HAAGA-HELIA
ammattikorkeakoulu

Implementing reliable Web services

Otto Koskipää

Thesis work

Business Information Technology

22.5.2012

Abstract

16.5.2012

DP in Business Information Technology

Author Otto Koskipää	**Group or year of entry** 2007
The title of thesis **Implementing reliable Web services**	**Number of pages and appendices** 62 + 1
Supervisor Martti Laiho	

Web services are a common and standard way to implement communication between information systems and provide documented interfaces. The Web services are usually using SOAP because it is a widely-spread, well-documented and used standard.

SOAP standard defines a message structure, an envelope, that is sent over internet using HTTP and contains XML data. An important part of the SOAP structure is the exception mechanism that returns a Fault element in the response. The SOAP Fault is a standard way of returning errors from a service. In this thesis work test cases with exceptions are investigated by creating a Web service and a counterpart (client) for it. A similar Web service is created with different programming languages and technologies. Tests are done by creating a test client program using the Web service description (WSDL).

The research was conducted between May 2011 and May 2012 using the DBTechNet Debian virtual laboratory.

The results in the test cases show that different vendors and contributors have implemented the standards pretty well, but there are still some problems with some products. The problems happen when trying to understand the service description (WSDL). One way to avoid this problem is to design the Web service interface carefully and how it is described to outside.

Keywords
Web service, Exception, SOAP, Fault

Tiivistelmä

16.5.2012

Tietojenkäsittelyn koulutusohjelma

Tekijä Otto Koskipää	**Ryhmätunnus tai aloitusvuosi** 2007
Raportin nimi **Implementing reliable Web services**	**Sivu- ja liitesivumäärä** 62 + 1
Opettajat tai ohjaajat Martti Laiho	

Web service -palvelut ovat yleinen ja standardisoitu tapa toteuttaa tietojärjestelmien välisiä yhteyksiä ja toiminnallisuuksia sekä tarjota hyvin kuvatut sovellusrajapinnat. Eniten käytetty ja tunnettu web service -muoto on SOAP-yhteyskäytäntö, joka on hyvin dokumentoitu ja laajalti käytössä oleva tapa.

SOAP-standardi määrittelee sanomalle tietynlaisen rakenteen, kehyksen, joka liikkuu HTTP-yhteyskäytännön avulla, XML-muotoisena ja internetin välityksellä. Yksi tärkeä osa SOAP-kehyksen määritelmää on poikkeus- ja virheenkäsittelymekanismi, joka virheen tapahtuessa palauttaa Fault-elementin vastaussanomassa. SOAP Fault -elementti on standardi tapa palauttaa tietoa virheistä, jotka tapahtuvat palvelussa. Tässä työssä tutkitaan erilaisia testitapauksia ja miten ne palauttavat virheilmoituksia web service -palvelusta kutsuvalle asiakasohjelmalle. Ensin tehdään web service -palvelu ja sen jälkeen palvelua käyttävä testiohjelma. Samanlainen palvelu toteutetaan eri arkkitehtuureihin kuuluvilla välineillä ja ohjelmointikielillä. Testiohjelma luodaan palvelun ulkoisesta rajanpintakuvauksesta (WSDL).

Tutkimus toteutettiin toukokuun 2011 ja toukokuun 2012 välisenä aikana. Virtuaalilaboratoriona käytettiin DBTechNetin Debian-ympäristöä.

Työhön valittujen testitapausten perusteella voidaan todeta, että eri tuotteiden tekijät ovat toteuttaneet standardia melko hyvin, vaikka joitakin ongelmiakin löytyi. Ongelmat työssä liittyivät siihen miten eri välineet osaavat tulkita toisella välineellä tehdyn web service:n kuvausta (WSDL). Yksi mahdollisuus välttää tällaisia ongelmia on suunnitella palvelun rajapinta huolellisesti ja tarkistaa miten rajapinta näkyy ulospäin.

Asiasanat
www-sovelluspalvelu, poikkeus, virhe, sanoma

Glossary

RESTful web services

Web services without SOAP payload.

SaaS

Software as a Service. This is used in conjunction with cloud-computing, which offers a set of open interfaces to use the application that resides in the Internet.

SOA

Service Oriented Architecture. A software architecture that is based on individual services that are connected together as a working application by rules and well-defined interfaces. SOA takes advantage of existing systems connecting them together.

SOAP

Simple Object Access Protocol.

W3C

World Wide Web consortium.

WSDL

Web services description language. Describes a Web service technically in XML format.

Contents

1 Introduction

There has always been a need to inter-communicate between different computer systems and therefore different kinds of solutions and mechanisms have been created to enable this. Web services is the current industry standard to implement the inter-system communication making use of familiar structures and standards to make this happen. Although the objective is good, there has been several different standards and techniques how to do this. The good thing is that some ideas have been matured to standards that all platform and tool vendors have implemented in their products.

The Word Wide Web (W3C) consortium defines Web services as ”A Web service is a software system designed to support inter-operable 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.” (W3C 2000. Web services architecture.)

In addition to SOAP, there are some other ways to create Web services too. Another popular way is using so-called RESTful services that are basically XML messages sent over HTTP, but without SOAP payload (envelope).

Web services were introduced in early 2000's and there has been a lot of development around this idea since that. Today's examples of a Web service use is SOA (Service Oriented Architecture) and different cloud-computing applications.

SOA is a software architecture, where applications are made of a collection of services. These individual services can be chosen and connected with each other in a loosely-coupled manner. (Hansen 2007, 548.) SOA takes advantage of existing, separate systems and brings them together, working as an integration between the parts.

Currently, the standards development around Web services have focused on issues like security and there are several new ”WS-” recommendations out there waiting to become real standard implementations.

In this thesis work, a set of common connecting system pairs are built and tested to find out a reliable way of communicating between services on different platforms in special and exceptional situations. In optimal circumstances, this might not be an issue, but in any serious business it becomes critical to be sure about the data transfer and results. With Web services there is also the lag, caused by the communication channel (Internet), which differ the solution from basic in-house system.

Web services are based on common request/response communication method. This means you actually get to know the final result of the operation by checking the response. It is also important to be aware how an exceptional event shows in this response. In Web services' case, there are a couple of alternatives, depending on the target area. These methods will be discussed in more detail in the following chapter.

The technologies and tools in this thesis work were chosen based on the free software in the DBTechNet's Debian virtual laboratory learning environment configuration. It has pre-installed Java, IBM Data Studio (based on Eclipse), Oracle database, IBM WAS Community Edition and Mono .NET port for Linux. Many things can be tested with one virtual laboratory. And all the pre-installed software is free to use.

The purpose of these tests is to perform a proof-of-concept with very basic and simple test cases and see how these different tools communicate with each other.

2 Error and exception handling

When two or more systems discuss with each other, there are several different kinds of levels of possible errors or exceptional events. First of all, it might occur that the other system is completely shut-down or a critical service in it does not respond. This means you don't even get to do what you were supposed to, when trying to invoke a desired

Web service of the target system. This is the uppermost level when thinking of possible errors in the process.

Secondly, there might be some security and/or sanity checks when a request arrives at the target Web service. If the service notices that the request is not valid or for some reason not allowed, then some kind of error should be returned at this point too. For example, there might not be a valid contract to use the service or some of the given data seems to be incorrect (checksums etc.).

Finally, there can be the validation checks from the business rules when trying to process the otherwise valid Web service request. For example, there could be missing name or address data in the request that is actually mandatory information. This must also be communicated back to source system in some informative way. There must be a clear and unambiguous message that can be handled by the source system or end-user.

Exceptions can also occur when processing the request data, just like when using any computer system. There might be, for example, some kind of resource lock situation in the system that is preventing the successful result of the whole process. In these cases, it must be carefully planned what kind of error message is returned to service client, so that it clearly tells what happened, what to do (if further actions are required) and not giving out too much information about the application implementation details.

There are at least two ways to implement exceptions and errors coming back to requester. One way is to define and describe special error code and message elements for the service response. These can be left empty when everything was successful. The requester just has to check the contents of these elements to be sure the operation went fine.

In SOAP (Simple Object Access Protocol), there is also another option that can be used for this purpose. There is a special element in the SOAP standard definition called SOAP Fault. In case of error, this element can return enough information to determine what went wrong and where.

2.1 SOAP message structure

The SOAP message is wrapped in an SOAP envelope that has a header and body. If an error happens in the SOAP processing or an exception is raised in the Web service, a Fault element is returned in the response body.

Structure of SOAP request:

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Header/>
   <S:Body>
    ...
   </S:Body>
</S:Envelope>
```

Structure of SOAP response (with optional Fault element in the Body):

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <S:Fault xmlns:ns4="http://www.w3.org/2003/05/soap-envelope">
         <faultcode>...</faultcode>
         <faultstring>...</faultstring>
         <detail>
          ...
         </detail>
      </S:Fault>
   </S:Body>
</S:Envelope>
```

2.2 SOAP Fault element

Basic SOAP Fault has two parts, faultcode and faultstring. Faultcode is used for distinguishing the source of error, meaning Client or Server. Faultstring is a more precise, textual presentation describing the source of problem.

A Client fault happens when the incoming request does not fulfill the WSDL description completely. For example, there might be an element called <customerSearchI-

tems> and for some reason the actual message payload included <customerSearchI-tem> (without the last s letter), this is interpreted as a Client fault. It means the SOAP request was incorrect BEFORE it ever reached the server-side part of the interface. In this case the faultcode would be Client and faultstring describing the more precise reason with the incorrect element name in the message.

Example of Client fault:

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <S:Fault xmlns:ns4="http://www.w3.org/2003/05/soap-envelope">
         <faultcode>S:Client</faultcode>
         <faultstring>Cannot find dispatch method for {http://ws/}getBalance</faultstring>
      </S:Fault>
   </S:Body>
</S:Envelope>
```

If the client request would agree with the WSDL format, the processing would continue to server side. If the Web service is defined to use/throw out a SOAP Fault, all server-side problems will map to Server faultcode and some faultstring accompanied. After an interface is developed and tested between two systems, it is more likely that any SOAP Fault that might come out is a Server fault.

Example of Server fault:

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <S:Fault xmlns:ns4="http://www.w3.org/2003/05/soap-envelope">
         <faultcode>S:Server</faultcode>
         <faultstring>Service error</faultstring>
      </S:Fault>
   </S:Body>
</S:Envelope>
```

2.3 SOAP Fault detail element

If there is a need to provide more detailed output about the exception that happened, the SOAP Fault detail element might become useful. For example, if you need to return detailed information on the specific item and it's value (validation), you can use the detail element to implement a common error message output.

Detail element is not a mandatory part of SOAP Fault element, but it may be utilized if needed. For example, there might be validation checks in the business rules that require the address information to be filled. If this failed, the Web service would return a SOAP Fault with faultcode ”Server” and faultstring ”Application error”. The detail element could have a more precise error message saying ”Address is mandatory field”. This information could be used then on the client side to decide the next actions. If it was a web client with a human entering the information, one could show the detail part of the message on screen and ask for the missing address data. If it was a client system, this definitely would look like a problem in the client system.

Example of the detail part in SOAP response:

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <S:Fault xmlns:ns4="http://www.w3.org/2003/05/soap-envelope">
         <faultcode>S:Server</faultcode>
         <faultstring>Service error</faultstring>
         <detail>
            <ns2:SOAPException xmlns:ns2="http://ws/">
               <code>WS-100001</code>
               <message>The requested service is currently unavailable</message>
            </ns2:SOAPException>
            <ns2:exception class="ws.SOAPException" xmlns:ns2="...">
               <message>Service error</message>
               <ns2:stackTrace>
                  ...
               </ns2:stackTrace>
            </ns2:exception>
         </detail>
      </S:Fault>
   </S:Body>
</S:Envelope>
```

3 Scenarios

To investigate proper exception handling solutions when using Web services, we need to do the same kind of scenario with several different platform implementations. The two most common technical environments today are Java and .NET, so it is fairly reasonable to pick these as the playgrounds we are trying to connect with each other. After all, this is exactly what the web services are designed for: To connect different kinds of systems in a standard and platform-independent way.

In this work, we are using quite simple programs that are sufficient enough to show the effects of exception handling in practice. The focus is only in exception handling and how to provide sufficient error information to the calling client.

It is also good to know that almost all major database vendors have implemented native Web services that can be utilized by using simple wizard-like functionality and offer a Web service interface to different, existing database programs (procedures etc.). We do not investigate these scenarios in this work, but this could be a natural test object in addition to the chosen ones. The future of these native Web services is not totally clear since there has been some news about discontinuing the support in some products.

4 Set up

There are couple of different client/server pairs in this work that we are interested of and that have been chosen as our research objects. The pairs are Java/Java, Java/.NET and .NET/Java. The idea is to create the same simple program published as a Web service in each scenario. Also in one of the pairs (Java/Java), the scenario is a bit more complex to showcase the possible error situations better. There are a lot more possibilities for environment combinations, but these are chosen for this thesis work.

In .NET Web service scenario we emulate possible error situations and return a fault accordingly. We don't use a real database connection in this scenario.

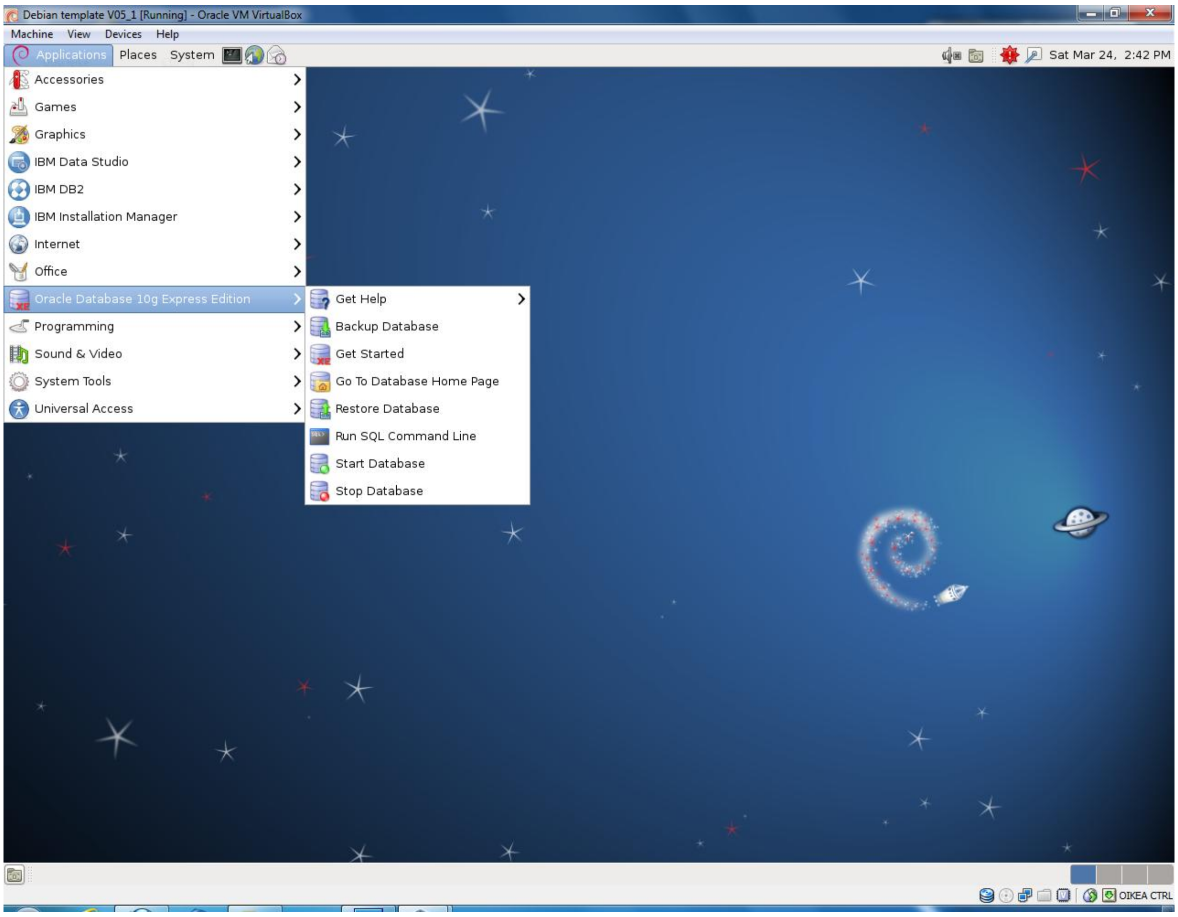
4.1 Java/Java

In the first test case we use Oracle 10g Express Edition database (Oracle XE), IBM Websphere Application Server Community Edition (WASCE) and Web service made with Java. We use the IBM Data Studio to create the Java Web service. All these products are pre-installed in the Debian virtual environment. See the accompanied documentation about the virtual environment for further information about the installed products.

We need to create some test data in the database, set up the database details in the application server, make some additions to configuration and see how to use all these with the tests.

4.1.1 Oracle XE setup and use

1) Start up the Oracle XE database from the menu:
Applications -> Oracle Database 10g Express Edition -> Start Database

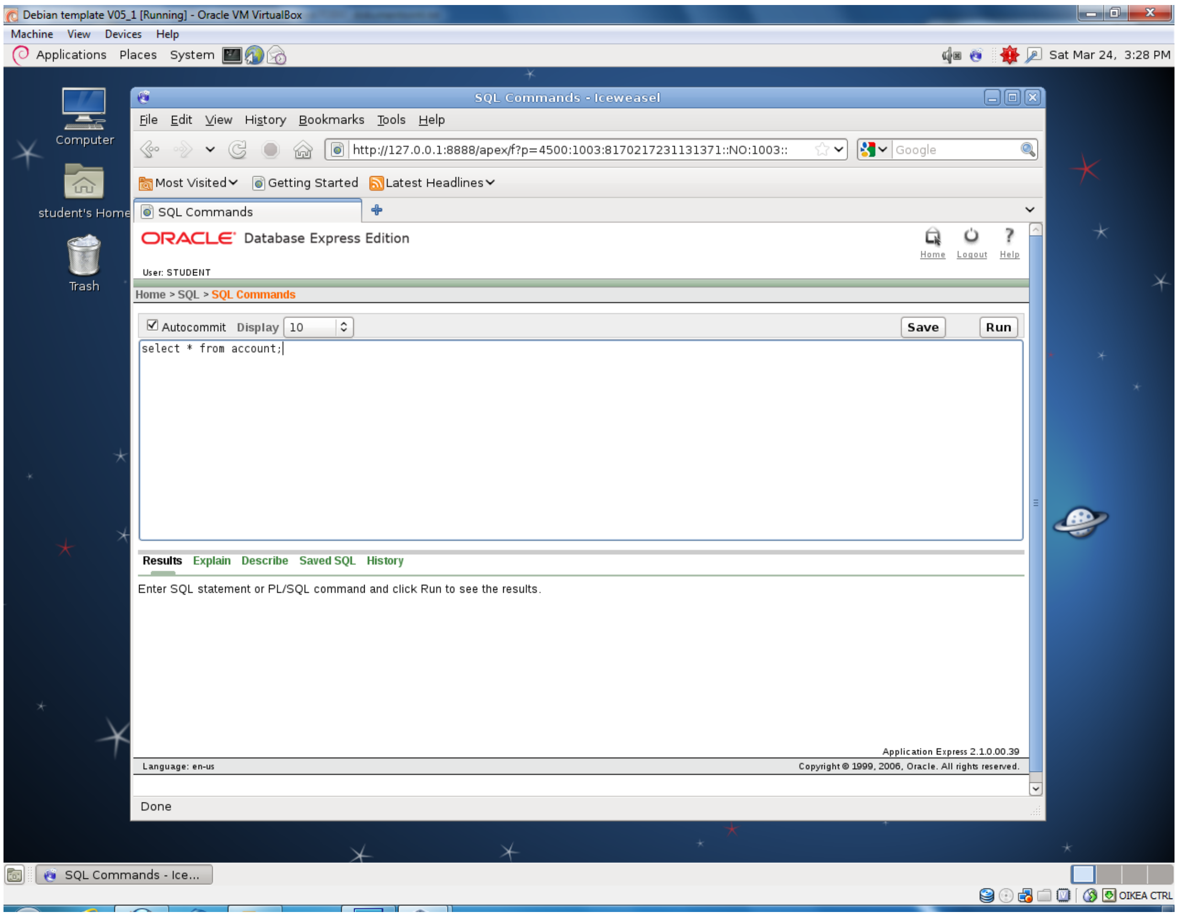


It takes about 30 seconds until the database is fully up and running.
2) Start the db console from the same menu:
Applications -> Oracle Database 10g Express Edition -> Go To Database Homepage

3) Login: user student, password Student1

Now when the home page is shown, we can create a database table and some test data for our tests.

4) Choose SQL -> SQL Commands -> Enter command from menu


SQL Commands - Iceweasel
http://127.0.0.1:8888/apex/f?p=4500:1003:8170217231131371::NO:1003::
ORACLE Database Express Edition
User: STUDENT
Home > SQL > SQL Commands
Autocommit Display 10
Save
Run
Results Explain Describe Saved SQL History

```
select * from account;
```

Enter SQL statement or PL/SQL command and click Run to see the results.

Copy/paste the needed SQL statements below and put them one-by-one to the SQL Commands area. Press Run after each statement. Last statement should be COMMIT. You may really have to enter all these statements one-by-one and press Run, before the next statement.

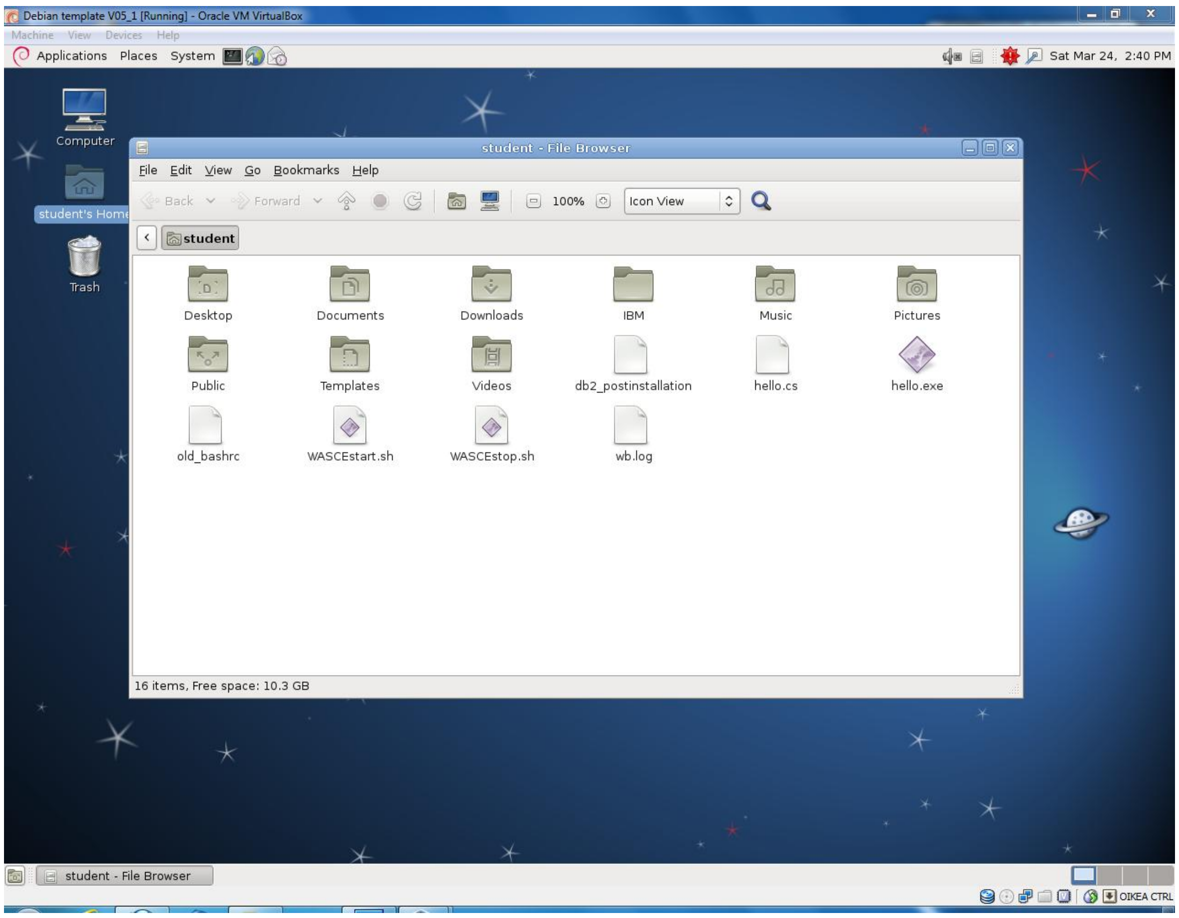
```
CREATE TABLE account (account_no VARCHAR2(60) NOT NULL,
                      balance    NUMBER(15,2) NOT NULL)
/

