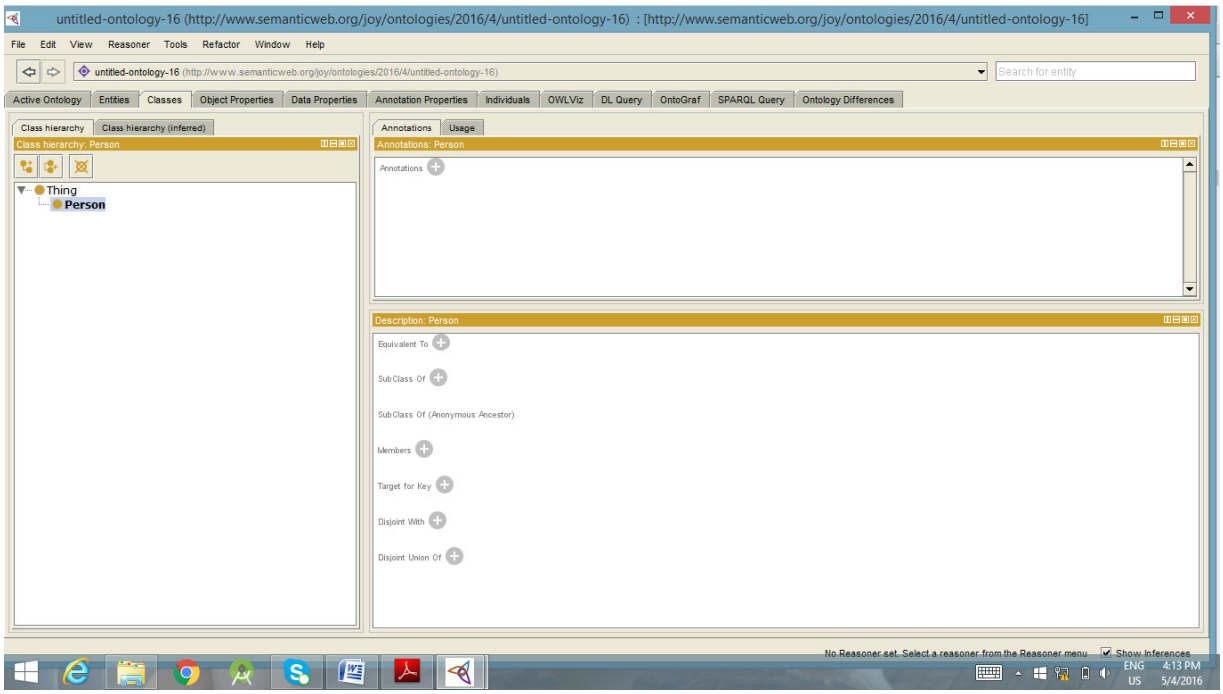
Procedure:

1. **Start protégé for OWL.**
2. **Create classes under Thing class (add classes/sub-classes).**
3. **Add property (object property)**
4. **Add data property/characteristics and its description.**

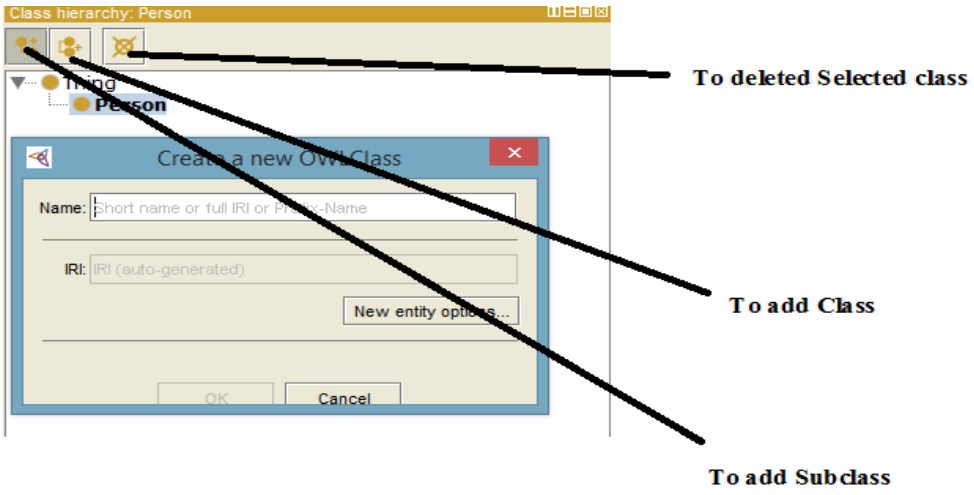
Owl Ontologies:

- Ontologies are used to capture knowledge about some domain of interest. An ontology describes the concepts in the domain and also the relationships that hold between those concepts. Different ontology languages provide different facilities. The most recent development in standard ontology languages is OWL from the World Wide Web Consortium.
- There are three type of OWL:
  1. Owl Lite
  2. Owl DL
  3. Owl Full
- Components of Owl are:
  1. Individual
  2. Classes
  3. Properties
- Using protégé Owl we can describe OWL and it provide rich sets of operators like and, or and negation.
- Furthermore, the logical model allows the use of a reasoner which can check whether or not all of the statements and definitions in the ontology are mutually consistent and can also recognise which concepts fit under which definitions. The reasoner can therefore help to maintain the hierarchy correctly.