INSERT INTO account (account_no, balance)
VALUES ('A10001',300.00)
/

COMMIT
/
```

Now we have a database table ACCOUNT and some test data for our tests. You can always shut down the database from menu: Applications -> Oracle Database 10g Express Edition -> Stop Database.

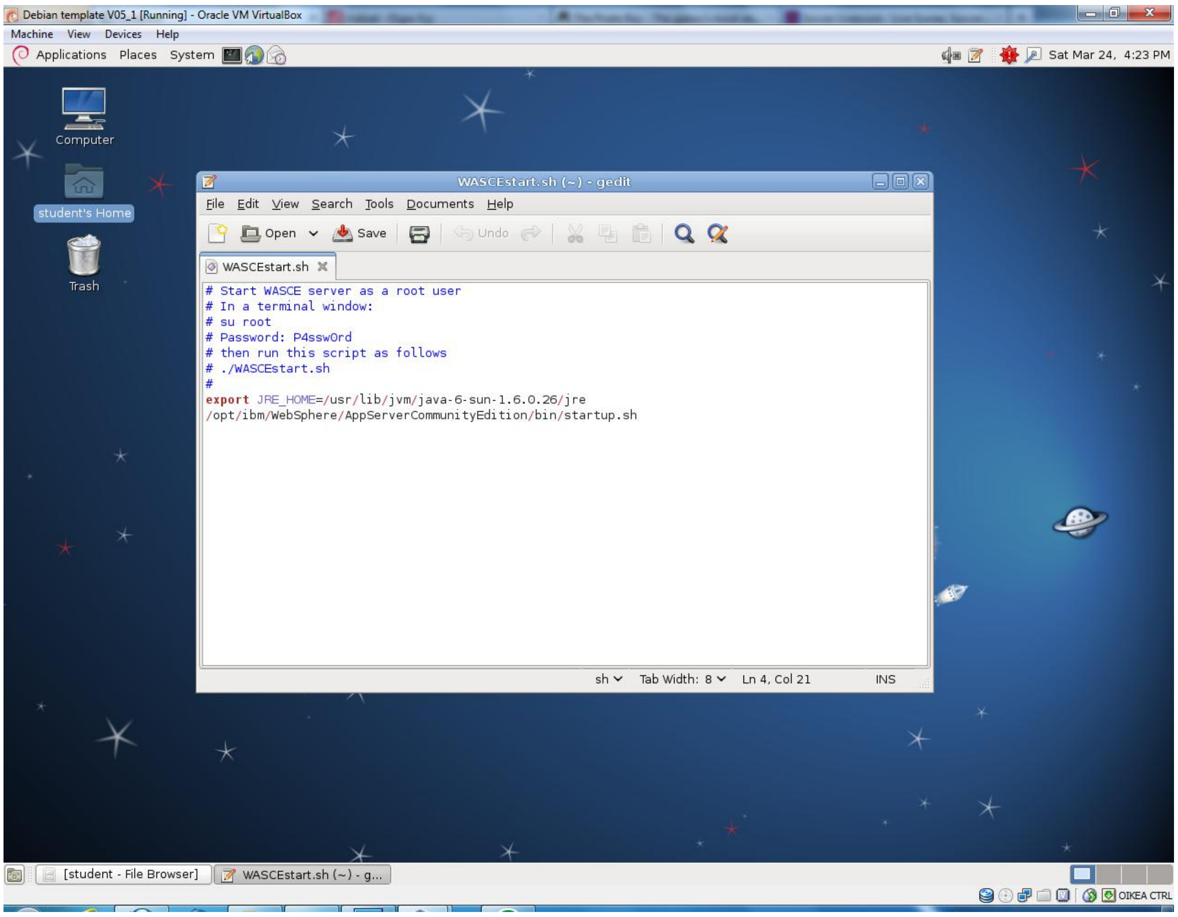
4.1.2 IBM WASCE setup and use

Now that we have the database part ready, we need to do some preparations to the application server. This requires two things: First add the Oracle JDBC driver to WASCE repository and then establish a new database (connection) pool that can be used in the Web service code.

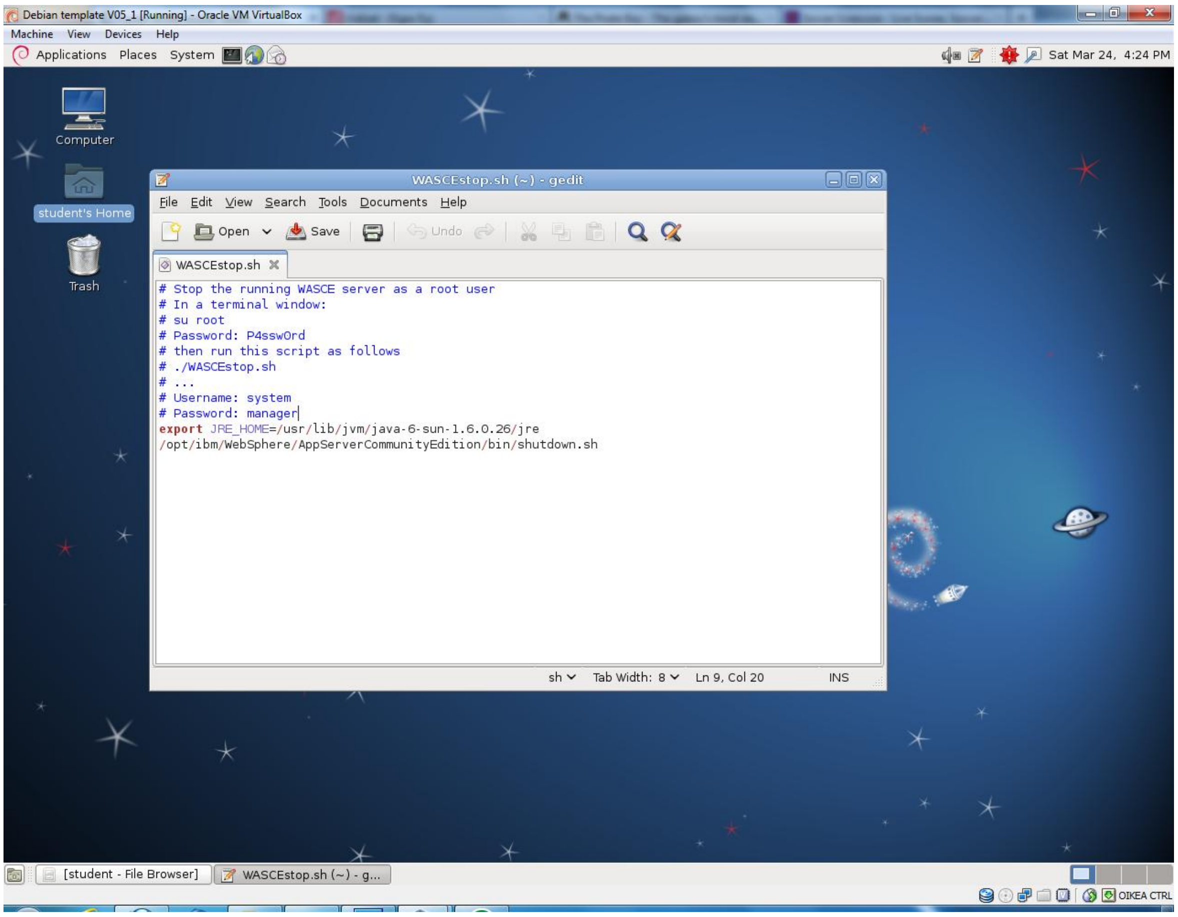
1) First we need to check that we have a startup and shutdown scripts for WASCE, placed in the /home/student directory. These scripts are used from the terminal as root user. One script for start and one for shutdown:



They should have contents like this. If they do not exist yet, create these now. The paths can be double checked using the virtual environment documentation and file browser.


Debian template V05_1 [Running] - Oracle VM VirtualBox
Machine View Devices Help
Applications Places System
Sat Mar 24, 4:23 PM
Computer
student's Home
Trash
WASCEstart.sh (~) - gedit
File Edit View Search Tools Documents Help
Open Save Undo
WASCEstart.sh
sh Tab Width: 8 Ln 4, Col 21 INS
[student - File Browser]
WASCEstart.sh (~) - g...

```
# Start WASCE server as a root user
# In a terminal window:
# su root
# Password: P4ssw0rd
# then run this script as follows
# ./WASCEstart.sh
#
export JRE_HOME=/usr/lib/jvm/java-6-sun-1.6.0.26/jre
/opt/ibm/WebSphere/AppServerCommunityEdition/bin/startup.sh
```

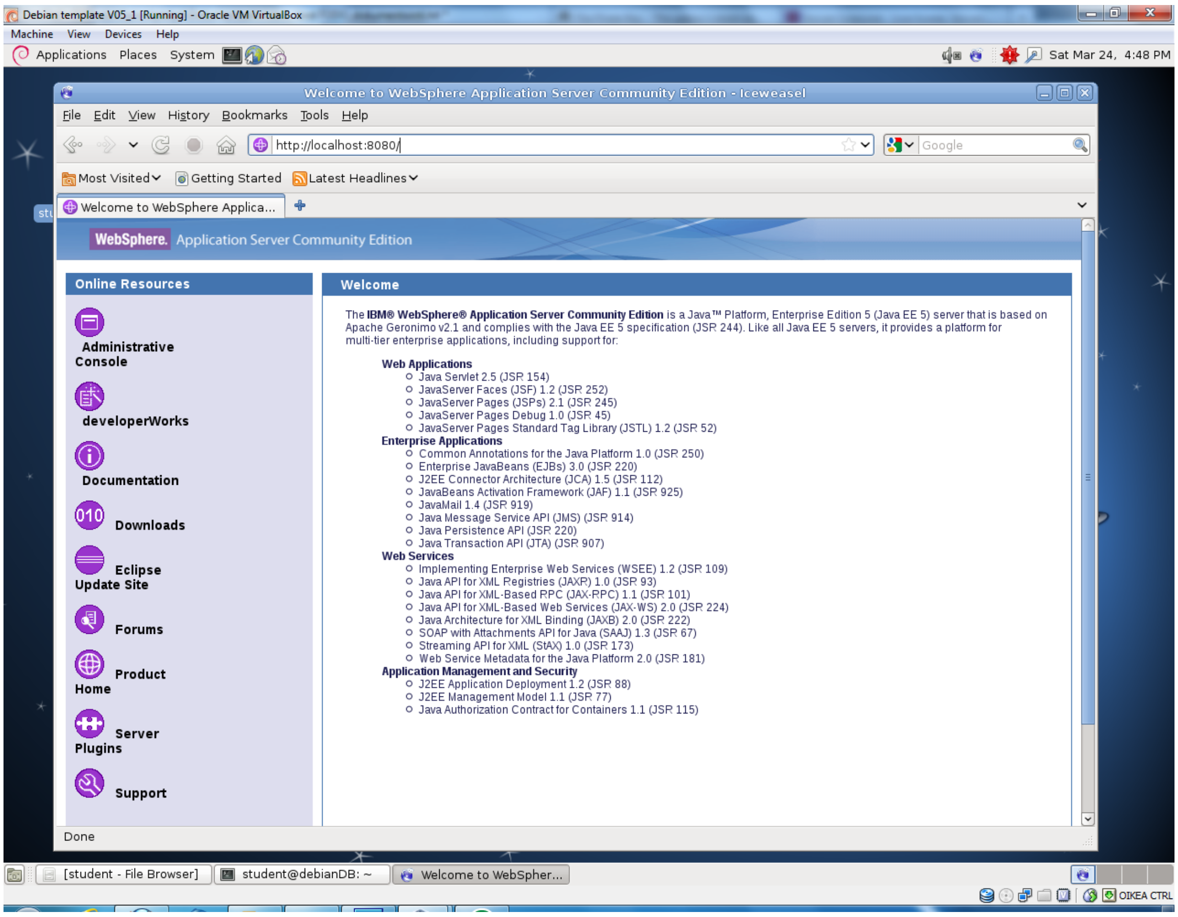