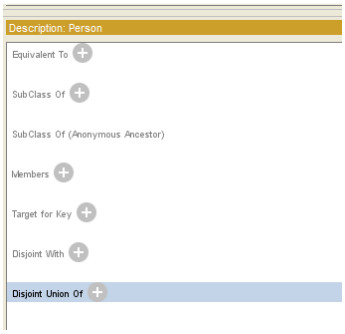
Protege:



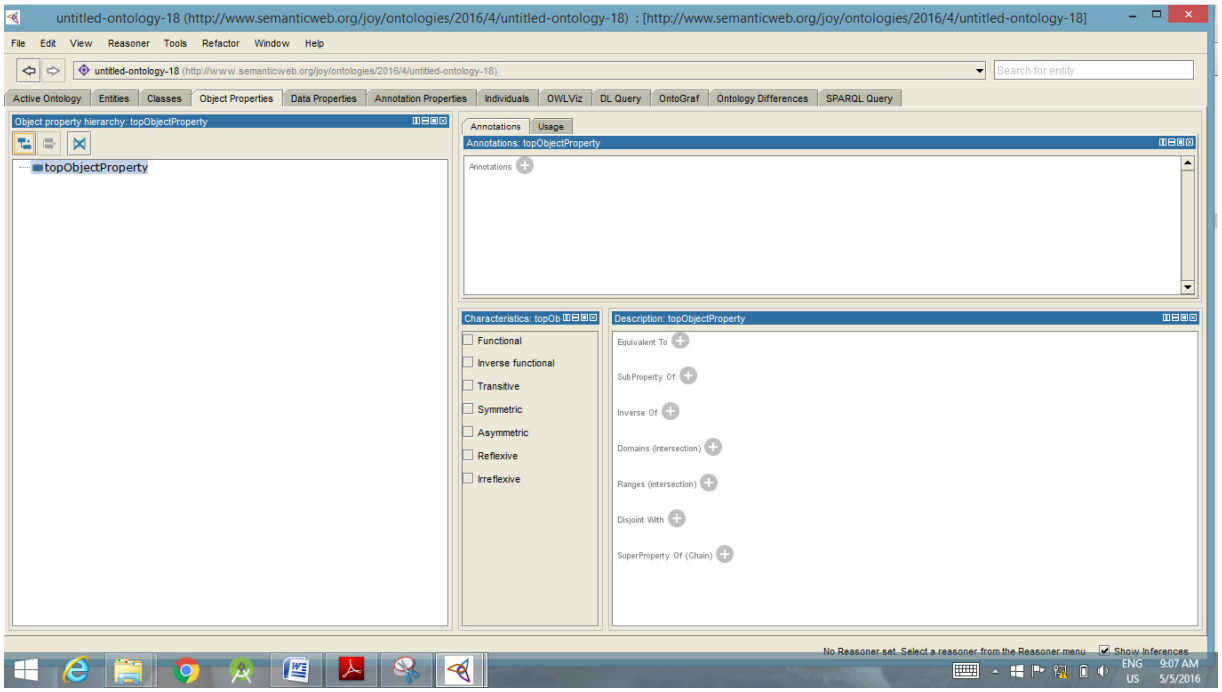
To add class, subclass and to remove class subclass:

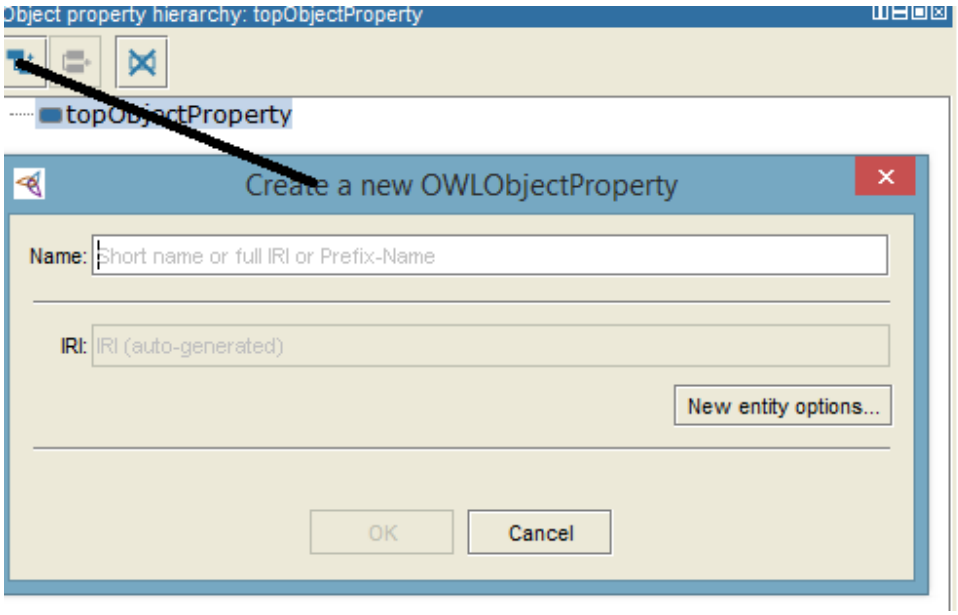


To Add Description about class:

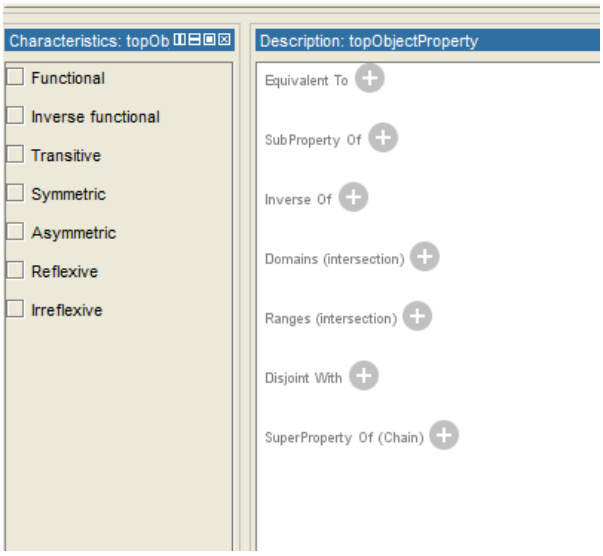


To add object Properties:

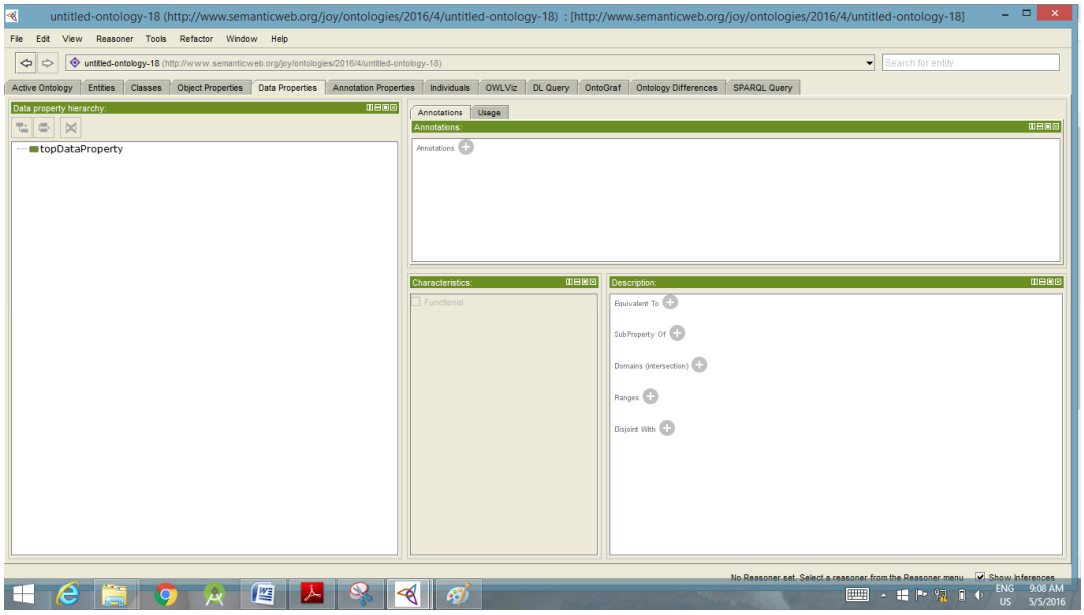


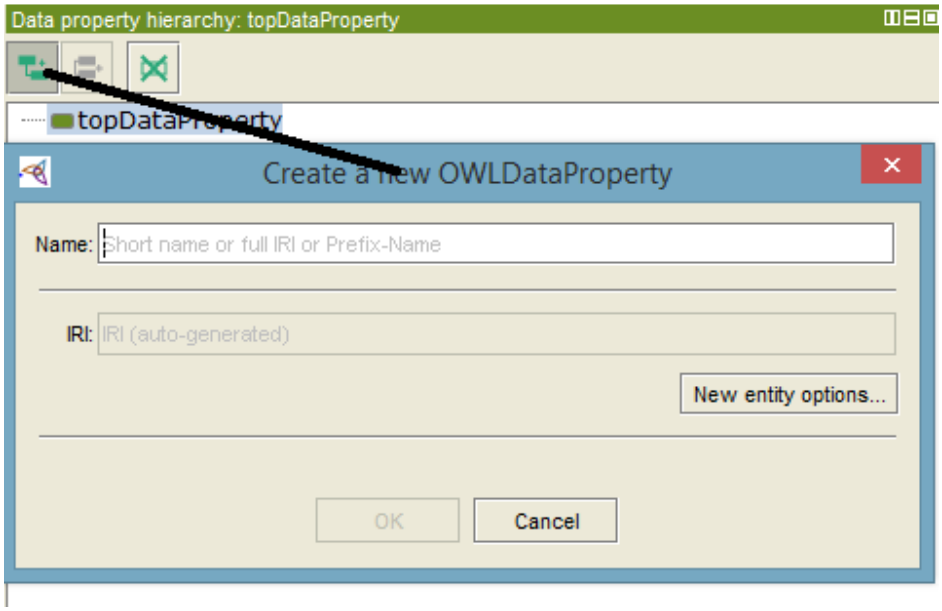


To add Object Properties Characteristics:

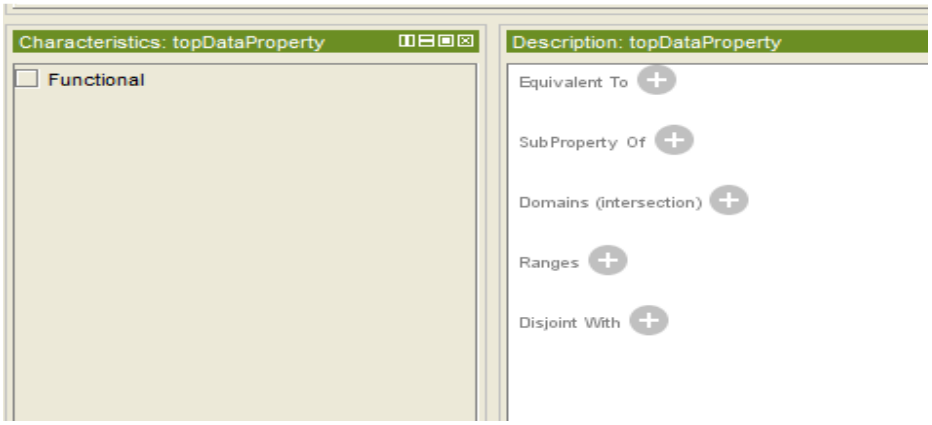


To Add Data properties:





To add Data Properties Characteristic and Description:



LABWORK BEYOND CURRICULA

Experiment 1

Aim: Implementing Publish/Subscribe Paradigm using Web Services, ESB and JMS

Tools/ Apparatus: Web service, BPEL Runtime Environment: GlassFish Server, GUI-IDE Tool: NetBeans 6.0, JMS, ESB: WSO2

Procedure:

1. Create user interface in .NET environment – for publisher and for subscriber

2. Create two web services - one for subscriber to register themselves in the database and another for the publisher to send message through JMS to Topic for inserting data into to data base.

3. Create two proxies for the two web-services by taking endpoints as the web services. These would provide the functionalities to logs, mediate messages route or transform them as per the settings of ESB (WSO2).

4. Create java message driven bean (JMS) for directing message from the web service to the message driven bean. The message driven bean would in turn through its onMessage() method would take info from the subscribers table and send mail accordingly

Experiment 2

Aim: Implementing Stateful grid services using Globus WS-Core-4.0.3

Tools/ Apparatus: Web service, Globus WS-Core-4.0.3, JNDI, Apache Ant

Procedure:

1. Define the web service's interface. This is done with WSDL.

2. Implement the web service in Java.

3. Define the deployment parameters by using WSDD and JNDI.

4. Compile everything and generate a GAR file using Ant.

5. Deploy service using Globus WS-Core-4.0.3 GT4 tool