```
# Stop the running WASCE server as a root user
# In a terminal window:
# su root
# Password: P4ssw0rd
# then run this script as follows
# ./WASCEstop.sh
# ...
# Username: system
# Password: manager
export JRE_HOME=/usr/lib/jvm/java-6-sun-1.6.0.26/jre
/opt/ibm/WebSphere/AppServerCommunityEdition/bin/shutdown.sh
```

With these scripts we can start and operate the WASCE application server.

1) As a root user start the WASCE server with the WASCEstart.sh script. You see a short confirmation about the start on the console.

Open the Iceweasel internet browser from the applications menu and locate to http://localhost:8080. You should arrive at WASCE server homepage if it is up and running.


Debian template V05_1 [Running] - Oracle VM VirtualBox
Machine View Devices Help
Applications Places System
Sat Mar 24, 4:48 PM
Welcome to WebSphere Application Server Community Edition - Iceweasel
File Edit View History Bookmarks Tools Help
http://localhost:8080/
Google
Most Visited Getting Started Latest Headlines
Welcome to WebSphere Applica...
WebSphere. Application Server Community Edition
Online Resources
Administrative Console
developerWorks
Documentation
Downloads
Eclipse Update Site
Forums
Product Home
Server Plugins
Support
Done
[student - File Browser]
student@debianDB: ~
Welcome to WebSpher...

Welcome

The **IBM® WebSphere® Application Server Community Edition** is a Java™ Platform, Enterprise Edition 5 (Java EE 5) server that is based on Apache Geronimo v2.1 and complies with the Java EE 5 specification (JSR 244). Like all Java EE 5 servers, it provides a platform for multi-tier enterprise applications, including support for:

Web Applications
- Java Servlet 2.5 (JSR 154)
- JavaServer Faces (JSF) 1.2 (JSR 252)
- JavaServer Pages (JSPs) 2.1 (JSR 245)
- JavaServer Pages Debug 1.0 (JSR 45)
- JavaServer Pages Standard Tag Library (JSTL) 1.2 (JSR 52)

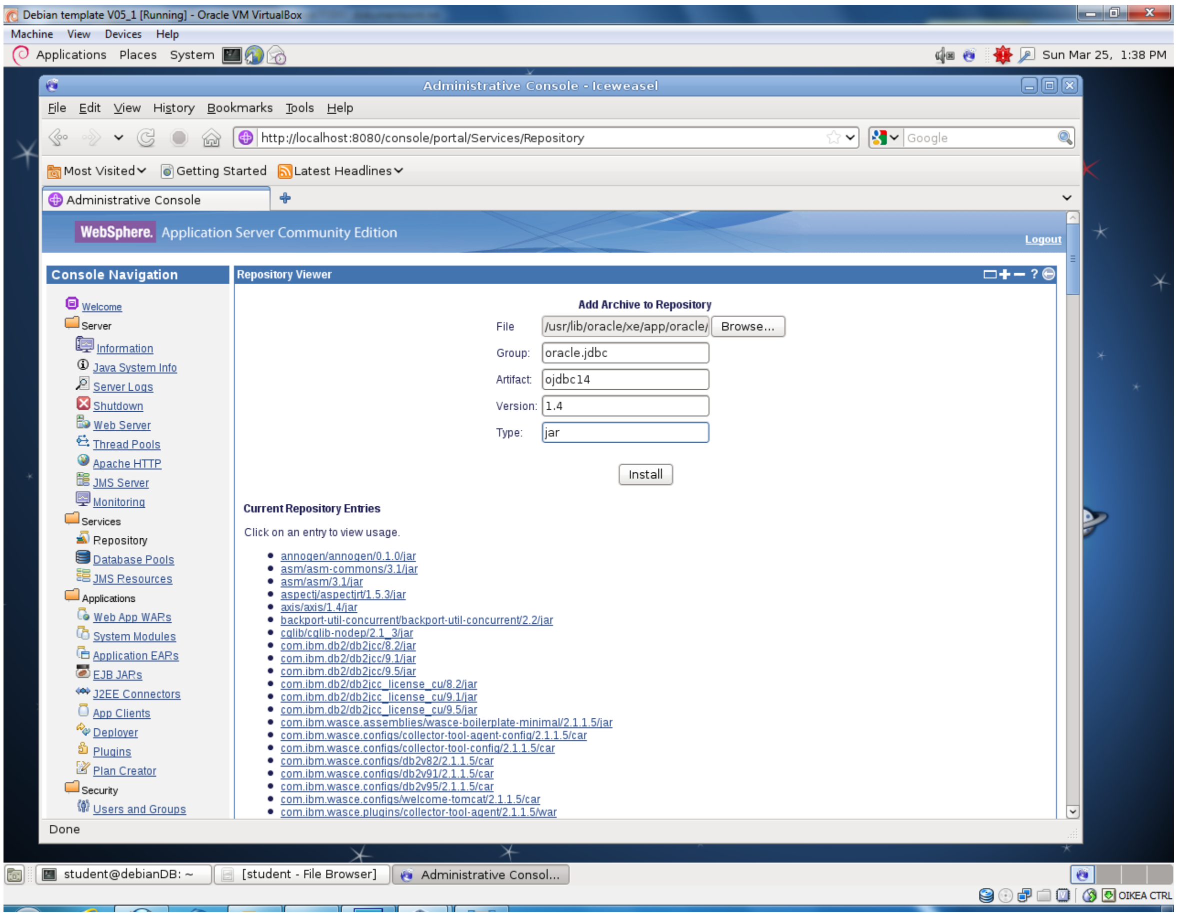
Enterprise Applications
- Common Annotations for the Java Platform 1.0 (JSR 250)
- Enterprise JavaBeans (EJBs) 3.0 (JSR 220)
- J2EE Connector Architecture (JCA) 1.5 (JSR 112)
- JavaBeans Activation Framework (JAF) 1.1 (JSR 925)
- JavaMail 1.4 (JSR 919)
- Java Message Service API (JMS) (JSR 914)
- Java Persistence API (JSR 220)
- Java Transaction API (JTA) (JSR 907)

Web Services
- Implementing Enterprise Web Services (WSEE) 1.2 (JSR 109)
- Java API for XML Registries (JAXR) 1.0 (JSR 93)
- Java API for XML-Based RPC (JAX-RPC) 1.1 (JSR 101)
- Java API for XML-Based Web Services (JAX-WS) 2.0 (JSR 224)
- Java Architecture for XML Binding (JAXB) 2.0 (JSR 222)
- SOAP with Attachments API for Java (SAAJ) 1.3 (JSR 67)
- Streaming API for XML (StAX) 1.0 (JSR 173)
- Web Service Metadata for the Java Platform 2.0 (JSR 181)

Application Management and Security
- J2EE Application Deployment 1.2 (JSR 88)
- J2EE Management Model 1.1 (JSR 77)
- Java Authorization Contract for Containers 1.1 (JSR 115)

Choose Administrative console from the left-side menu. The server requires you to login. Use system/manager. Now we should do some changes into Services items Repository and Database pools, in this order.

2) Open the Services → Repository. Here we want to locate and add the Oracle XE JDBC driver. The driver is located in /usr/lib/oracle/xe/app/oracle/product/10.2.0/server/jdbc/lib/ojdbc14.jar. Browse this location and pick the jar file and fill in the rest of the details like shown:



Repository Viewer

Add Archive to Repository

File	/usr/lib/oracle/xe/app/oracle/
Group:	oracle.jdbc
Artifact:	ojdbc14
Version:	1.4
Type:	jar

Current Repository Entries

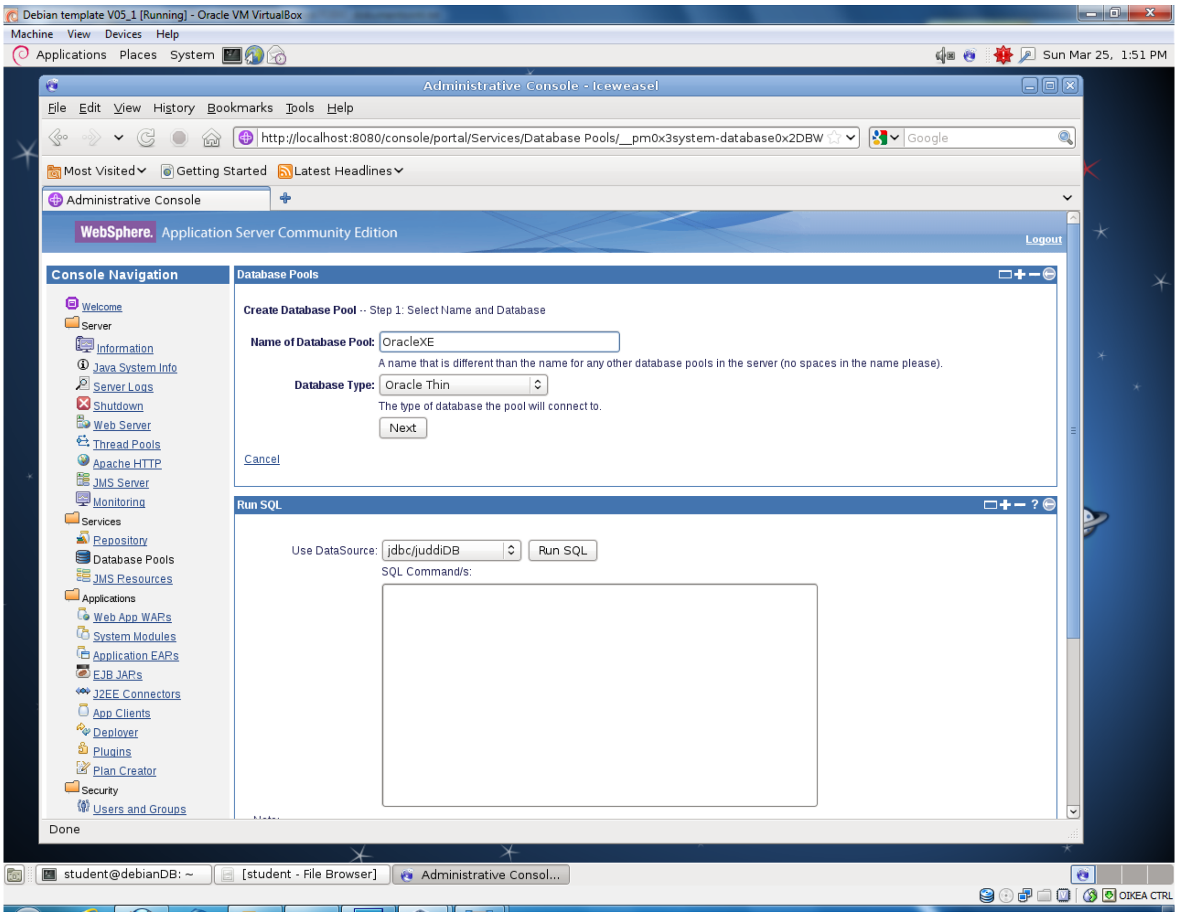
Click on an entry to view usage.

- annogen/annogen/0.1.0/jar
- asm/asm-commons/3.1/jar
- asm/asm/3.1/jar
- aspectj/aspectjrt/1.5.3/jar
- axis/axis/1.4/jar
- backport-util-concurrent/backport-util-concurrent/2.2/jar
- cglib/cglib-nodep/2.1_3/jar
- com.ibm.db2/db2jcc/8.2/jar
- com.ibm.db2/db2jcc/9.1/jar
- com.ibm.db2/db2jcc/9.5/jar
- com.ibm.db2/db2jcc_license_cu/8.2/jar
- com.ibm.db2/db2jcc_license_cu/9.1/jar
- com.ibm.db2/db2jcc_license_cu/9.5/jar
- com.ibm.wasce.assemblies/wasce-boilerplate-minimal/2.1.1.5/jar
- com.ibm.wasce.configs/collector-tool-agent-config/2.1.1.5/car
- com.ibm.wasce.configs/collector-tool-config/2.1.1.5/car
- com.ibm.wasce.configs/db2v82/2.1.1.5/car
- com.ibm.wasce.configs/db2v91/2.1.1.5/car
- com.ibm.wasce.configs/db2v95/2.1.1.5/car
- com.ibm.wasce.configs/welcome-tomcat/2.1.1.5/car
- com.ibm.wasce.plugins/collector-tool-agent/2.1.1.5/war

Then press Install and now the driver is added to the WASCE repository. It should be visible among all the repository entries listed below. The name starts with oracle.jdbc.

3) Open Services → Database pools. Here we can see the existing database pools that are available for applications. We want to add a new one for our Oracle XE database connection.

4) Choose Create a new database pool -> Using the Geronimo database pool wizard. Enter the following details in Step 1 and press Next:


Administrative Console - Iceweasel
http://localhost:8080/console/portal/Services/Database Pools/__pm0x3system-database0x2DBW
WebSphere. Application Server Community Edition
Console Navigation
Database Pools
Run SQL

Create Database Pool -- Step 1: Select Name and Database

Name of Database Pool: OracleXE

A name that is different than the name for any other database pools in the server (no spaces in the name please).

Database Type: Oracle Thin

The type of database the pool will connect to.

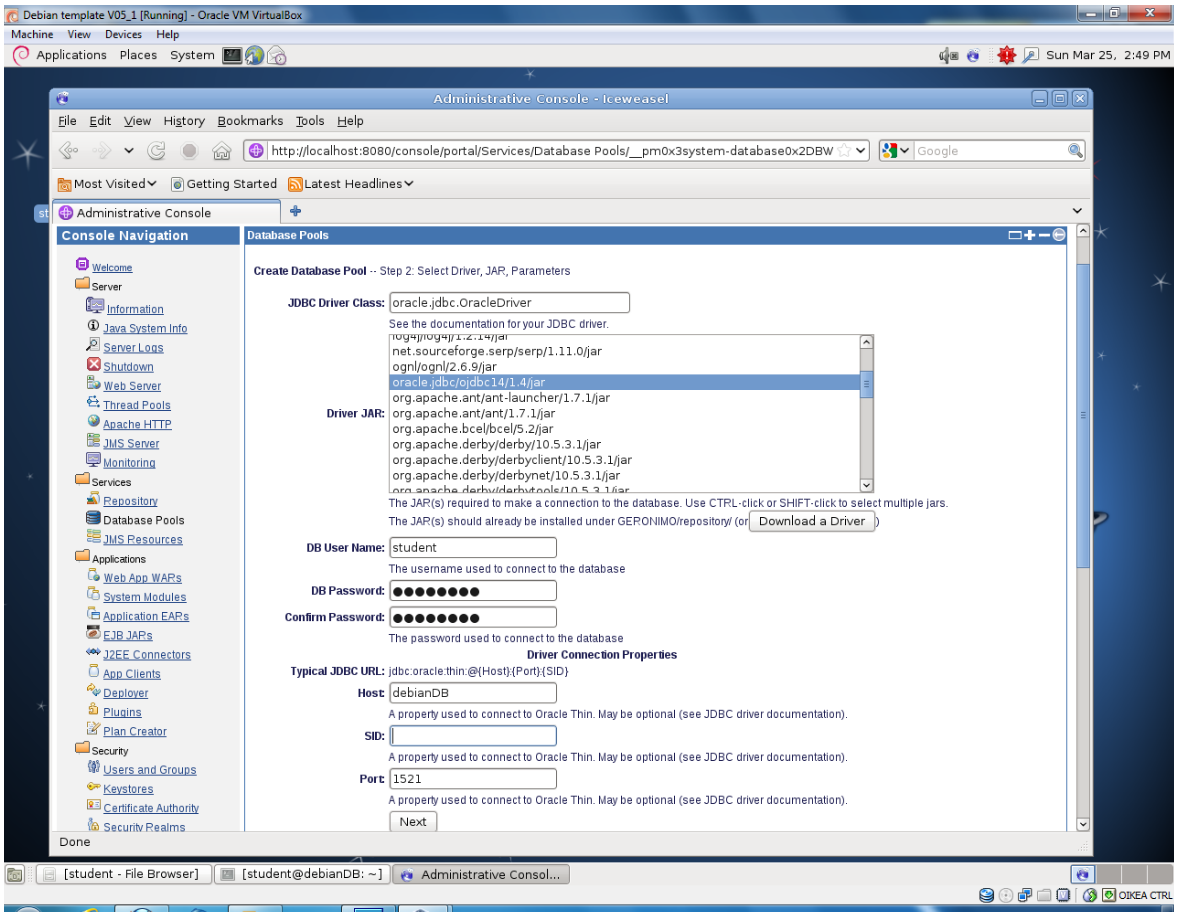
Next

Cancel

Use DataSource: jdbc/juddiDB Run SQL

SQL Command/s:

5) In Step 2 we pick up our new driver class from the list. Add username, password and hostname like shown in the picture and press Next:



Create Database Pool -- Step 2: Select Driver, JAR, Parameters

JDBC Driver Class: oracle.jdbc.OracleDriver
See the documentation for your JDBC driver.

Driver JAR:
```
net.sourceforge.serp/serp/1.11.0/jar
ognl/ognl/2.6.9/jar
oracle.jdbc/ojdbc14/1.4/jar
org.apache.ant/ant-launcher/1.7.1/jar
org.apache.ant/ant/1.7.1/jar
org.apache.bcel/bcel/5.2/jar
org.apache.derby/derby/10.5.3.1/jar
org.apache.derby/derbyclient/10.5.3.1/jar
org.apache.derby/derbynet/10.5.3.1/jar
```
The JAR(s) required to make a connection to the database. Use CTRL-click or SHIFT-click to select multiple jars.
The JAR(s) should already be installed under GERONIMO/repository/ (or Download a Driver)

DB User Name: student
The username used to connect to the database

DB Password: ●●●●●●●●
Confirm Password: ●●●●●●●●
The password used to connect to the database

Driver Connection Properties

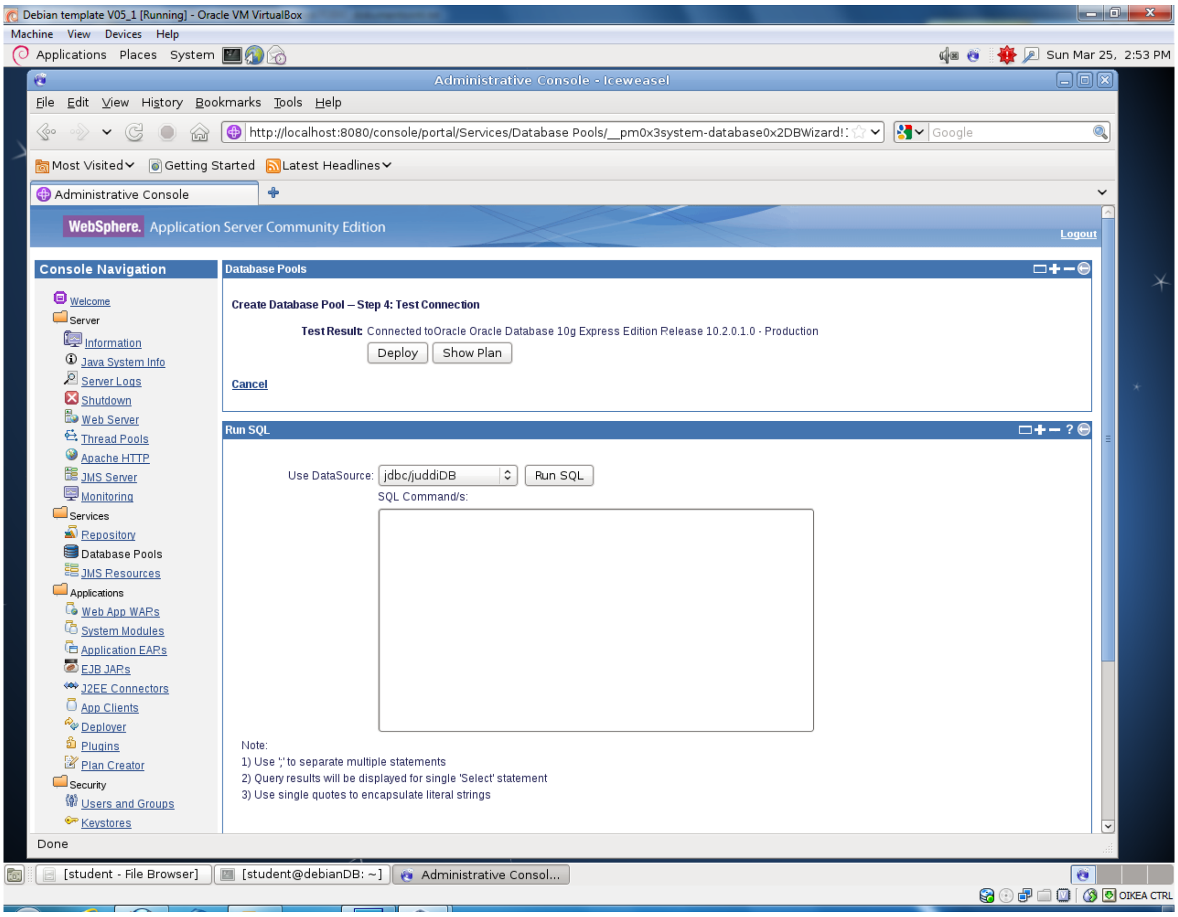
Typical JDBC URL: jdbc:oracle:thin:@{Host}:{Port}:{SID}

Host: debianDB
A property used to connect to Oracle Thin. May be optional (see JDBC driver documentation).

SID:
A property used to connect to Oracle Thin. May be optional (see JDBC driver documentation).

Port: 1521
A property used to connect to Oracle Thin. May be optional (see JDBC driver documentation).

6) In Step 3 you can choose to test the connection or skip and deploy. We can test the connection and then you should see a successful message like this:



http://localhost:8080/console/portal/Services/Database Pools/__pm0x3system-database0x2DBWizard!

Create Database Pool – Step 4: Test Connection

Test Result: Connected to Oracle Oracle Database 10g Express Edition Release 10.2.0.1.0 - Production

Deploy | Show Plan

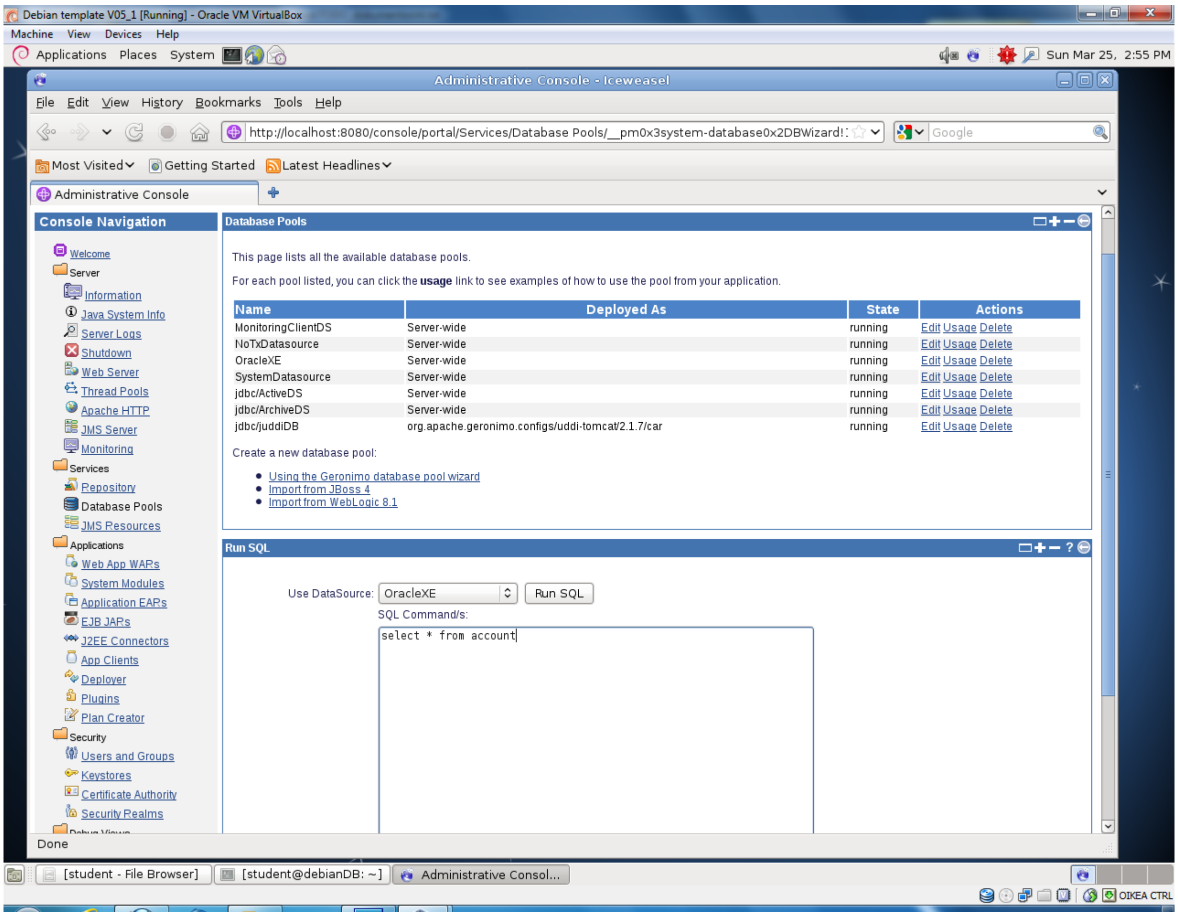
Cancel

Use DataSource: jdbc/juddiDB | Run SQL

SQL Command/s:

Note:
1) Use ';' to separate multiple statements
2) Query results will be displayed for single 'Select' statement
3) Use single quotes to encapsulate literal strings

Then choose Deploy. Now the newly-created Database pool should be shown on list with name OracleXE. You can do a test query to our test table (ACCOUNT) in the Run SQL box on screen. Choose our new database pool and enter a query. If it gets the result it is shown below.



Database Pools

This page lists all the available database pools.

For each pool listed, you can click the **usage** link to see examples of how to use the pool from your application.

Name	Deployed As	State	Actions
MonitoringClientDS	Server-wide	running	Edit Usage Delete
NoTxDatasource	Server-wide	running	Edit Usage Delete
OracleXE	Server-wide	running	Edit Usage Delete
SystemDatasource	Server-wide	running	Edit Usage Delete
jdbc/ActiveDS	Server-wide	running	Edit Usage Delete
jdbc/ArchiveDS	Server-wide	running	Edit Usage Delete
jdbc/juddiDB	org.apache.geronimo.configs/uddi-tomcat/2.1.7/car	running	Edit Usage Delete

Create a new database pool:

- Using the Geronimo database pool wizard
- Import from JBoss 4
- Import from WebLogic 8.1

Run SQL

Use DataSource: OracleXE Run SQL

SQL Command/s:

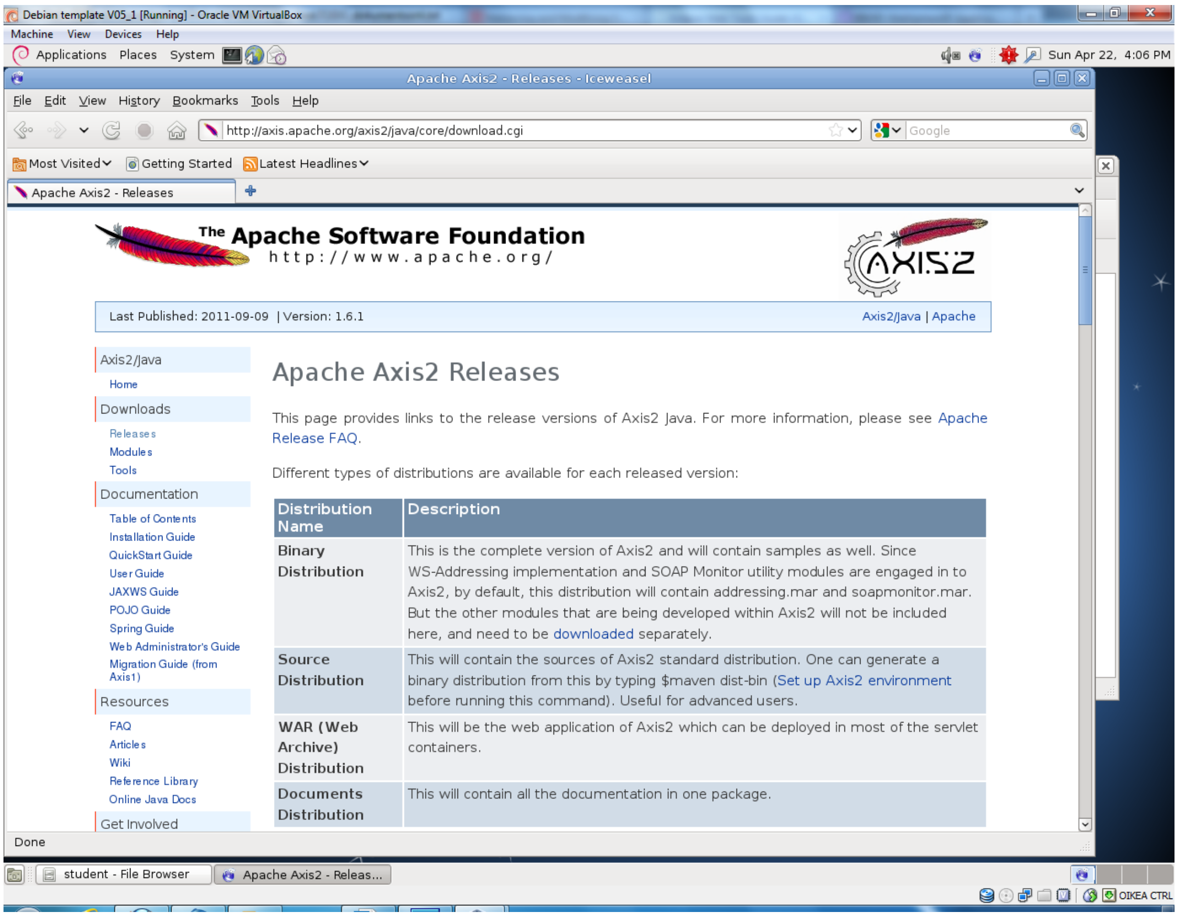
```
select * from account
```

Now we have needed Oracle DB connection for our Java Web service test. (Chong 2010, 65-75.)

4.1.3 Setup workspace and IBM Data Studio use

When using the IBM Data Studio IDE to build our Java Web service, we need to do some changes to default configurations of the tool. I decided to use JAX-WS style to create the Web service and therefore we need to have Apache Axis2 runtime configured for the Data Studio.

1) Open the web browser and go to Axis2 download page http://ws.apache.org/axis2/download.cgi:


Debian template V05_1 [Running] - Oracle VM VirtualBox
Machine View Devices Help
Applications Places System
Sun Apr 22, 4:06 PM
Apache Axis2 - Releases - Iceweasel
File Edit View History Bookmarks Tools Help
http://axis.apache.org/axis2/java/core/download.cgi
Google
Most Visited Getting Started Latest Headlines
Apache Axis2 - Releases
The Apache Software Foundation
http://www.apache.org/
AXIS2
Done
student - File Browser
Apache Axis2 - Releas...

Last Published: 2011-09-09 | Version: 1.6.1 Axis2/Java | Apache

Axis2/Java
- Home

Downloads
- Releases
- Modules
- Tools

Documentation
- Table of Contents
- Installation Guide
- QuickStart Guide
- User Guide
- JAXWS Guide
- POJO Guide
- Spring Guide
- Web Administrator's Guide
- Migration Guide (from Axis1)

Resources
- FAQ
- Articles
- Wiki
- Reference Library
- Online Java Docs

Get Involved

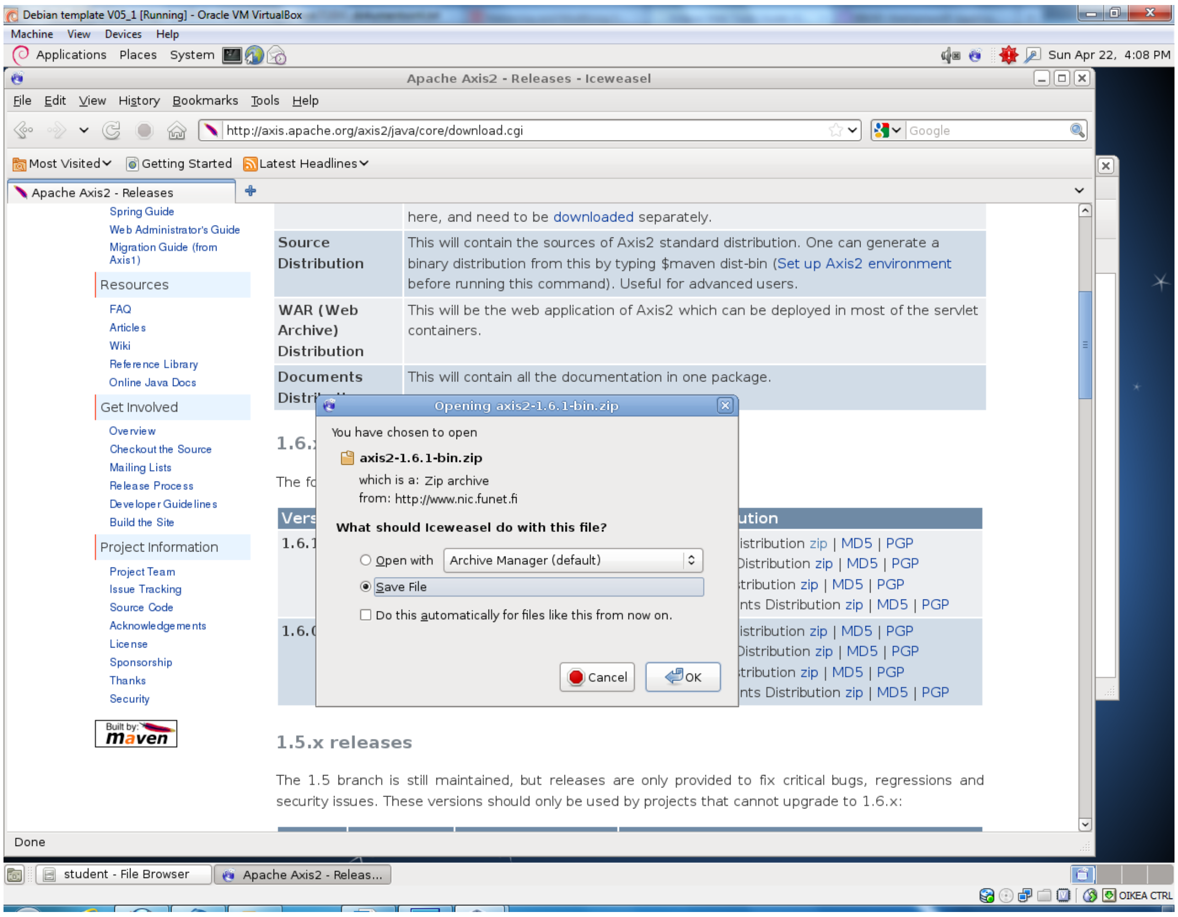
Apache Axis2 Releases

This page provides links to the release versions of Axis2 Java. For more information, please see Apache Release FAQ.

Different types of distributions are available for each released version:

Distribution Name	Description
Binary Distribution	This is the complete version of Axis2 and will contain samples as well. Since WS-Addressing implementation and SOAP Monitor utility modules are engaged in to Axis2, by default, this distribution will contain addressing.mar and soapmonitor.mar. But the other modules that are being developed within Axis2 will not be included here, and need to be downloaded separately.
Source Distribution	This will contain the sources of Axis2 standard distribution. One can generate a binary distribution from this by typing $maven dist-bin (Set up Axis2 environment before running this command). Useful for advanced users.
WAR (Web Archive) Distribution	This will be the web application of Axis2 which can be deployed in most of the servlet containers.
Documents Distribution	This will contain all the documentation in one package.

2) Pick the latest release 1.6.1 binary distribution (zip):



	here, and need to be downloaded separately.
Source Distribution	This will contain the sources of Axis2 standard distribution. One can generate a binary distribution from this by typing $maven dist-bin (Set up Axis2 environment before running this command). Useful for advanced users.
WAR (Web Archive) Distribution	This will be the web application of Axis2 which can be deployed in most of the servlet containers.
Documents Distri	This will contain all the documentation in one package.

Opening axis2-1.6.1-bin.zip

You have chosen to open

axis2-1.6.1-bin.zip

which is a: Zip archive

from: http://www.nic.funet.fi

What should Iceweasel do with this file?

( ) Open with Archive Manager (default)

(•) Save File

☐ Do this automatically for files like this from now on.

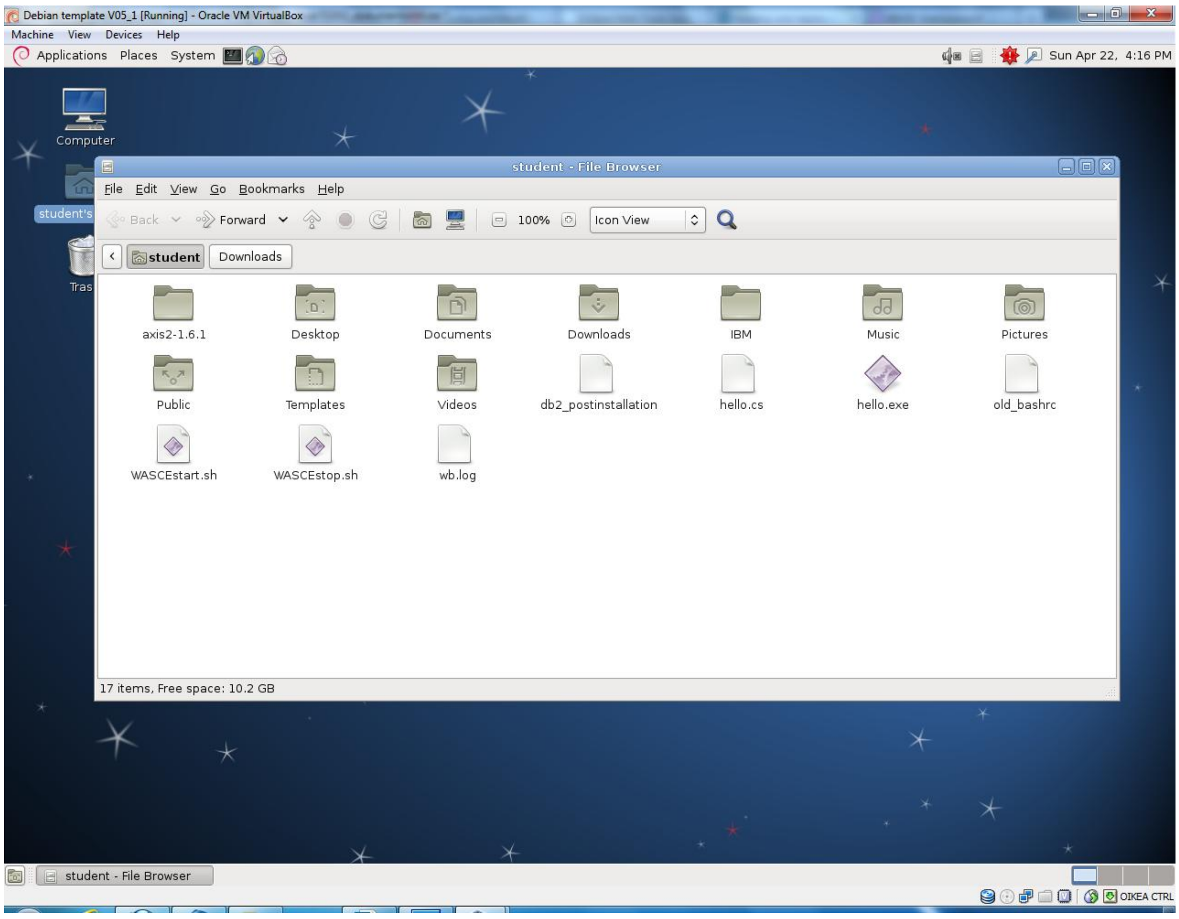
Cancel OK

1.5.x releases

The 1.5 branch is still maintained, but releases are only provided to fix critical bugs, regressions and security issues. These versions should only be used by projects that cannot upgrade to 1.6.x:

3) Save the zip file. It should be now under home/student/Downloads directory

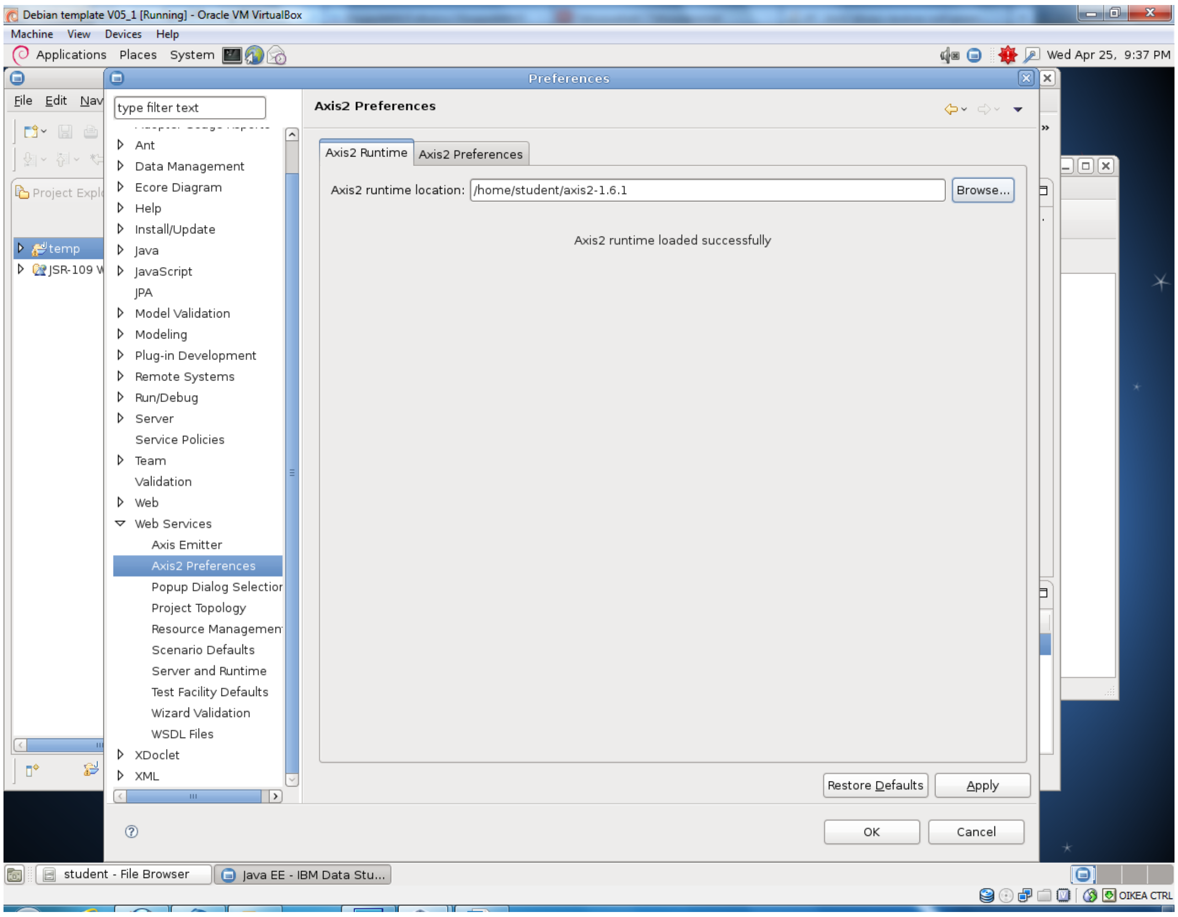
4) Unzip file under home/student/. This creates a directory axis2-1.6.1:



Now we have the needed Axis2 runtime downloaded and we can choose it as our default Web service runtime in IBM Data Studio tool. (Linuxtopia 2012. Eclipse Web Tools Guide.)

Now we can start up the IBM Data Studio from the Applications menu. Applications → Programming → IBM Data Studio 2.2.1.0. Choose the default workspace by pressing OK.

1) Choose Window → Preferences → Web services → Axis2 preferences. Browse to the Axis2 installation directory /home/student/ axis2-1.6.1 and the runtime should be now successfully loaded to preferences:



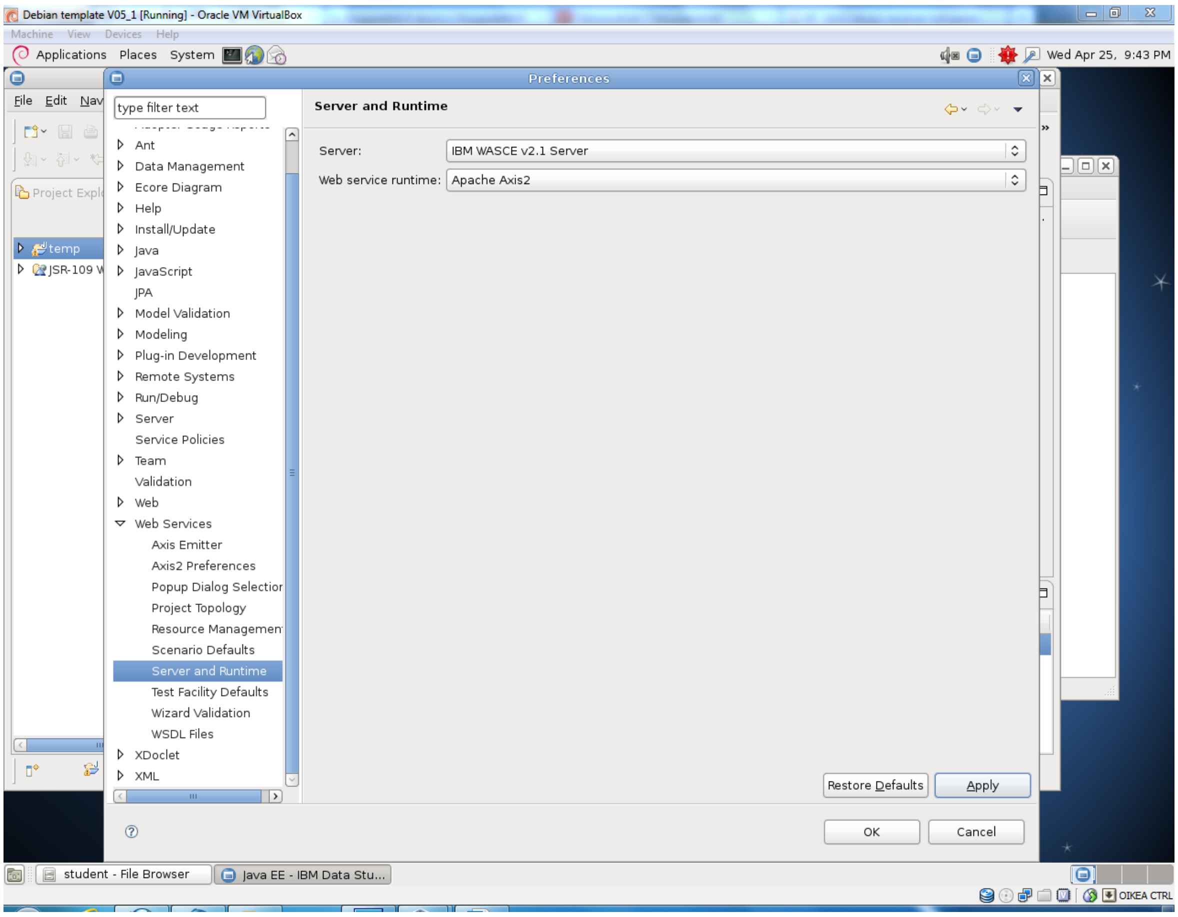
Axis2 Preferences

Axis2 runtime location: /home/student/axis2-1.6.1

Axis2 runtime loaded successfully

2) Press Apply to save this setting.

3) Choose Server and Runtime option and select following values:



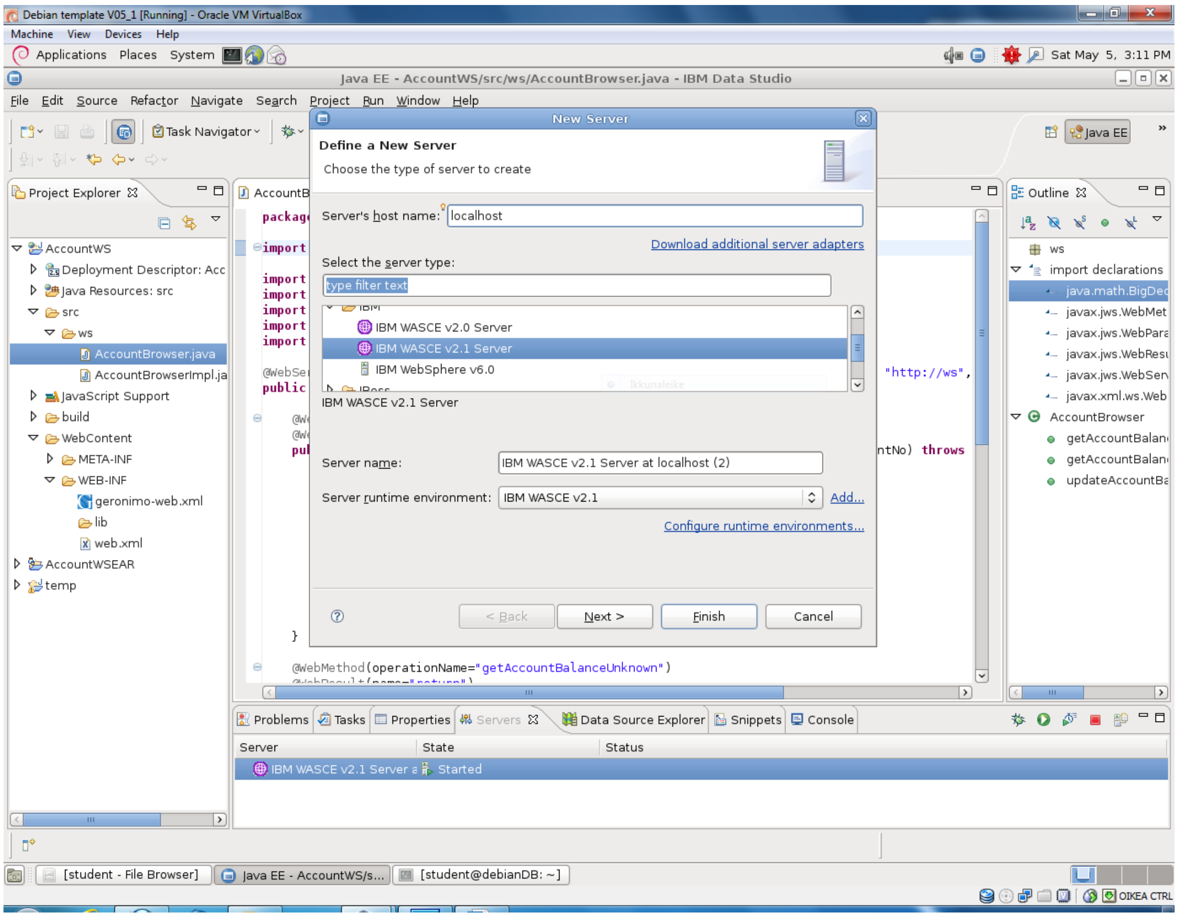
Server and Runtime

Server: IBM WASCE v2.1 Server

Web service runtime: Apache Axis2

4) Check also that WSDL Files option shows the Default Target Namespace value as http://localhost:8080. Press Apply and OK to close the preferences.

5) Check that on Servers view/tab there is a IBM WASCE server (localhost) listed. If not, add it on Servers view by right-clicking and choosing New → Server and choose these values for it:


New Server

Define a New Server

Choose the type of server to create

Server's host name: localhost

Select the server type:

- IBM
  - IBM WASCE v2.0 Server
  - IBM WASCE v2.1 Server
  - IBM WebSphere v6.0

IBM WASCE v2.1 Server

Server name: IBM WASCE v2.1 Server at localhost (2)

Server runtime environment: IBM WASCE v2.1

Server	State	Status
IBM WASCE v2.1 Server a	Started	

Java EE - AccountWS/src/ws/AccountBrowser.java - IBM Data Studio
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
Download additional server adapters
Add...
Configure runtime environments...
< Back
Next >
Finish
Cancel
Problems Tasks Properties Servers Data Source Explorer Snippets Console
Outline

Choose Finish and now the application server should be listed on Servers tab (if not already there in the beginning).

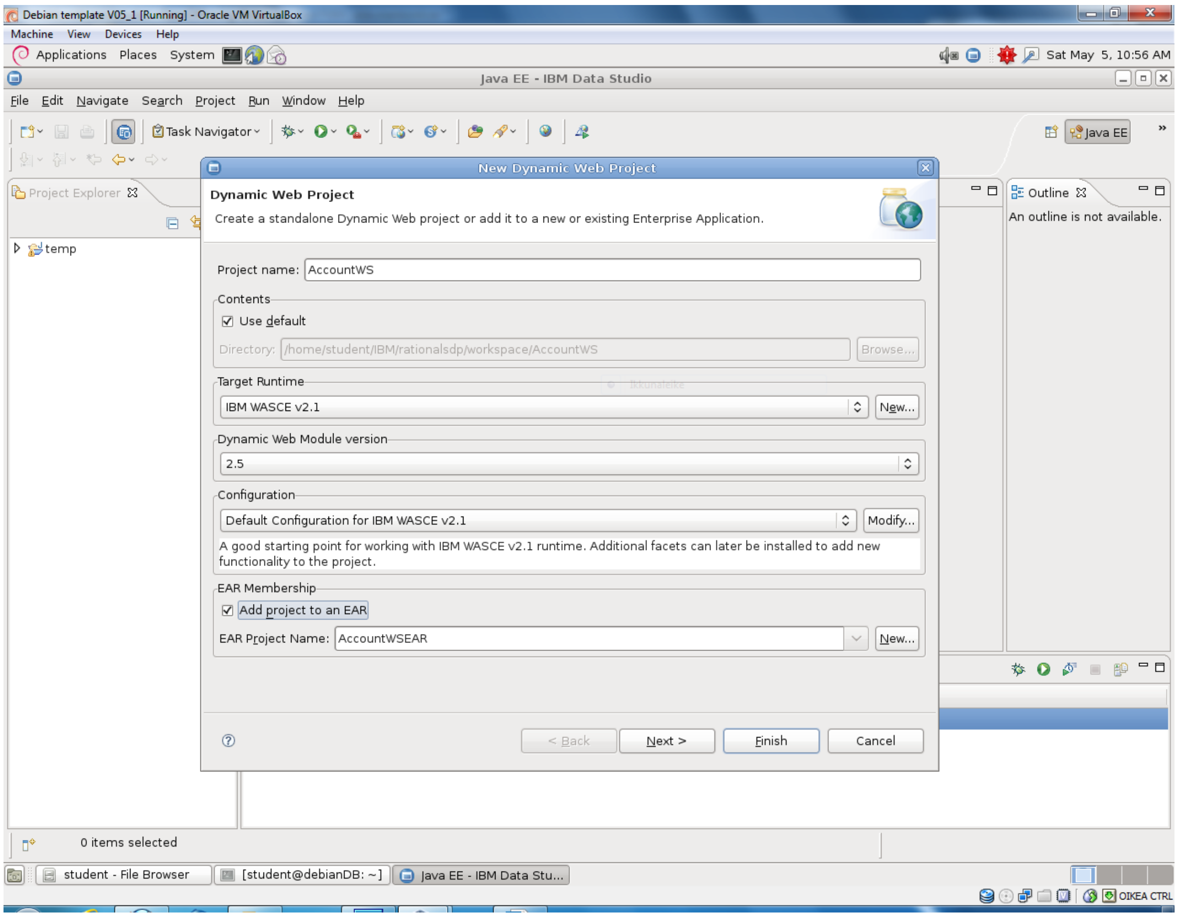
Now we are able to build our Web service and run it on IBM WASCE server and Apache Axis2 Web service engine.

4.1.4 Create Web service project and deploy the service

Next task is to create a Java project for Web service, check all the classes and deployment descriptors, save the project, start up the database and application server and deploy the Web service.

1) Select File → New → Dynamic Web Project

2) Enter the name ”AccountWS” for the project, choose ”Add project to an EAR” and accept the following default values by choosing Finish:


Dynamic Web Project

Create a standalone Dynamic Web project or add it to a new or existing Enterprise Application.

Project name: AccountWS

Contents

☑ Use default

Directory: /home/student/IBM/rationalsdp/workspace/AccountWS

Target Runtime: IBM WASCE v2.1

Dynamic Web Module version: 2.5

Configuration: Default Configuration for IBM WASCE v2.1

A good starting point for working with IBM WASCE v2.1 runtime. Additional facets can later be installed to add new functionality to the project.

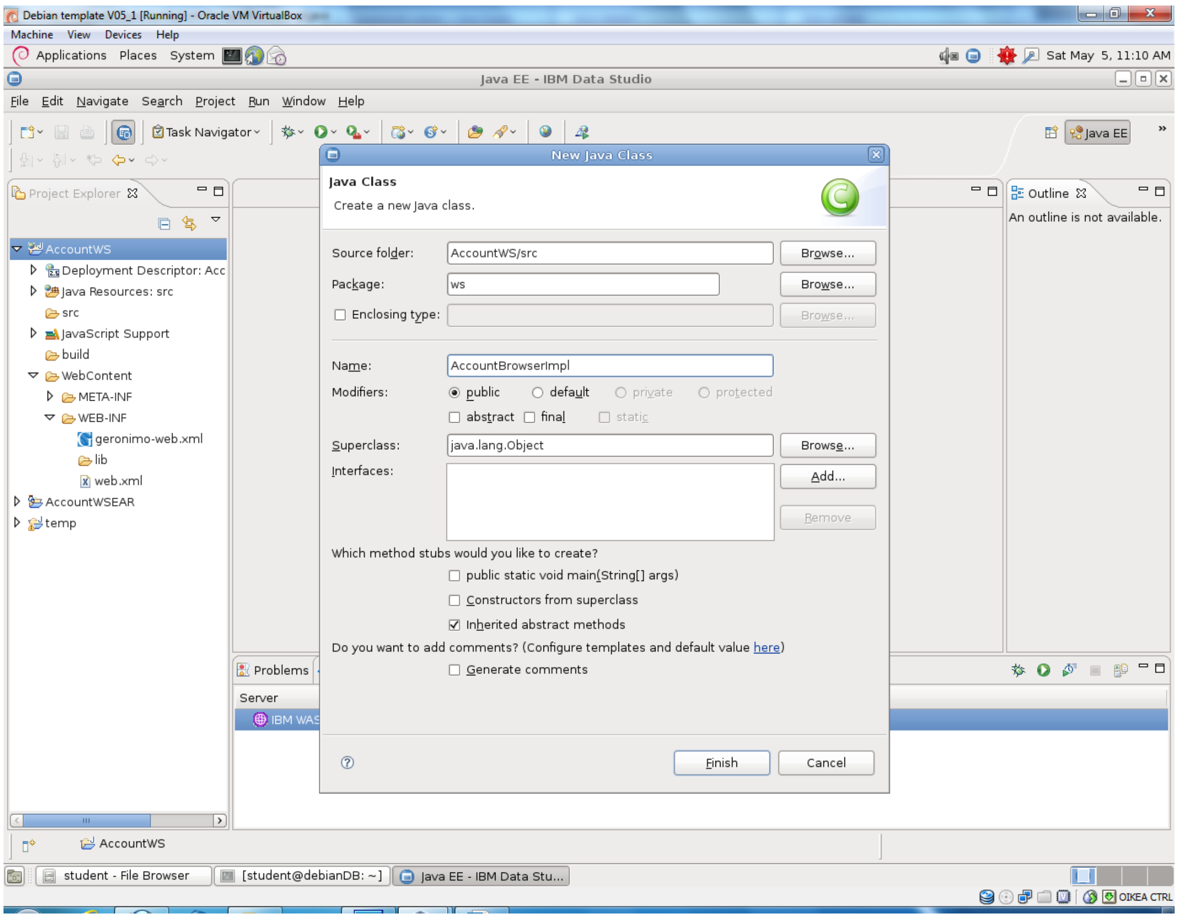
EAR Membership

☑ Add project to an EAR

EAR Project Name: AccountWSEAR

Now a base for the project should be created. Only default deployment descriptors web.xml and geronimo-web.xml now exist. Next thing is to create our Web service classes and modify the deployment descriptors accordingly.

1) Pick the new AccountWS project, right-click and choose New → Class. Add package name ws and class name AccountBrowserImpl. The rest is OK with the defaults:


Java Class

Create a new Java class.

Source folder: AccountWS/src

Package: ws

Enclosing type:

Name: AccountBrowserImpl

Modifiers: [✓ public] default private protected

abstract final static

Superclass: java.lang.Object

Interfaces:

Which method stubs would you like to create?

public static void main(String[] args)

Constructors from superclass

[✓ Inherited abstract methods]

Do you want to add comments? (Configure templates and default value here)

Generate comments

2) Copy/paste the source code for AccountBrowserImpl.java class:

```
package ws;

import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.xml.ws.WebServiceException;
import java.math.BigDecimal;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

public class AccountBrowserImpl {

    private Context ctx;
    private Connection conn;
    private PreparedStatement  stmt;
    private ResultSet  rs;

    public BigDecimal getAccountBalanceOk(String accountNo) throws WebServiceException {

        BigDecimal accountBalance = new BigDecimal(0.0);

        try {
              ctx = new InitialContext(); // This knows the jdbc datasource

              javax.sql.DataSource ds =
                  (javax.sql.DataSource)ctx.lookup("java:comp/env/jdbc/MyDataSource"); // This is the Oracle XE

              conn = ds.getConnection(); // Try to make a connection

              // Select from ACCOUNT table
```

```
            String sqlStatement = "SELECT balance FROM account " +
                                  "WHERE account_no = ?";


            stmt = conn.prepareStatement(sqlStatement);
            stmt.setString(1, accountNo);
            rs = stmt.executeQuery();

            if (rs.next()) {
              accountBalance = rs.getBigDecimal(1);
            }  else {
              throw new SQLException("No data found");
            }

          } catch (NamingException nex) {
              nex.printStackTrace();
              throw new WebServiceException("Database service not found: " + nex.getMessage());
          } catch (SQLException sex) {
              sex.printStackTrace();
              throw new WebServiceException("SQL Exception " + sex.getErrorCode() + " " + sex.getMessage());
          } catch (Exception ex) {
              ex.printStackTrace();
              throw new WebServiceException("Other exception: " + ex.getMessage());
          } finally {
              try {
                  if (rs != null) { rs.close(); }
                  if (stmt != null) { stmt.close(); }
                  if (conn != null) { conn.close(); }
              } catch (SQLException se) {
                  se.printStackTrace();
                  throw new WebServiceException("Error closing resources " + se.getErrorCode() +" " + se.getMessage());
              }
          }

      return accountBalance;
  }

  public BigDecimal getAccountBalanceUnknown(String accountNo) throws WebServiceException {

      BigDecimal accountBalance = new BigDecimal(0.0);

      try {
            ctx = new InitialContext();

            javax.sql.DataSource ds =
                (javax.sql.DataSource)ctx.lookup("java:comp/env/jdbc/MyDataSource");

            conn = ds.getConnection();

            // Try to select from ACCOUNT2 table. Table does not actually exist!

            String sqlStatement = "SELECT balance FROM account2 " +
                                  "WHERE account_no = ?";

            stmt = conn.prepareStatement(sqlStatement);
            stmt.setString(1, accountNo);
            rs = stmt.executeQuery();

            if (rs.next()) {
              accountBalance = rs.getBigDecimal(1);
            }

          } catch (NamingException nex) {
              nex.printStackTrace();
              throw new WebServiceException("Database service not found: " + nex.getMessage());
          } catch (SQLException sex) {
              sex.printStackTrace();
              throw new WebServiceException("SQL Exception " + sex.getErrorCode() +" " + sex.getMessage());
          } catch (Exception ex) {
              ex.printStackTrace();
              throw new WebServiceException("Other exception: " + ex.getMessage());
            } finally {
                try {
                    if (rs != null) { rs.close(); }
                    if (stmt != null) { stmt.close(); }
                    if (conn != null) { conn.close(); }
                } catch (SQLException se) {
                    se.printStackTrace();
                    throw new WebServiceException("Error closing resources " + se.getErrorCode() +" " + se.getMessage());
                }
            }

        return accountBalance;
    }

    public void updateAccountBalanceLock(String accountNo, BigDecimal addBalance) throws WebServiceException {

        try {
              ctx = new InitialContext();

              javax.sql.DataSource ds =
                  (javax.sql.DataSource)ctx.lookup("java:comp/env/jdbc/MyDataSource");

              conn = ds.getConnection();

              // Try to SELECT from account table by requesting a lock with FOR UPDATE NOWAIT
              // NOWAIT returns error immediately if another session has the lock (in this case yes)

              String sqlStatement = "SELECT balance FROM account " +
                                    "WHERE account_no = ? " +
                                    "FOR UPDATE NOWAIT";

              stmt = conn.prepareStatement(sqlStatement);
              stmt.setString(1, accountNo);
              rs = stmt.executeQuery();

            } catch (NamingException nex) {
                nex.printStackTrace();
                throw new WebServiceException("Database service not found: " + nex.getMessage());
            } catch (SQLException sex) {
                sex.printStackTrace();
                throw new WebServiceException("SQL Exception "+ sex.getErrorCode() +" " + sex.getMessage());
            } catch (Exception ex) {
                ex.printStackTrace();
                throw new WebServiceException("Other exception: " + ex.getMessage());
            } finally {
                try {
                    if (rs != null) { rs.close(); }
                    if (stmt != null) { stmt.close(); }
                    if (conn != null) { conn.close(); }
                } catch (SQLException se) {
                    se.printStackTrace();
                    throw new WebServiceException("Error closing resources " + se.getErrorCode() +" " + se.getMessage());
                }
            }

    }
}
```

3) Create another class called AccountBrowser. Use same package ws. This is the actual Web service class in this test case, done by using JAX-WS style (annotations) which is very easy way to tell that I want to publish this code as a Web service. Copy/paste the source code for AccountBrowser.java class file:

```
package ws;
```

```
import java.math.BigDecimal;

import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebResult;
import javax.jws.WebService;
import javax.xml.ws.WebServiceException;

@WebService(name="AccountBrowser", serviceName = "AccountService", targetNamespace = "http://ws", portName = "AccountSer-
vicePort")
public class AccountBrowser {

    @WebMethod(operationName="getAccountBalanceOk")
    @WebResult(name="return")
    public BigDecimal getAccountBalanceOk(@WebParam(name = "accountNo") String accountNo) throws WebServiceException {

        AccountBrowserImpl accBrowser = new AccountBrowserImpl();
        BigDecimal retBalance;

        try {
            retBalance = accBrowser.getAccountBalanceOk(accountNo);
        } catch (WebServiceException ex) {
            throw ex;
        }

        return retBalance;
    }

    @WebMethod(operationName="getAccountBalanceUnknown")
    @WebResult(name="return")
    public BigDecimal getAccountBalanceUnknown(@WebParam(name = "accountNo") String accountNo) throws WebServiceException
{

        AccountBrowserImpl accBrowser = new AccountBrowserImpl();
        BigDecimal retBalance;

        try {
            retBalance = accBrowser.getAccountBalanceUnknown(accountNo);
        } catch (WebServiceException ex) {
            throw ex;
        }

        return retBalance;
    }

    @WebMethod(operationName="updateAccountBalanceLock")
    public void updateAccountBalanceLock(@WebParam(name = "accountNo") String accountNo,
                                        @WebParam(name = "addBalance") BigDecimal addBalance) throws WebServiceException
{

        AccountBrowserImpl accBrowser = new AccountBrowserImpl();

        try {
            accBrowser.updateAccountBalanceLock(accountNo, addBalance);
        } catch (WebServiceException ex) {
            throw ex;
        }

    }

}
```

4) Open web.xml deployment descriptor file and choose Source view. Copy/paste following content for the file:

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

```
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee" xmlns:web="http://java.sun.com/xml/ns/javaee/web-
app_2_5.xsd" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" id="WebApp_ID" version="2.5">
  <display-name>AccountWS</display-name>
  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
    <welcome-file>index.htm</welcome-file>
    <welcome-file>index.jsp</welcome-file>
    <welcome-file>default.html</welcome-file>
    <welcome-file>default.htm</welcome-file>
    <welcome-file>default.jsp</welcome-file>
  </welcome-file-list>
  <servlet>
    <servlet-name>AccountBrowser</servlet-name>
    <servlet-class>ws.AccountBrowser</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>AccountBrowser</servlet-name>
    <url-pattern>/AccountBrowser</url-pattern>
  </servlet-mapping>
  <resource-ref>
    <res-ref-name>jdbc/MyDataSource</res-ref-name>
    <res-type>javax.sql.DataSource</res-type>
    <res-auth>Container</res-auth>
    <res-sharing-scope>Shareable</res-sharing-scope>
  </resource-ref>
</web-app>
```

5) Open geronimo-web.xml deployment descriptor file and choose Source view. Copy/paste following content for the file:

```
<?xml version="1.0" encoding="UTF-8"?><web:web-app
xmlns:app="http://geronimo.apache.org/xml/ns/j2ee/application-2.0"
xmlns:client="http://geronimo.apache.org/xml/ns/j2ee/application-client-2.0"
xmlns:conn="http://geronimo.apache.org/xml/ns/j2ee/connector-1.2"
xmlns:dep="http://geronimo.apache.org/xml/ns/deployment-1.2"
xmlns:ejb="http://openejb.apache.org/xml/ns/openejb-jar-2.2"
xmlns:log="http://geronimo.apache.org/xml/ns/loginconfig-2.0"
xmlns:name="http://geronimo.apache.org/xml/ns/naming-1.2"
xmlns:pers="http://java.sun.com/xml/ns/persistence"
xmlns:pkgen="http://openejb.apache.org/xml/ns/pkgen-2.1"
xmlns:sec="http://geronimo.apache.org/xml/ns/security-2.0"
xmlns:web="http://geronimo.apache.org/xml/ns/j2ee/web-2.0.1">
    <dep:environment>
        <dep:moduleId>
            <dep:groupId>default</dep:groupId>
            <dep:artifactId>AccountWS</dep:artifactId>
            <dep:version>1.0</dep:version>
            <dep:type>car</dep:type>
        </dep:moduleId>
        <dep:dependencies>
            <dep:dependency>
                <dep:groupId>console.dbpool</dep:groupId>
                <dep:artifactId>OracleXE</dep:artifactId>
            </dep:dependency>
        </dep:dependencies>
    </dep:environment>
    <web:context-root>/AccountWS</web:context-root>
    <name:resource-ref>
        <name:ref-name>jdbc/MyDataSource</name:ref-name>
        <name:resource-link>OracleXE</name:resource-link>
    </name:resource-ref>
</web:web-app>
```

These additions are necessary to define the Oracle datasource for our application use. Now everything is ready for the service deployment part.

6) Add a new file under AccountWSEAR project. Choose project, right-click and New → File. Location should be under META-INF and name application.xml. Copy/paste following content into file:

```
<?xml version="1.0" encoding="UTF-8"?>
<application xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:application="http://java.sun.com/xml/ns/javaee/application_5.xsd"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/application_5.xsd" id="Application_ID" version="5">
  <display-name>
AccountWSEAR</display-name>
  <module>
    <web>
      <web-uri>AccountWS.war</web-uri>
      <context-root>AccountWS</context-root>
    </web>
  </module>
</application>
```

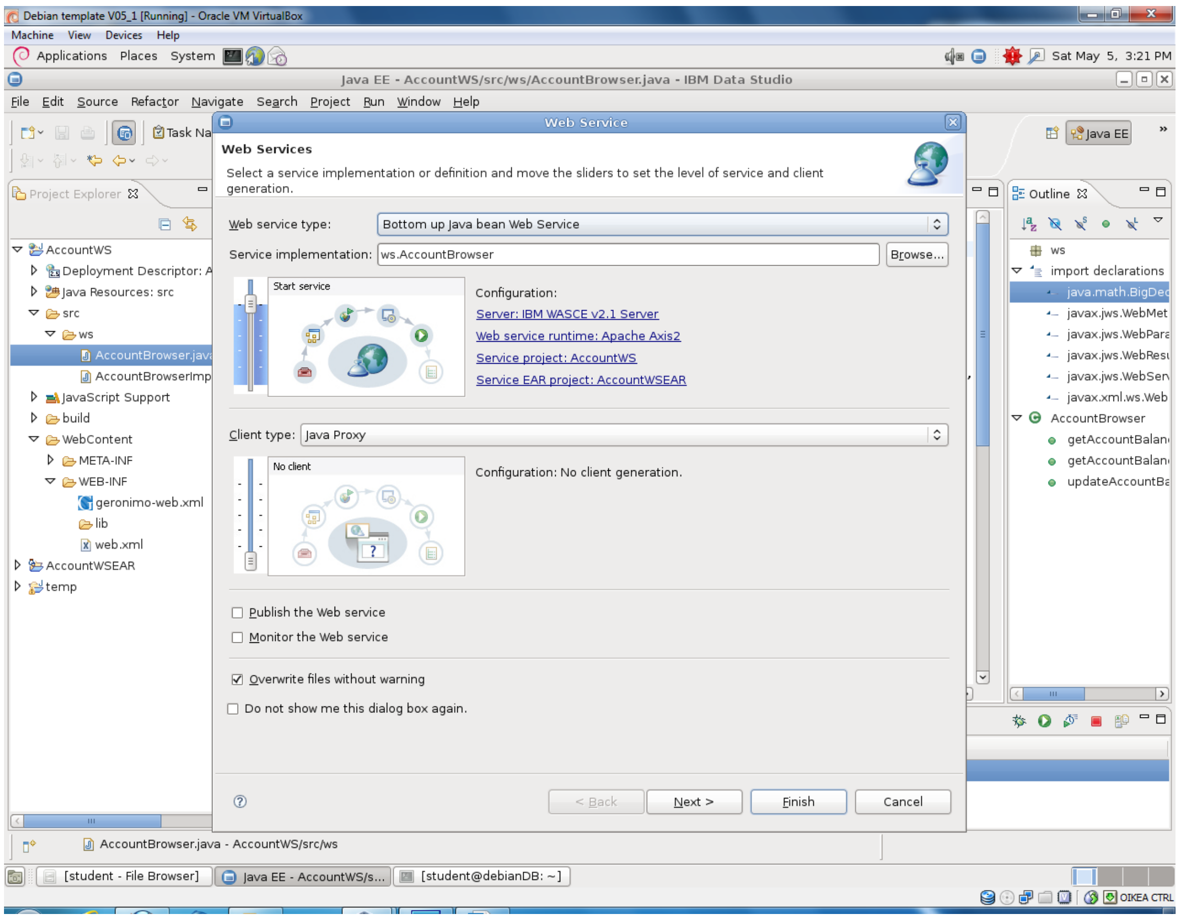
7) Start up Oracle XE database from menu (if not already running): Applications -> Oracle Database 10g Express Edition -> Start Database. It takes about 30 seconds to have the db fully running.

8) Start up the WASCE application server from the terminal using the script WASCEs-tart.sh as root user (if not already running). You can see how the server status changes in Data Studio, after couple of seconds. Wait until it seems to be in Started status.

Before begin the assemble and deployment of the Web service, check that all the following files are now either added or modified:

```
AccountBrowserImpl.java
AccountBrowser.java
web.xml
geronimo-web.xml
application.xml (under EAR project)
```

9) When both Oracle and WASCE are up and running, choose AccountBrowser source file, right-click it and choose Web Services → Create web service. Check that all the options and values are like this:



Web Service

Web Services

Select a service implementation or definition and move the sliders to set the level of service and client generation.

Web service type: Bottom up Java bean Web Service

Service implementation: ws.AccountBrowser

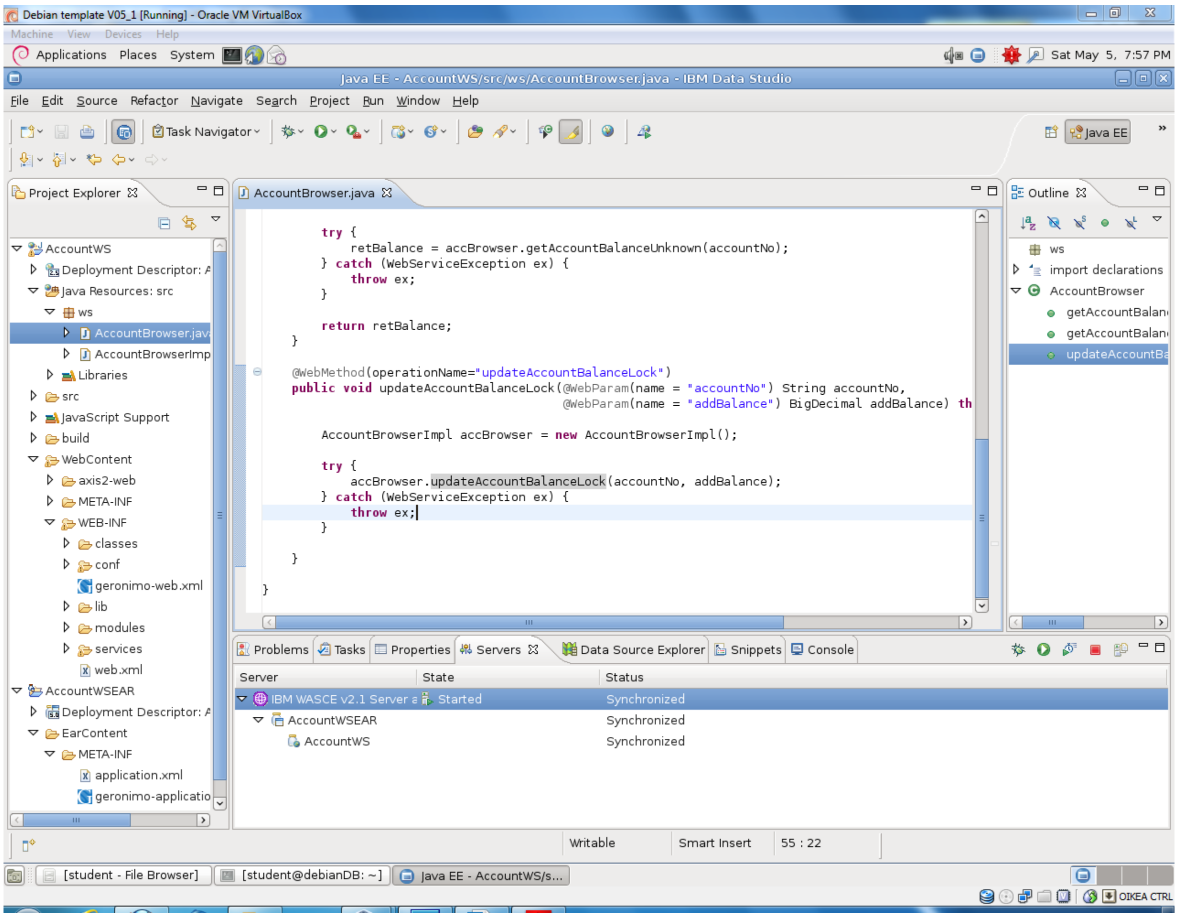
Configuration:
Server: IBM WASCE v2.1 Server
Web service runtime: Apache Axis2
Service project: AccountWS
Service EAR project: AccountWSEAR

Client type: Java Proxy

Configuration: No client generation.

☐ Publish the Web service
☐ Monitor the Web service
☑ Overwrite files without warning
☐ Do not show me this dialog box again.

Press Finish and Data Studio starts to create and deploy the Web service on WASCE server. If everything goes smoothly there should not be any errors reported on screen and you should see the application deployed under the server:



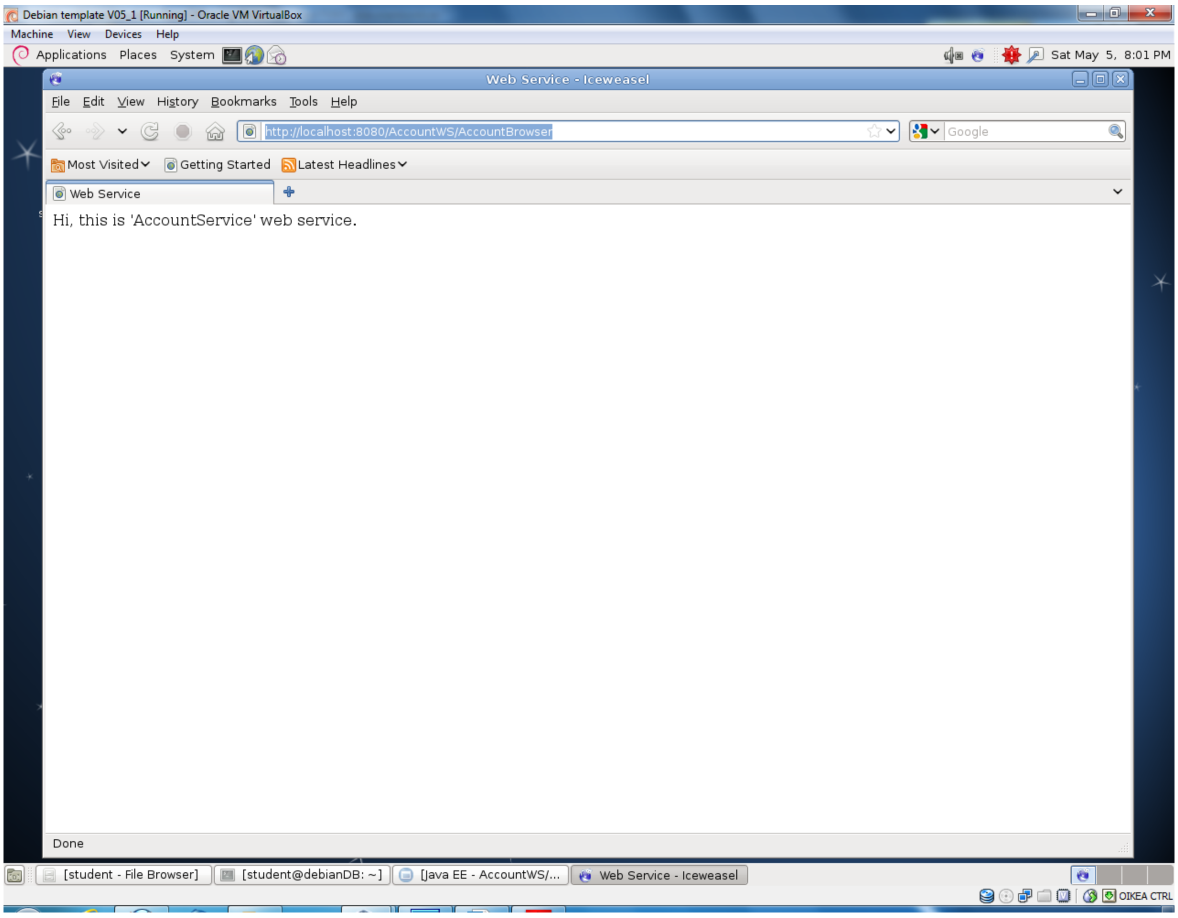
```
        try {
            retBalance = accBrowser.getAccountBalanceUnknown(accountNo);
        } catch (WebServiceException ex) {
            throw ex;
        }

        return retBalance;
    }

    @WebMethod(operationName="updateAccountBalanceLock")
    public void updateAccountBalanceLock(@WebParam(name = "accountNo") String accountNo,
                                         @WebParam(name = "addBalance") BigDecimal addBalance) th

        AccountBrowserImpl accBrowser = new AccountBrowserImpl();

        try {
            accBrowser.updateAccountBalanceLock(accountNo, addBalance);
        } catch (WebServiceException ex) {
            throw ex;
        }

    }

}
```

Server	State	Status
IBM WASCE v2.1 Server a	Started	Synchronized
AccountWSEAR		Synchronized
AccountWS		Synchronized

As the last step we can confirm that the service can be found and is working. Open the Iceweasel web browser and enter following url:

http://localhost:8080/AccountWS/AccountBrowser

You should see a greeting from Axis2 Web service engine:


Hi, this is 'AccountService' web service.

Now the Web service is up and running.

NOTE: A known problem during the development of this test Web service was errors during the deployment process. Usually these errors can be solved by restarting the application server and trying to deploy again. Wait until the WASCE server is again in Started status and retry the Create Web Service process.

4.1.5 Create client project and test program

Now the Web service is ready and waiting for the tests. Next thing is to use the WSDL of the service to create needed Java artifacts for the client project. First create the Java classes from the service description and then a new client project that uses these generated classes.

1) Open a new terminal window and create a new directory structure under /home/student:

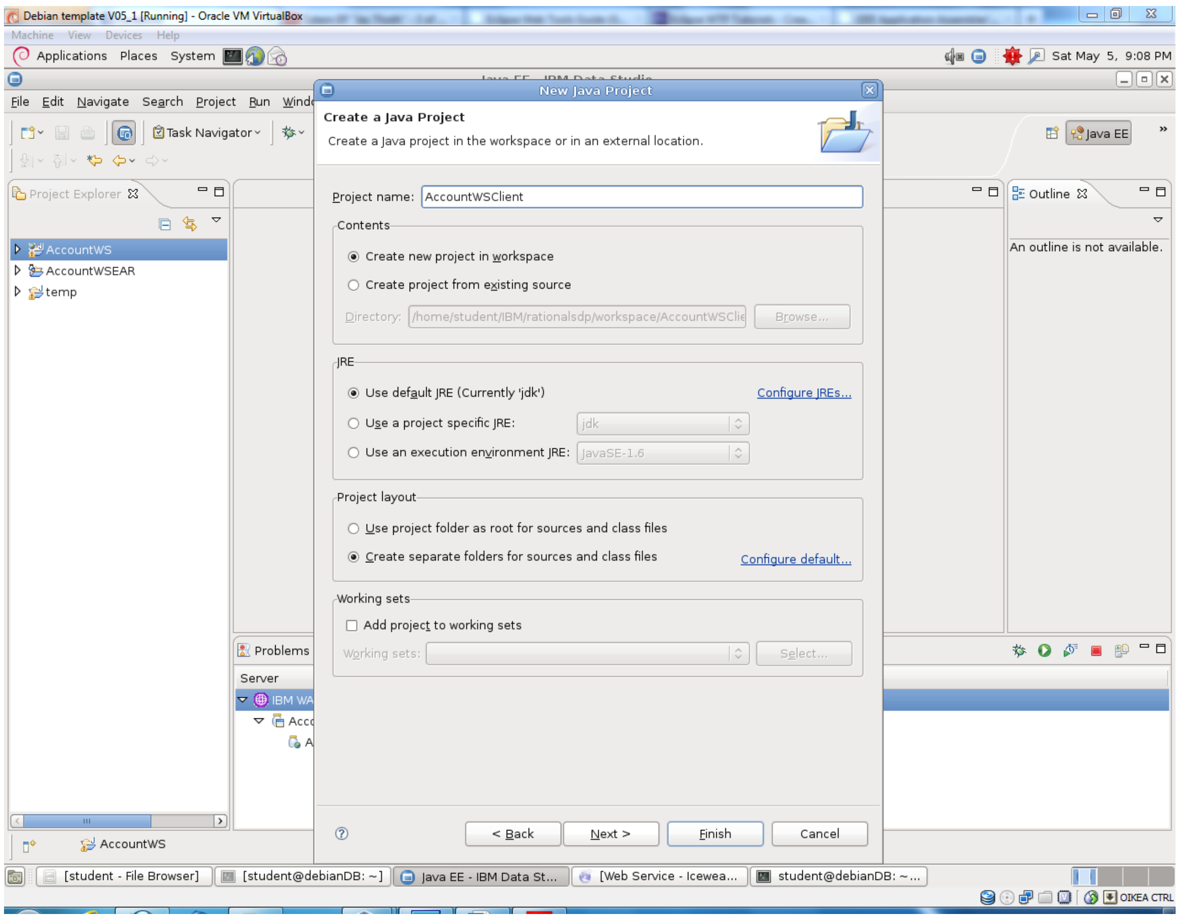
```
mkdir client
mkdir client/src
mkdir client/src/output
```

2) Go to client/src directory and run following command:

```
wsimport -d /home/student/client/src/output -p ws -s /home/student/client/src
http://localhost:8080/AccountWS/AccountBrowser?WSDL
```

Now there should be a new subdirectory client/src/ws and there some new, generated Java classes. These classes take care of the XML handling for our Java client program. These classes are imported next to a client project.

3) Open Data Studio and create a new Java project. File → New → Project → Java Project. Press Next. Enter a name for the client project: AccountWSClient. Accept the rest default values:



New Java Project

Create a Java Project

Create a Java project in the workspace or in an external location.

Project name: AccountWSClient

Contents

[✓ Create new project in workspace]
Create project from existing source
Directory: /home/student/IBM/rationalsdp/workspace/AccountWSClie

JRE

[✓ Use default JRE (Currently 'jdk')]
Use a project specific JRE: jdk
Use an execution environment JRE: JavaSE-1.6

Project layout

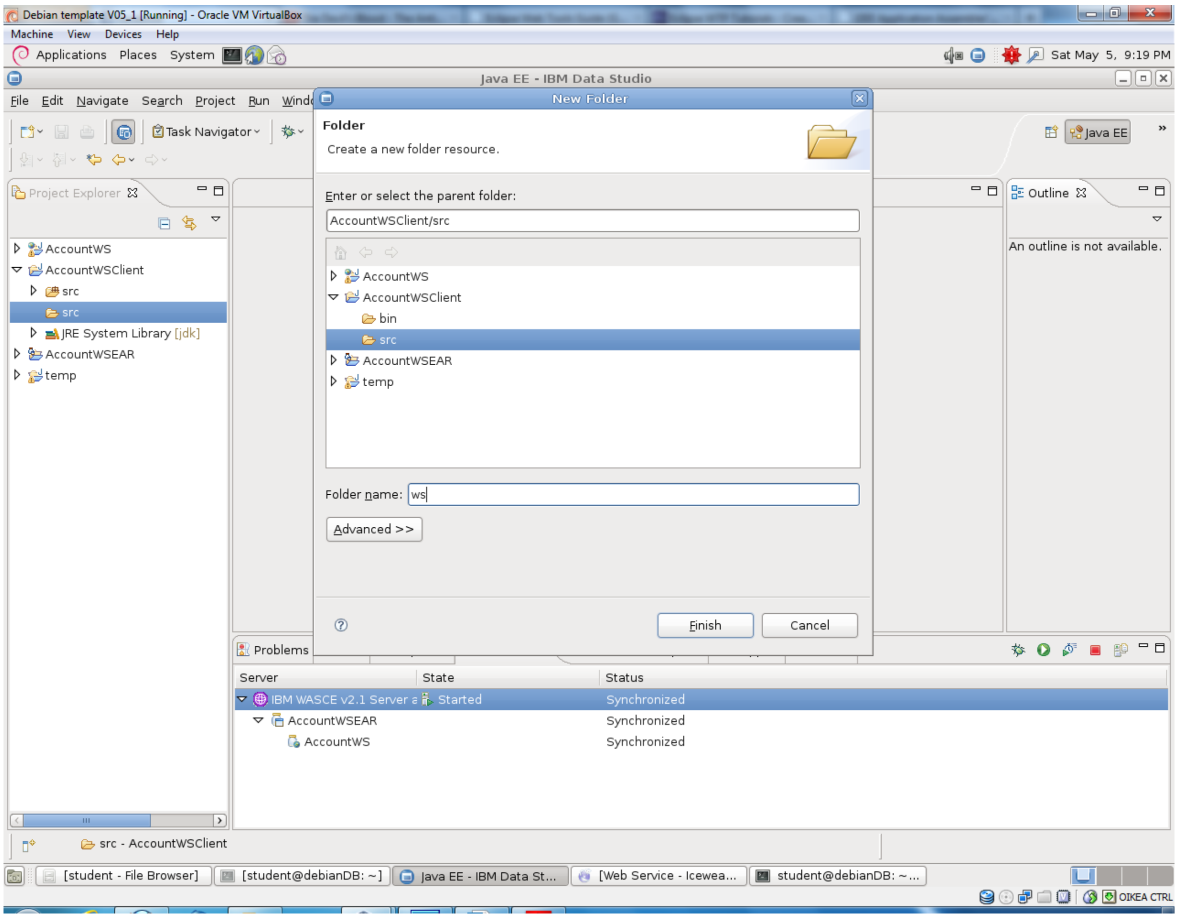
Use project folder as root for sources and class files
[✓ Create separate folders for sources and class files]

Working sets

☐ Add project to working sets
Working sets:

Press Next and Finish. And answer No to a question about changing the perspective. Now there is a new, empty project for our client classes.

4) Create a new folder under client source, called ws:



Folder

Create a new folder resource.

Enter or select the parent folder: AccountWSClient/src

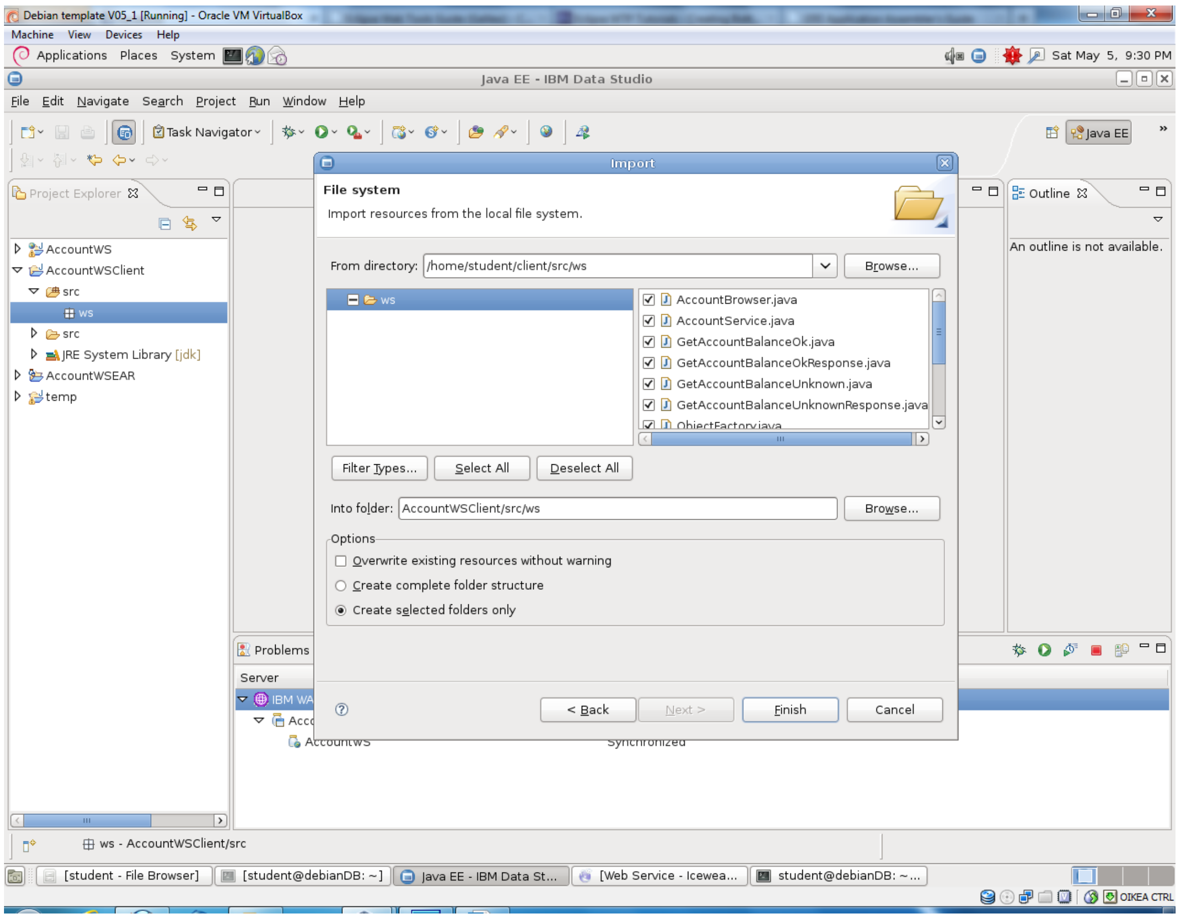
Folder name: ws

An outline is not available.

Server	State	Status
IBM WASCE v2.1 Server a	Started	Synchronized
AccountWSEAR		Synchronized
AccountWS		Synchronized

5) Choose the new subfolder ws, right-click and Import... Choose General → File System → Next.

6) Browse to the generated client classes /home/student/client/src and choose directory ws. Press OK. Choose all the Java classes from the right-side list and press Finish:


Import

File system

Import resources from the local file system.

From directory: /home/student/client/src/ws

- [✓] AccountBrowser.java
- [✓] AccountService.java
- [✓] GetAccountBalanceOk.java
- [✓] GetAccountBalanceOkResponse.java
- [✓] GetAccountBalanceUnknown.java
- [✓] GetAccountBalanceUnknownResponse.java
- [✓] ObjectFactory.java

Into folder: AccountWSClient/src/ws

Options
- [ ] Overwrite existing resources without warning
- ○ Create complete folder structure
- ● Create selected folders only

Now the generated Java artifacts should be in the client project under ws folder/package. The generated Java classes provide all the needed methods and classes to implement a Java client for the Web service.

7) Now add the final class to client project. Choose package/folder ws, right-click and New → Class. Use the same package ws and name TestClient. And finish.

8) Copy/paste following source code for the class content.

```
package ws;

import java.math.BigDecimal;

import javax.xml.ws.WebServiceRef;

public class TestClient {
    @WebServiceRef
    private static AccountService service;

    public static void main(String[] args) {

        BigDecimal result = new BigDecimal(0.0);
        BigDecimal addBalance = new BigDecimal(0.0);
        String accountNo;
        service = new AccountService();
        AccountBrowser accBrowser = service.getAccountServicePort();
        // First scenario 1 with correct account
        accountNo = "A10001";
        try {
            System.out.println("Call getAccountBalanceOk() with account " + accountNo);
            result = accBrowser.getAccountBalanceOk(accountNo);
            System.out.println("Result from getAccountBalanceOk() with account " + accountNo + ": " + result.toString());
        } catch (WebServiceException_Exception wex) {
            System.out.println("Fault from getAccountBalanceOk() account " + accountNo + ": " + wex.getMessage());
        } catch (Exception ex) {
            System.out.println("Other error from getAccountBalanceOk() account " + accountNo + ": " + ex.getMessage());
        }

        // Scenario 2 with incorrect account (not found)
        accountNo = "A10002";
        try {
            System.out.println("\nCall getAccountBalanceOk() with account " + accountNo);
            result = accBrowser.getAccountBalanceOk(accountNo);
            System.out.println("Result from getAccountBalanceOk() with account " + accountNo + ": " + result.toString());
        } catch (WebServiceException_Exception wex) {
            System.out.println("Fault from getAccountBalanceOk() account " + accountNo + ": " + wex.getMessage());
        } catch (Exception ex) {
            System.out.println("Other error from getAccountBalanceOk() account " + accountNo + ": " + ex.getMessage());
        }

        // Scenario 3 with incorrect object in query (account2 table, not exist)
        accountNo = "A10001";
        try {
            System.out.println("\nCall getAccountBalanceUnknown() with account " + accountNo);
            result = accBrowser.getAccountBalanceUnknown(accountNo);
            System.out.println("Result from getAccountBalanceUnknown() with account " + accountNo + ": " + re-
sult.toString());
        } catch (WebServiceException_Exception wex) {
            System.out.println("Fault from getAccountBalanceUnknown() account " + accountNo + ": " + wex.getMessage());
        } catch (Exception ex) {
            System.out.println("Other error from getAccountBalanceUnknown() account " + accountNo + ": " +
ex.getMessage());
        }

        // Scenario 4 with a locked record, only testing a lock situation, not really updating!
        accountNo = "A10001";
        addBalance = new BigDecimal(125.00);
        try {
            System.out.println("\nCall updateAccountBalanceLock() with account " + accountNo + " adding amount " + addBa-
lance.toString());
            accBrowser.updateAccountBalanceLock(accountNo, addBalance);
            System.out.println("Successful return from updateAccountBalanceLock()");
        } catch (WebServiceException_Exception wex) {
            System.out.println("Fault from updateAccountBalanceLock() account " + accountNo + " adding amount " + addBa-
lance.toString() + ": " + wex.getMessage());
        } catch (Exception ex) {
            System.out.println("Other error from updateAccountBalanceLock() account " + accountNo + " adding amount " +
addBalance.toString() + ": " + ex.getMessage());
        }

      }
}
```

This is now a test program to test our Web service scenarios. See more in the Results section. (Chong 2010, 131-136.)

4.2 Java/.NET

In this test case a .NET Web service is created and deployed to a web server. Then the service is called from a Java client program. The Web service is a bit more simple than in Java Web service case, the exceptions in the fault scenarios are just thrown out from service instead of implementing the database connection (ADO.NET) in reality. Otherwise this includes similar test cases like in Java-Java.

4.2.1 Mono setup and use

Mono offers a light-weight web server for this kind of test purposes. The server is called XSP and here are instructions how to set-up and use it as the container for Web service. There is already some installation of Mono in Debian virtual environment, but XSP needs to be installed.

1) Open terminal and login as root. Install Mono XSP by running this command:

```
student@debianDB:~$ su root
Password:
root@debianDB:/home/student# sudo apt-get install mono-xsp2 mono-xsp2-base asp.net2-
examples
```

This should install the needed server for .NET Web service use. Output is like this:

```
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  asp.net2-examples mono-xsp2 mono-xsp2-base
0 upgraded, 3 newly installed, 0 to remove and 149 not upgraded.
Need to get 277 kB of archives.
After this operation, 1,425 kB of additional disk space will be used.
Get:1 http://ftp.fi.debian.org/debian/ squeeze/main mono-xsp2-base all 2.6.5-3 [71.8 kB]
Get:2 http://ftp.fi.debian.org/debian/ squeeze/main mono-xsp2 all 2.6.5-3 [78.4 kB]
Get:3 http://ftp.fi.debian.org/debian/ squeeze/main asp.net2-examples all 2.6.5-3 [127 kB]
Fetched 277 kB in 0s (753 kB/s)
Preconfiguring packages ...
Selecting previously deselected package mono-xsp2-base.
(Reading database ... 140402 files and directories currently installed.)
Unpacking mono-xsp2-base (from .../mono-xsp2-base_2.6.5-3_all.deb) ...
Selecting previously deselected package mono-xsp2.
Unpacking mono-xsp2 (from .../mono-xsp2_2.6.5-3_all.deb) ...
Selecting previously deselected package asp.net2-examples.
```

```
Unpacking asp.net2-examples (from .../asp.net2-examples_2.6.5-3_all.deb) ...
Processing triggers for man-db ...
Setting up mono-xsp2-base (2.6.5-3) ...
Setting up mono-xsp2 (2.6.5-3) ...
Using Mono XSP 2 port: 8082
Binding Mono XSP 2 address: 0.0.0.0
Setting up asp.net2-examples (2.6.5-3) ...
root@debianDB:/home/student#
```

Now the rest can be done as student user.

2) Create a directory for the source code, for example /home/student/monoservice. Basically, the Web service source code can be saved in this directory and then from the same directory start the XSP server and deploy the service to server.

3) Go to /home/student/monoservice directory and create a new file AccountBrowser.asmx. Copy/paste following code as the file content and save:

```
<%@ WebService Language="C#" Class="AccountService.AccountBrowser" %>

using System;
using System.Web.Services;
using System.Web.Services.Protocols;
using System.Xml.Serialization;
using System.Xml;

namespace AccountService
{
    [WebService (Namespace = "http://ws/AccountBrowser")]
    public class AccountBrowser : WebService
    {
        [WebMethod]
        public decimal getAccountBalanceOk (String accountNo)
        {
            decimal balance = 0.0m;

            // In this test case account number A10001 returns a valid balance
            // from the service, but all other values throw an exception
            // (no data found). Simulated database error.

            if (accountNo == "A10001") {
                balance = 300.00m;
                return balance;
            } else {

                // Build the detail element of the SOAP fault.
                System.Xml.XmlDocument doc = new System.Xml.XmlDocument();
                System.Xml.XmlNode node = doc.CreateNode(XmlNodeType.Element,
                                          SoapException.DetailElementName.Name,
                                          SoapException.DetailElementName.Namespace);

                // Build specific details for the SoapException.
                // Add first child of detail XML element.
                System.Xml.XmlNode details =
                  doc.CreateNode(XmlNodeType.Element, "message", "http://ws/");

                // Append details to node
                node.AppendChild(details);
                //Throw the exception
                SoapException se = new SoapException("No data found",
                                       SoapException.ServerFaultCode,
                                       Context.Request.Url.AbsoluteUri ,
                                       node);

                throw se;
                return balance;
            }

        }

        [WebMethod]
        public decimal getAccountBalanceUnknown (String accountNo)
        {
            decimal balance = 0.0m;

            // In this test case a query to an unknown database object is simulated.
            // As a result an SQL error is returned out from the service.

            // Build the detail element of the SOAP fault.
            System.Xml.XmlDocument doc = new System.Xml.XmlDocument();
            System.Xml.XmlNode node = doc.CreateNode(XmlNodeType.Element,
                                      SoapException.DetailElementName.Name,
                                      SoapException.DetailElementName.Namespace);

            // Build specific details for the SoapException.
            // Add first child of detail XML element.
            System.Xml.XmlNode details =
              doc.CreateNode(XmlNodeType.Element, "message", "http://ws/");

            // Append details to node
            node.AppendChild(details);

            //Throw the exception
            SoapException se = new SoapException("SQL error",
                                   SoapException.ServerFaultCode,
                                   Context.Request.Url.AbsoluteUri ,
                                   node);

            throw se;
            return balance;

        }

        [WebMethod]
        public void updateAccountBalanceLock (String accountNo, decimal addBalance)
        {

            // In this test case a database lock situation is simulated. Another session
            // has reserved a record and an SQL error is returned from the service.

            // Build the detail element of the SOAP fault.
            System.Xml.XmlDocument doc = new System.Xml.XmlDocument();
            System.Xml.XmlNode node = doc.CreateNode(XmlNodeType.Element,
                                      SoapException.DetailElementName.Name,
                                      SoapException.DetailElementName.Namespace);

            // Build specific details for the SoapException.
            // Add first child of detail XML element.
            System.Xml.XmlNode details =
              doc.CreateNode(XmlNodeType.Element, "message",
             "http://ws/");

            //details.setValue("Record locked for another session");

            // Append details to node
            node.AppendChild(details);

            //Throw the exception
```

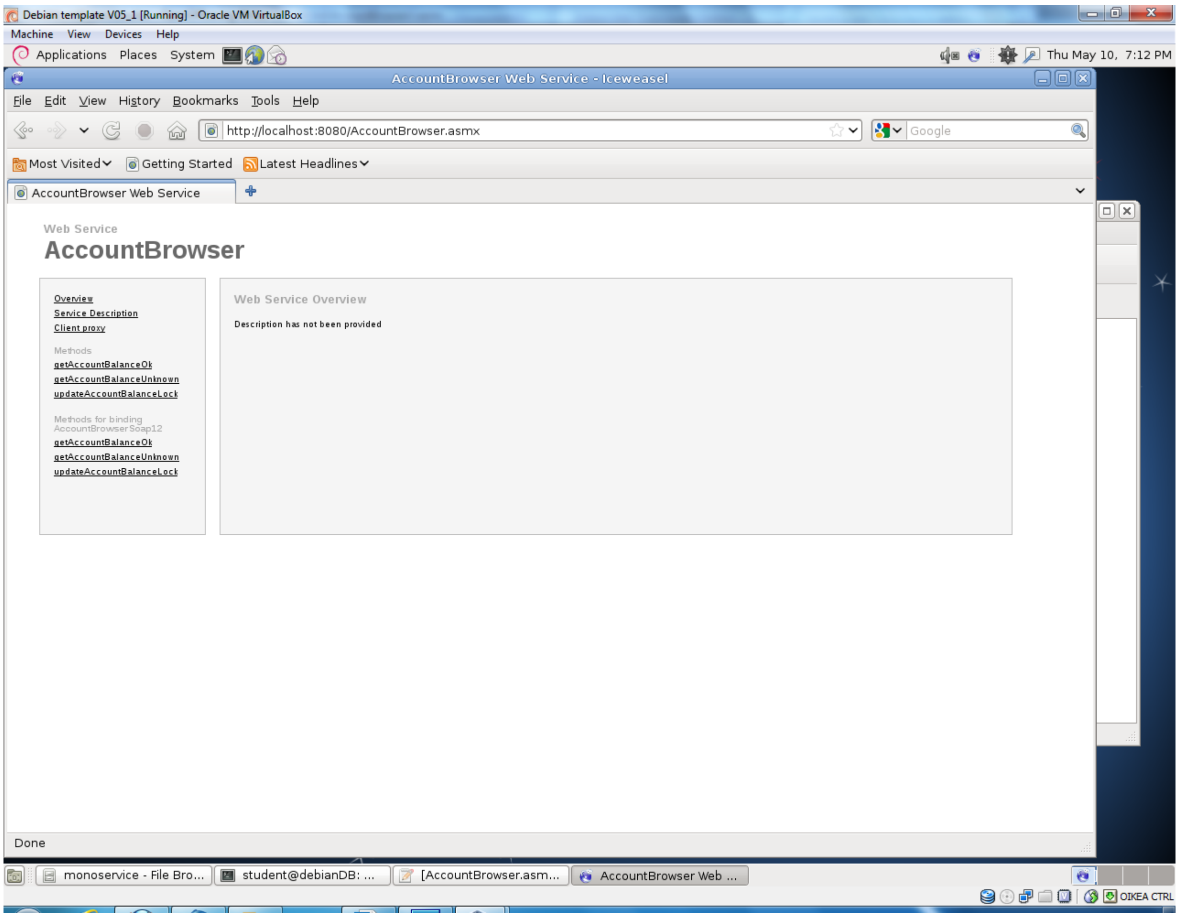
```
            SoapException se = new SoapException("Record locked",
                                   SoapException.ServerFaultCode,
                                   Context.Request.Url.AbsoluteUri ,
                                   node);

            throw se;

        }

    }
}

(Microsoft 2012. Handling and Throwing Exceptions in XML Web Services.)
```

4) From the same directory run command "xsp2" and the server will start and run as long as you don't press enter again in the same terminal window. This will also make the new AccountBrowser.asmx available for use on server.

5) Open Iceweasel internet browser and enter following url:

http://localhost:8080/AccountBrowser.asmx

If there are no compilation errors or other problems, you should see the Web service page like this:



Description has not been provided

If this page is shown then the .NET Web service is up and running.

The XSP server stops when you press enter in the terminal window where you started the service. (Mono Project 2012. Writing a Web service.)

4.2.2 Create a Java client to consume the .NET Web service

Now the .NET Web service is up and running and we can create a Java test client for the service. This is done in the same way as the client project was done in Java-Java part. Make sure that XSP and the Web service is up and running when start creating the client.

1) Open new terminal and create new directories under /home/student/client.

```
mkdir mono
mkdir mono/src
mkdir mono/src/output
```

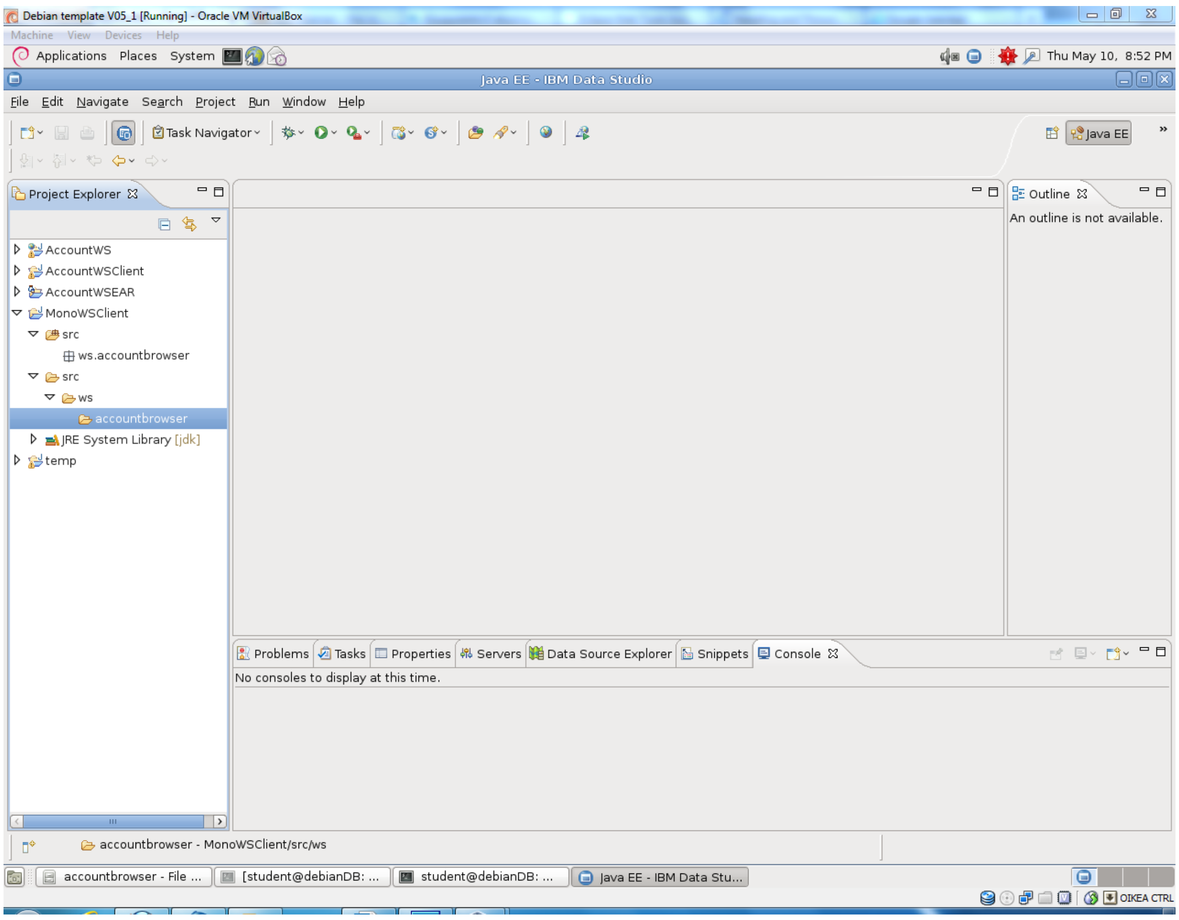
2) Go to directory /home/student/client/mono/src and issue command:

```
wsimport -d /home/student/client/mono/src/output -s /home/student/client/mono/src
http://localhost:8080/AccountBrowser.asmx?wsdl
```

This should now create needed Java artifacts for a client project that accesses the new .NET Web service. Don't worry about the warnings about some non-standard parts in WSDL, we only need the one version that IS created under mono/src/ws/accountbrowser.

3) Open IBM Data Studio and choose New → Project → Java Project. Press Next. Enter project name MonoWSClient and accept the rest of default values by pressing Finish. Choose No when asking about change of Perspective.

4) Open the MonoWSClient project and choose src folder. Create new folder structure (New → Folder) src/ws/accountbrowser. Now it should look like this:


Debian template V05_1 [Running] - Oracle VM VirtualBox
Machine View Devices Help
Applications Places System
Thu May 10, 8:52 PM
Java EE - IBM Data Studio
File Edit Navigate Search Project Run Window Help
Task Navigator
Java EE
Project Explorer
Outline
Problems Tasks Properties Servers Data Source Explorer Snippets Console
accountbrowser - File ... [student@debianDB: ... student@debianDB: ... Java EE - IBM Data Stu...

- AccountWS
- AccountWSClient
- AccountWSEAR
- MonoWSClient
  - src
    - ws.accountbrowser
  - src
    - ws
      - accountbrowser
  - JRE System Library [jdk]
- temp

An outline is not available.

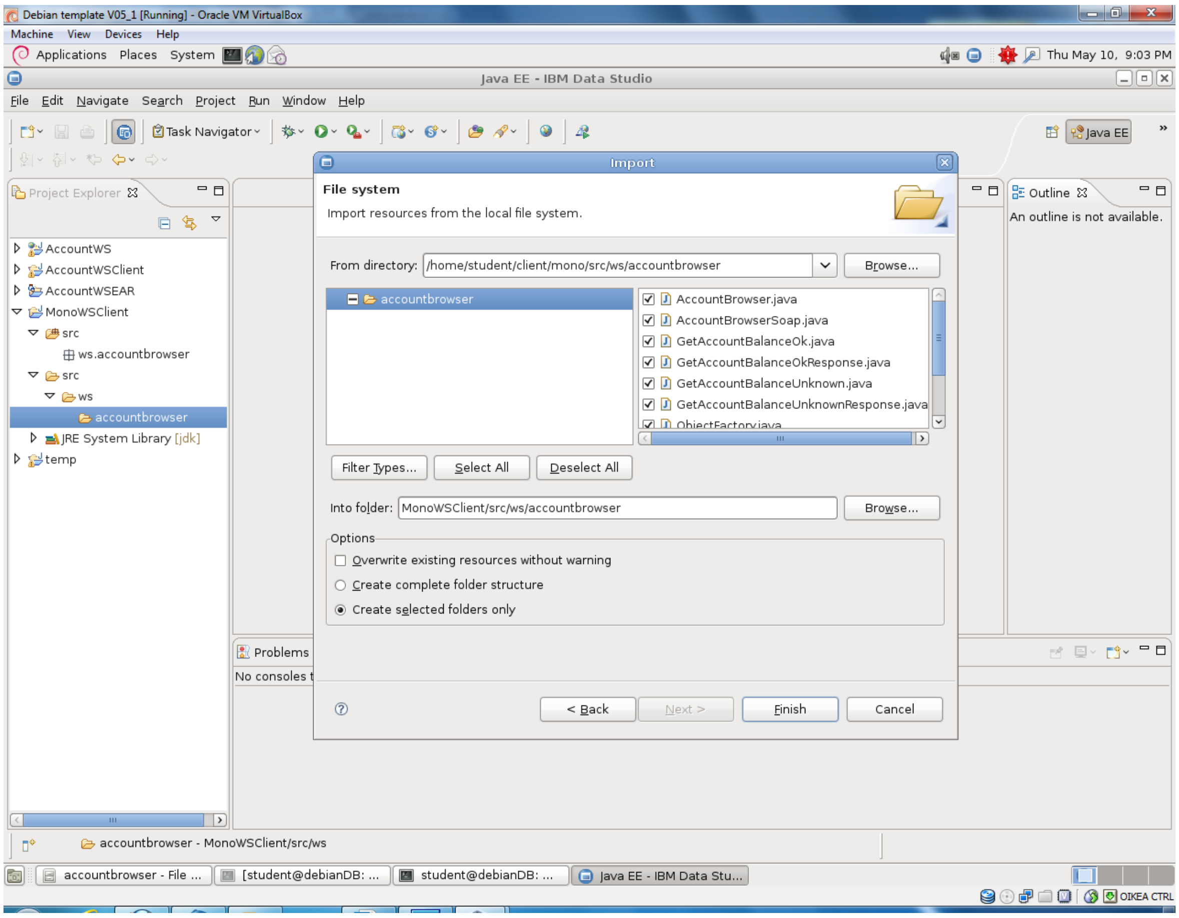
No consoles to display at this time.

accountbrowser - MonoWSClient/src/ws

5) Keep folder accountbrowser selected, right-click and choose Import → File system → Next.

6) Browse to /home/student/client/mono/src/ws/ and choose accountbrowser directory → OK.

7) Choose all the individual classes from the right-side list and press Finish:


Import

File system

Import resources from the local file system.

From directory: /home/student/client/mono/src/ws/accountbrowser

accountbrowser

- [✓] AccountBrowser.java
- [✓] AccountBrowserSoap.java
- [✓] GetAccountBalanceOk.java
- [✓] GetAccountBalanceOkResponse.java
- [✓] GetAccountBalanceUnknown.java
- [✓] GetAccountBalanceUnknownResponse.java
- [✓] ObjectFactory.java

Into folder: MonoWSClient/src/ws/accountbrowser

Options
- [ ] Overwrite existing resources without warning
- ○ Create complete folder structure
- ● Create selected folders only

Now the generated artifacts are added to our project.

8) Add another Java class under same directory (/src/ws/accountbrowser), called TestClient. Copy/paste following code into class file:

```
package ws.accountbrowser;

import java.math.BigDecimal;

import javax.xml.soap.SOAPException;
import javax.xml.ws.WebServiceRef;

public class TestClient {
    @WebServiceRef
    private static AccountBrowser service;
    public static void main(String[] args) {

        BigDecimal result = new BigDecimal(0.0);
        BigDecimal addBalance = new BigDecimal(0.0);
        String accountNo;
        service = new AccountBrowser();
        AccountBrowserSoap accBrowser = service.getAccountBrowserSoap();

        // First scenario 1 with correct account
        accountNo = "A10001";
        try {
            System.out.println("Call getAccountBalanceOk() with account " + accountNo);
            result = accBrowser.getAccountBalanceOk(accountNo);
            System.out.println("Result from getAccountBalanceOk() with account " + accountNo + ": " + result.toString());
        } catch (Exception ex) {
            System.out.println("Other error from getAccountBalanceOk() account " + accountNo + ": " + ex.getMessage());
        }

        // Scenario 2 with incorrect account (not found)
        accountNo = "A10002";
        try {
            System.out.println("\nCall getAccountBalanceOk() with account " + accountNo);
            result = accBrowser.getAccountBalanceOk(accountNo);
            System.out.println("Result from getAccountBalanceOk() with account " + accountNo + ": " + result.toString());
        } catch (Exception ex) {
            System.out.println("Other error from getAccountBalanceOk() account " + accountNo + ": " + ex.getMessage());
        }

        // Scenario 3 with incorrect object in query (account2 table, not exist)
        accountNo = "A10001";
        try {
            System.out.println("\nCall getAccountBalanceUnknown() with account " + accountNo);
            result = accBrowser.getAccountBalanceUnknown(accountNo);
            System.out.println("Result from getAccountBalanceUnknown() with account " + accountNo + ": " + re-
sult.toString());
        } catch (Exception ex) {
            System.out.println("Other error from getAccountBalanceUnknown() account " + accountNo + ": " +
ex.getMessage());
        }

        // Scenario 4 with a locked record, only testing a lock situation, not really updating!
        accountNo = "A10001";
        addBalance = new BigDecimal(125.00);
        try {
            System.out.println("\nCall updateAccountBalanceLock() with account " + accountNo + " adding amount " + addBa-
lance.toString());
            accBrowser.updateAccountBalanceLock(accountNo, addBalance);
            System.out.println("Successful return from updateAccountBalanceLock()");
        } catch (Exception ex) {
            System.out.println("Other error from updateAccountBalanceLock() account " + accountNo + " adding amount " +
addBalance.toString() + ": " + ex.getMessage());
        }

    }

}
```

This is now a test program to test our Web service scenarios. See more in the Results section.

4.3 .NET/Java

Use the Java Web service created during the Java-Java part and create a .NET client program to access the service. Mono offers the wsdl command that reads the Web service description from the running service or from a file.

4.3.1 Generate .NET artifacts from WSDL

1) Open new terminal session and create a new directory /home/student/monoclient.

2) Make sure that the Java Web service is up and running on WASCE server.

3) Go to the new directory monoclient and enter following command in terminal:

```
wsdl http://localhost:8080/AccountWS/AccountBrowser?WSDL
```

This command accesses the Web service description (WSDL) and should create the needed C# artifacts for a local stub. In this case the generation is done with a warning:

```
student@debianDB:~/monoclient$ wsdl http://localhost:8080/AccountWS/AccountBrowser?WSDL
Web Services Description Language Utility
Mono Framework v2.0.50727.1433

There where some warnings while generating the code:

  http://localhost:8080/AccountWS/AccountBrowser?WSDL
    - This web reference does not conform to WS-I Basic Profile v1.1
        R2718: A wsdl:binding in a DESCRIPTION MUST have the same set of
        wsdl:operations as the wsdl:portType to which it refers.
          * Binding 'AccountServicePortBinding', in Service Description
            'http://ws'

Writing file 'AccountService.cs'
```

4) Now a C# source code is created in the directory, but with warnings. Next step is to try to compile this program:

```
student@debianDB:~/monoclient$ mcs /target:library AccountService.cs -r:System.Web.Services
```

The compilation should produce a usable local stub for client program use, but the result is a compilation error:

```
AccountService.cs(151,84): error CS0234: The type or namespace name `AsyncCompletedEven-
tArgs' does not exist in the namespace `System.ComponentModel'. Are you missing an assembly
reference?
AccountService.cs(170,89): error CS0234: The type or namespace name `AsyncCompletedEven-
tArgs' does not exist in the namespace `System.ComponentModel'. Are you missing an assembly
reference?
Compilation failed: 2 error(s), 0 warnings
student@debianDB:~/monoclient$
```

The use of Mono library program wsdl fails to understand the Java Web service description and the local stub (AccountService.dll) can not be created. (Mono Project 2012. Consuming a Web service.)

This result means that the WSDL documents are not always 100% perfect and may need some manual adjustment. In this case the Mono wsdl command does not understand the default Web service description from the Java Web service deployed on WASCE. This test case is left for possible further investigation in the future and also acts as an example of a non-compatible Web service description or implementation by default. In Internet forums there are several notes about the same issue with Mono parsing a WSDL and some need for altering the description somehow (Stackoverflow.com 2011. Mono wsdl tool fails to parse salesforce enterprise.wsdl).

4.3.2 Create a .NET client program

In this test case there was no need to create a client program now because the needed Web service stub could not be created successfully.

5 Results

Here are the test results from each scenario and test run.

5.1 Java/Java

The Java test client is run two times to show the different result in the last of the included test scenarios in the program (updateAccountBalanceLock). The SOAP operations / Java methods tested are following ones in this order:

1) getAccountBalanceOk() with a valid account number (A10001) that exists in the database (test runs 1 and 2)

2) getAccountBalanceOk() with an invalid account number (A10002), data not found in database (test runs 1 and 2)

3) getAccountBalanceUnknown() with any account number, resulting in an SQL error due to non-existing database object (test runs 1 and 2)

4) updateAccountBalanceLock() first without the record locked, resulting in a successful response from service (Test run 1 only)

5) updateAccountBalanceLock() with a database lock acquired by another session, resulting in an SQL error from the lock situation (Test run 2 only)

5.1.1 Plain TestClient run without database lock

In Data Studio, choose TestClient class, right-click and Run As → Java Application. Following results are printed out on Data Studio console, call and result from each Web service call in our TestClient program.

```
Call getAccountBalanceOk() with account A10001
Result from getAccountBalanceOk() with account A10001: 300

Call getAccountBalanceOk() with account A10002
Fault from getAccountBalanceOk() account A10002: SQL Exception 0 No data found

Call getAccountBalanceUnknown() with account A10001
Fault from getAccountBalanceUnknown() account A10001: SQL Exception 942 ORA-00942: table or
view does not exist

Call updateAccountBalanceLock() with account A10001 adding amount 125
```

```
Successful return from updateAccountBalanceLock()
```

The SOAP requests and responses are shown here in the same order from the same Web service operations:

```
Request:

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ws="http://ws">
   <soapenv:Header/>
   <soapenv:Body>
      <ws:getAccountBalanceOk>
         <!--Optional:-->
         <accountNo>A10001</accountNo>
      </ws:getAccountBalanceOk>
   </soapenv:Body>
</soapenv:Envelope>

Response (successful):

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
   <soapenv:Body>
      <ns2:getAccountBalanceOkResponse xmlns:ns2="http://ws">
         <return>300</return>
      </ns2:getAccountBalanceOkResponse>
   </soapenv:Body>
</soapenv:Envelope>

Request:

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ws="http://ws">
   <soapenv:Header/>
   <soapenv:Body>
      <ws:getAccountBalanceOk>
         <!--Optional:-->
         <accountNo>A10002</accountNo>
      </ws:getAccountBalanceOk>
   </soapenv:Body>
</soapenv:Envelope>

Response (fault, no data found):

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
   <soapenv:Body>
      <soapenv:Fault>
         <faultcode>soapenv:Server</faultcode>
         <faultstring>SQL Exception 0 No data found</faultstring>
         <detail>
            <ns2:WebServiceException xmlns:ns2="http://ws">
               <message>SQL Exception 0 No data found</message>
            </ns2:WebServiceException>
         </detail>
      </soapenv:Fault>
   </soapenv:Body>
</soapenv:Envelope>
```

Request:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ws="http://ws">
   <soapenv:Header/>
   <soapenv:Body>
      <ws:getAccountBalanceUnknown>
         <!--Optional:-->
         <accountNo>A10001</accountNo>
      </ws:getAccountBalanceUnknown>
   </soapenv:Body>
</soapenv:Envelope>
```

Response (fault, account2 table not exist):

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
   <soapenv:Body>
      <soapenv:Fault>
         <faultcode>soapenv:Server</faultcode>
         <faultstring>SQL Exception 942 ORA-00942: table or view does not    ex-
ist</faultstring>
         <detail>
            <ns2:WebServiceException xmlns:ns2="http://ws">
               <message>SQL Exception 942 ORA-00942: table or view does not exist</message>
            </ns2:WebServiceException>
         </detail>
      </soapenv:Fault>
   </soapenv:Body>
</soapenv:Envelope>
```

Request:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ws="http://ws">
   <soapenv:Header/>
   <soapenv:Body>
      <ws:updateAccountBalanceLock>
         <!--Optional:-->
         <accountNo>A10001</accountNo>
         <!--Optional:-->
         <addBalance>75.00</addBalance>
      </ws:updateAccountBalanceLock>
   </soapenv:Body>
</soapenv:Envelope>
```

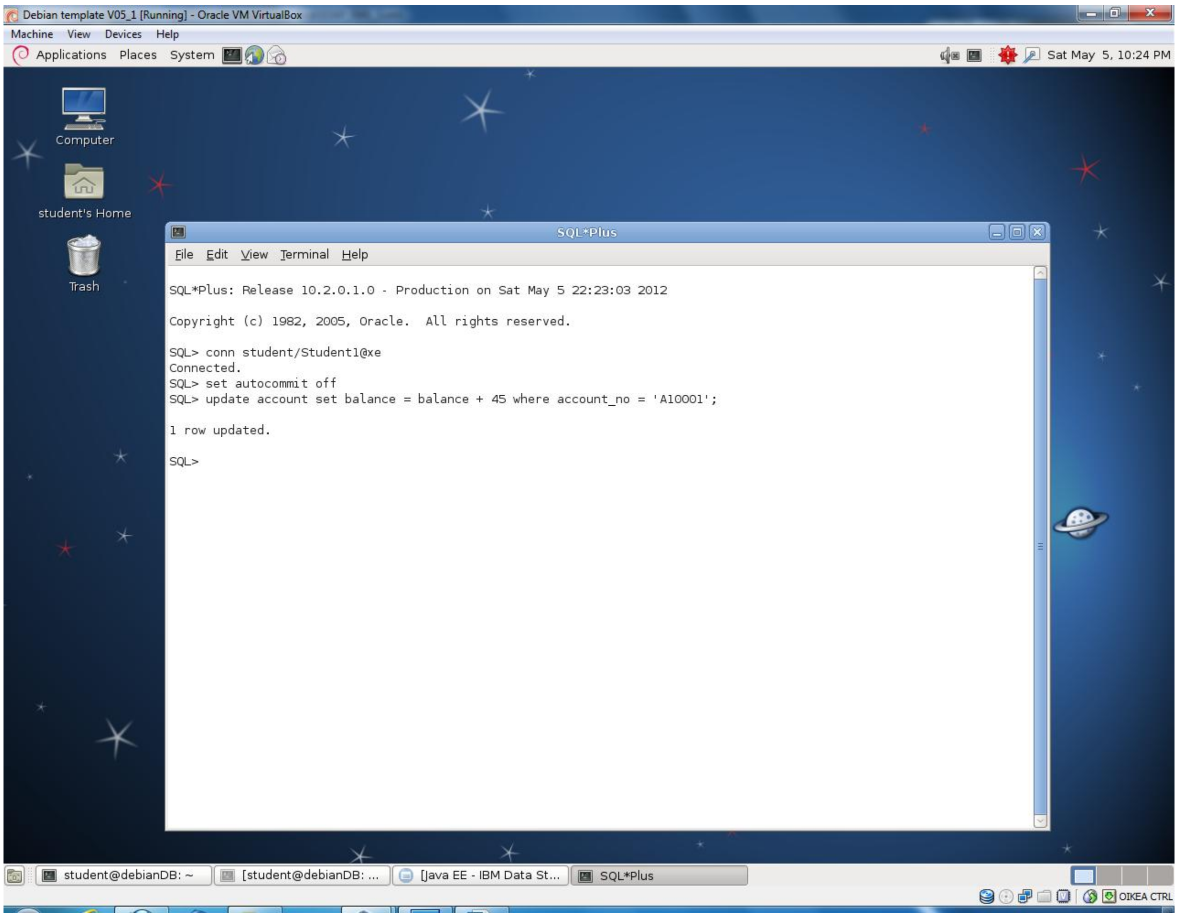
Response (successful, return type is void):

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
   <soapenv:Body>
      <ns2:updateAccountBalanceLockResponse xmlns:ns2="http://ws"/>
   </soapenv:Body>
</soapenv:Envelope>
```

5.1.2 TestClient run with a database lock

This is otherwise same TestClient run, but first a lock situation is created in the database.

1) Open Applications → Oracle Database 10g Express Edition → Run SQL Command Line. Enter following commands to acquire a lock in the database table:


Debian template V05_1 [Running] - Oracle VM VirtualBox
Machine View Devices Help
Applications Places System
Sat May 5, 10:24 PM
Computer
student's Home
Trash
SQL*Plus
File Edit View Terminal Help
student@debianDB: ~
[student@debianDB: ...
[Java EE - IBM Data St...
SQL*Plus

```
SQL*Plus: Release 10.2.0.1.0 - Production on Sat May 5 22:23:03 2012

Copyright (c) 1982, 2005, Oracle.  All rights reserved.

SQL> conn student/Student1@xe
Connected.
SQL> set autocommit off
SQL> update account set balance = balance + 45 where account_no = 'A10001';

1 row updated.

SQL>
```

Do NOT enter COMMIT or ROLLBACK before running the Java TestClient now.

2) Run the TestClient now again and the results are following. Otherwise the same, but the last operation returns fault because of the lock situation. After running the TestClient you can issue ROLLBACK in SQL window.

```
Call getAccountBalanceOk() with account A10001
Result from getAccountBalanceOk() with account A10001: 300

Call getAccountBalanceOk() with account A10002
Fault from getAccountBalanceOk() account A10002: SQL Exception 0 No data found

Call getAccountBalanceUnknown() with account A10001
Fault from getAccountBalanceUnknown() account A10001: SQL Exception 942 ORA-00942: table or
view does not exist

Call updateAccountBalanceLock() with account A10001 adding amount 125
Fault from updateAccountBalanceLock() account A10001 adding amount 125:
SQL Exception 54 ORA-00054: resource busy and acquire with NOWAIT specified
```

The SOAP requests and responses are shown here in the same order from the same Web service operations:

```
Request:

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ws="http://ws">
   <soapenv:Header/>
   <soapenv:Body>
      <ws:getAccountBalanceOk>
         <!--Optional:-->
         <accountNo>A10001</accountNo>
      </ws:getAccountBalanceOk>
   </soapenv:Body>
</soapenv:Envelope>

Response (successful):

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
   <soapenv:Body>
      <ns2:getAccountBalanceOkResponse xmlns:ns2="http://ws">
         <return>300</return>
      </ns2:getAccountBalanceOkResponse>
   </soapenv:Body>
</soapenv:Envelope>

Request:

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ws="http://ws">
   <soapenv:Header/>
   <soapenv:Body>
```

```
      <ws:getAccountBalanceOk>
         <!--Optional:-->
         <accountNo>A10002</accountNo>
      </ws:getAccountBalanceOk>
   </soapenv:Body>
</soapenv:Envelope>
```

```
Response (fault, no data found):
```

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
   <soapenv:Body>
      <soapenv:Fault>
         <faultcode>soapenv:Server</faultcode>
         <faultstring>SQL Exception 0 No data found</faultstring>
         <detail>
            <ns2:WebServiceException xmlns:ns2="http://ws">
               <message>SQL Exception 0 No data found</message>
            </ns2:WebServiceException>
         </detail>
      </soapenv:Fault>
   </soapenv:Body>
</soapenv:Envelope>
```

```
Request:
```

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ws="http://ws">
   <soapenv:Header/>
   <soapenv:Body>
      <ws:getAccountBalanceUnknown>
         <!--Optional:-->
         <accountNo>A10001</accountNo>
      </ws:getAccountBalanceUnknown>
   </soapenv:Body>
</soapenv:Envelope>
```

```
Response (fault, account2 table not exist):
```

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
   <soapenv:Body>
      <soapenv:Fault>
         <faultcode>soapenv:Server</faultcode>
         <faultstring>SQL Exception 942 ORA-00942: table or view does not    ex-
ist</faultstring>
         <detail>
            <ns2:WebServiceException xmlns:ns2="http://ws">
               <message>SQL Exception 942 ORA-00942: table or view does not exist</message>
            </ns2:WebServiceException>
         </detail>
      </soapenv:Fault>
   </soapenv:Body>
</soapenv:Envelope>
Request:
```

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ws="http://ws">
   <soapenv:Header/>
   <soapenv:Body>
      <ws:updateAccountBalanceLock>
         <!--Optional:-->
         <accountNo>A10001</accountNo>
         <!--Optional:-->
         <addBalance>75.00</addBalance>
      </ws:updateAccountBalanceLock>
   </soapenv:Body>
</soapenv:Envelope>

Response (fault, record is locked by another session now):

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
   <soapenv:Body>
      <soapenv:Fault>
         <faultcode>soapenv:Server</faultcode>
         <faultstring>SQL Exception 54 ORA-00054: resource busy and acquire with NOWAIT
specified</faultstring>
         <detail>
            <ns2:WebServiceException xmlns:ns2="http://ws">
               <message>SQL Exception 54 ORA-00054: resource busy and acquire with NOWAIT
specified</message>
            </ns2:WebServiceException>
         </detail>
      </soapenv:Fault>
   </soapenv:Body>
</soapenv:Envelope>
```

NOTE: A known issue with this configuration is that if the UPDATE statement in the database console seems to halt and take forever, you have to cancel and do the following and then continue this test run: 1. Shutdown WASCE, 2. shutdown database. 3. Start database again and 4. start WASCE again. Now it should work again. Don't know why this issue happens, maybe there is some connection state or limit somewhere that causes the problem.

5.2 Java/.NET

Use Java test client to test the .NET Web service. The Java test client is run once to show the results on console. The SOAP operations / Java methods tested are following ones in this order:

1) getAccountBalanceOk() with a valid account number (A10001) that exists in the database

2) getAccountBalanceOk() with an invalid account number (A10002), data not found in database

3) getAccountBalanceUnknown() with any account number, resulting in an SQL error due to non-existing database object

4) updateAccountBalanceLock() with a database lock acquired by another session, resulting in an SQL error from the lock situation

5.2.1 Test client run

In Data Studio, choose TestClient class, right-click and Run As → Java Application. Following results are printed out on Data Studio console, call and result from each Web service call in our TestClient program.

```
Call getAccountBalanceOk() with account A10001
Result from getAccountBalanceOk() with account A10001: 300.00

Call getAccountBalanceOk() with account A10002
Other error from getAccountBalanceOk() account A10002: No data found

Call getAccountBalanceUnknown() with account A10001
Other error from getAccountBalanceUnknown() account A10001: SQL error

Call updateAccountBalanceLock() with account A10001 adding amount 125
Other error from updateAccountBalanceLock() account A10001 adding amount 125: Record locked
```

The SOAP requests and responses are shown here in the same order from the same Web service operations:

```
Request (with valid account number):

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:acc="http://tempuri.org/AccountBrowser">
   <soapenv:Header/>
   <soapenv:Body>
      <acc1:getAccountBalanceOk xmlns:acc1="http://ws/AccountBrowser">
         <!--Optional:-->
         <acc1:accountNo>A10001</acc1:accountNo>
      </acc1:getAccountBalanceOk>
   </soapenv:Body>
</soapenv:Envelope>
```

Response:

```
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>
      <getAccountBalanceOkResponse xmlns="http://ws/AccountBrowser">
         <getAccountBalanceOkResult>300.00</getAccountBalanceOkResult>
      </getAccountBalanceOkResponse>
   </soap:Body>
</soap:Envelope>
```

Request (with non-existing account number):

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:acc="http://tempuri.org/AccountBrowser">
   <soapenv:Header/>
   <soapenv:Body>
      <acc1:getAccountBalanceOk xmlns:acc1="http://ws/AccountBrowser">
         <!--Optional:-->
         <acc1:accountNo>A10002</acc1:accountNo>
      </acc1:getAccountBalanceOk>
   </soapenv:Body>
</soapenv:Envelope>
```

Response:

```
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>
      <soap:Fault>
         <faultcode>soap:Server</faultcode>
         <faultstring>No data found</faultstring>
         <faultactor>http://localhost:8080/AccountBrowser.asmx</faultactor>
         <detail>
            <detail>
               <message xmlns="http://ws"/>
            </detail>
         </detail>
      </soap:Fault>
   </soap:Body>
</soap:Envelope>
```

Request:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:acc="http://tempuri.org/AccountBrowser">
```

```
   <soapenv:Header/>
   <soapenv:Body>
      <acc1:getAccountBalanceUnknown xmlns:acc1="http://ws/AccountBrowser">
         <!--Optional:-->
         <acc1:accountNo>A10001</acc1:accountNo>
      </acc1:getAccountBalanceUnknown>
   </soapenv:Body>
</soapenv:Envelope>
```

Response:

```
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>
      <soap:Fault>
         <faultcode>soap:Server</faultcode>
         <faultstring>SQL error</faultstring>
         <faultactor>http://localhost:8080/AccountBrowser.asmx</faultactor>
         <detail>
            <detail>
               <message xmlns="http://ws"/>
            </detail>
         </detail>
      </soap:Fault>
   </soap:Body>
</soap:Envelope>
```

Request:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:acc="http://tempuri.org/AccountBrowser">
   <soapenv:Header/>
   <soapenv:Body>
      <acc1:updateAccountBalanceLock xmlns:acc1="http://ws/AccountBrowser">
         <!--Optional:-->
         <acc1:accountNo>A10001</acc1:accountNo>
         <acc1:addBalance>45</acc1:addBalance>
      </acc1:updateAccountBalanceLock>
   </soapenv:Body>
</soapenv:Envelope>
```

Response:

```
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>
      <soap:Fault>
         <faultcode>soap:Server</faultcode>
         <faultstring>Record locked</faultstring>
         <faultactor>http://localhost:8080/AccountBrowser.asmx</faultactor>
         <detail>
```

```
            <detail>
               <message xmlns="http://ws"/>
            </detail>
         </detail>
      </soap:Fault>
   </soap:Body>
</soap:Envelope>
```

5.3 .NET/Java

In this study we were not able to produce a working .NET client out from an existing Java Web service. No test results from this scenario.

6 Conclusions

This thesis work and study shows that there are some differences in the Web services standards implementations depending on the product manufacturer or contributor. All the basic standard functionality has been implemented, but there are some flavors too. It is interesting to create same test cases with different tools and check the results. This time a part of the tests went like expected, only one part failed with default values.

The scenarios and set ups in this study are definitely not the only options, but these were chosen to demonstrate the idea of Web services in practice. Same kind of scenarios could be done in other environments too. For example, one could implement the same idea with DB2 database and services. If the DBMS provides the tools to create Web services based on existing procedures etc., it could be also tested in the same manner.

The .NET Web service created in this work could be enhanced with the real database connection and functionality (ADO.NET). And the Java Web service could be further developed and try custom FaultBean classes and get them working and return a good fault response from the service.

The problematic .NET client generation with Mono also deserves a second look, but it may require some manual work with the Java Web service WSDL file. Or maybe the future releases of Mono/wsdl library can understand the service description made with Java during this study.

During this study there has been valuable test and use of the chosen tools and technologies in Debian virtual laboratory: Web services with IBM WASCE and Mono XSP servers and Oracle database use with WASCE etc. This can further lead to new kinds of usage of these tools in this learning environment.

References

Chong, R., Hua, C., Lei, W., Martins, J., Ming X., Quan J., Xuan D., Ying T. 2010. Getting started with WebSphere Application Server Community Edition. 1st edition. DB2 on Campus series. IBM, Ontario, Canada.

Hansen, Mark D. 2007. SOA Using Java Web Services. Prentice Hall, NJ, United States.

Linuxtopia. 2012. Eclipse Web Tools Guide. Creating Web services with the Apache Axis2 runtime environments. Readable at: http://www.linuxtopia.org/online_books/eclipse_documentation/eclipse_web_tools_platform_guide/topic/org.eclipse.jst.ws.axis2.ui.doc.user/topics/eclipse_web_tools_caxis2tover.html. Accessed: 19.4.2012.

Microsoft. 2012. Handling and Throwing Exceptions in XML Web Services. Readable at: http://msdn.microsoft.com/en-us/library/ds492xtk%28v=vs.71%29.aspx. Accessed: 9.5.2012.

Mono Project. 2012. Consuming a Web service. Readable at: http://mono-project.com/Consuming_a_WebService. Accessed: 10.5.2012.

Mono Project. 2012. Writing a Web service. Readable at: http://www.mono-project.com/Writing_a_WebService. Accessed: 10.5.2012.

W3C 2000. Web services architecture. Readable at: http://www.w3.org/TR/ws-arch/. Accessed: 19.9.2011.

W3C 2000. Simple Object Access Protocol. Readable at: http://www.w3.org/TR/2000/NOTE-SOAP-20000508/. Accessed: 19.9.2011.

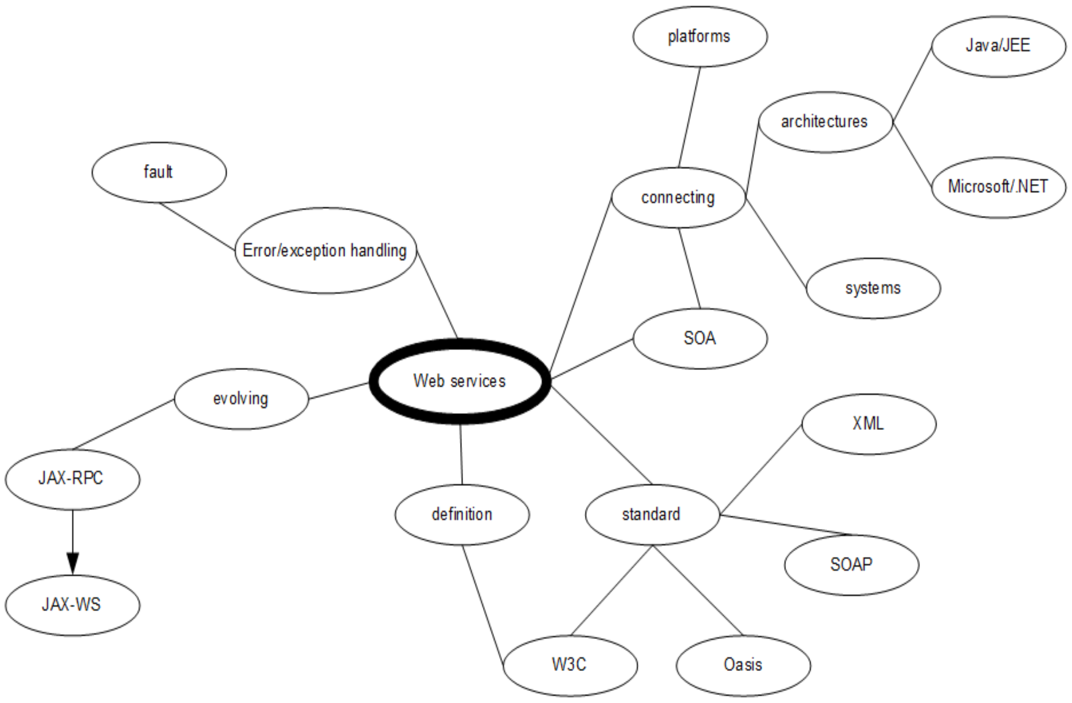
W3C 2000. SOAP Fault. Readable at: http://www.w3.org/TR/2000/NOTE-SOAP-20000508/#_Toc478383507. Accessed: 19.9.2011.

W3C 2011. Web services activity. Readable at: http://www.w3.org/2002/ws/. Accessed: 19.9.2011.

Stackoverflow.com. 2011. Mono wsdl tool fails to parse salesforce enterprise.wsdl. Readable at: http://stackoverflow.com/questions/2567466/mono-wsdl-tool-fails-to-parse-salesforce-enterprise-wsdl. Accessed: 14.5.2012.

Appendixes

Appendix 1. A mind map of the Web services world


platforms
Java/JEE
architectures
fault
connecting
Microsoft/.NET
Error/exception handling
systems
SOA
Web services
evolving
XML
JAX-RPC
definition
standard
SOAP
JAX-WS
W3C
Oasis