Report de Stage le semaine de 19 au 25 mai 2003

Following monday mornings meeting a certain number of deadlines were agreed.

Beta version ready for testing the 30th of June.

Version 1.0 for the 31st of August 2003.

That would leave the month of september for correction of bug's and documentation in english.

For the week of the 19 to the 25 I had a certain number of tasks to complete.

Firstly was to write a summary on Ksoap which I reaserched on the net and from several articles and technical descriptions I compiled a large amount of material from whih I wrote a brief summary.

The second task is one which has contiulaly cauesd problems for me and which as of now I have found no concrete solution to.

I have to run both the cds web services the name resolver and the GLU tag resolver.

Th problem appears to be from two possible sources one is that all the requirements for web services specify that you need java version 1.3 or above as of now java 1.1 is installed on my machine.

However I have sucessfully downloaded file:/usr/java/jdk1.3.1_08 by following the following instructions

****

B. Installing the Java SDK

Before we start, please note that Sun has provided an installation notes page for Linux installs of the Java SDK. That page is currently located (as of this writing) at:

http://java.sun.com/j2se/1.3/install-linux-sdk.html

You'll need the Java SDK from Sun. Go to http://java.sun.com and look for their downloads area. As of current writing its located at:
http://java.sun.com/j2se/1.3/download.html

If the url has changed, just go find the current Java SDK.

You can grab either the Linux RedHat RPM shell script (the one I have experience in) or the Linux GNUZIP Tar shell script. For Linux RedHat, download the SDK (NOT JRE!) version of the RPM shell script. (You'll get asked to accept the license agreement and then be asked to pick a location to download from.) Download this into the directory:

/home/yoda/downloads/

that you created a moment ago.

Now go to the directory and verify it is there:

```
cd /home/yoda/downloads/
ls -l
```

This should display the file in your directory.

```
j2sdk-1_3_1_03-linux-i386-rpm.bin
```

Confusion warning: This is a binary executable. You will have to run this before you can call the rpm command (if you even know what that is). And chances are, the binary won't run yet due to not being 'recognized' by Linux as an executable. So, you need to deal with these things one at a time:

Step 1: Make the binary executable by typing this:

```
chmod ug+x *.*
```

This makes all files (there's only one right now) executable.

Then, to run the file, you will probably have to prefix it when typing the command with:

```
./
```

Go ahead and do this:

```
./ j2sdk-1_3_1_03-linux-i386-rpm.bin
```

This should cause the binary to execute. If it doesn't, your bug is beyond my power to solve and its time to hit a newsgroup.

If the binary does run, then you will need to hit the space bar to page through the user agreement, then you'll be asked to type yes or no (type yes obviously and hit the enter key) and then it should extract the bin file out to an rpm file. For some non-intuitive reason, the resulting rpm name is not identical to the file name above.

Once it's done processing, verify that the rpm file exists by typing:

```
ls -l
```

Now you are ready to use the rpm command.

First, you must change to the superuser, because the rpm command is going to be installing java at the path:

```
/usr/java/
```

Since the /usr directory is protected, you will have to become superuser:

```
su root
```

Enter the password. You are now ready to proceed with the rpm command. You can research it if you want by typing rpm --help, but all you should need to do is simply type:

```
rpm --install rpm_filename_here_whatever_it_is.rpm
```

This should install the Java SDK into its default directory which is /usr/java/. Check this by going to the directory and doing a listing:

```
cd /usr/java/
ls -l
```

While you're here, write down the exact path of the java sdk BINARY directory (i.e. one level below the java sdk directory), as you are going to need to add it to the PATH environment variable shortly. For example, it might be:

```
/usr/java/jdk1.3.1_03/bin
```

Now, in order for the java sdk to be recognized in the system, you need to set the path environment variable. In RedHat Linux, this is set in the shell profile for each user, found in the primary directory for that user.

If the user's name were yoda, for example, on login they would be taken by default to the /home/yoda/ directory.

If the user were root, they would be taken to /root/ on login.

The point is that in that primary directory for the user there is a profile named:

.bash_profile

Why is it called bash? Because bash is the default shell script language used in RedHat Linux (as opposed to other unix shell script languages sh, csh, or tcsh).

This file needs to be edited. The editor I am familiar with using is emacs, so I will instruct you how to do this in emacs.

Go to your primary directory.:

```
cd /home/yoda/
```

If you do a listing, you should see your .bash_profile in the list:

```
ls -l
```

To edit it from emacs, type:

```
emacs .bash_profile
```

This should cause the file to open up in a new emacs editable window, and its PATH should look something like this:

```
PATH=$PATH:$HOME/bin
```

You want to append the path to the java binary directory to the end of the path statement. Check the piece of paper where you wrote down the name of the java sdk directory. When you have appended the path, your final result will be something like:

```
PATH=$PATH:$HOME/bin:/usr/java/jdk1.3.1_03/bin
```

Note that path names are separated by a colon, not a semicolon as in Windows. And note that no quotes are needed.

Now you are ready to save your .bash_profile. To save a file in emacs, you need to do two keystroke combinations:

Ctrl - x (prepares emacs to receive a command)
Ctrl - s (saves changes)

I have to assume that worked. If so, you are ready to exit emacs. To do this, you need to do

another two keystroke combinations:
Ctrl - x (prepares emacs to receive a command)
Ctrl - c (exits emacs)

Now, the act of logging out and logging back in to Linux should run the bash profile and thereby add Java to the path. Do this now.

Now that you are logged back in, type:

```
java
```

If your PATH editing was successful, then you should see a Usage listing of java command options. If it wasn't, then the command 'java' won't be recognized.

If this happens, recheck your path. If you are still having problems, contact one of the sun java newsgroups or ask your favourite Linux-knowledgeable friend (hoping you have one) for help.

****
I have followed all the instruction BUT I cannot get the new version to work over the old one, I have been using several java online fourms to help get answers to my various problems and so far the only plausable solution was "I have to correct my path to point to the new one" as of yet I have been unable to do so and I continue to look for a solution to this. I believe it is probanly something quiet simple and rudimentary but due to lack of experience in this domaine I continue to have difficulty.

However by using JBuilder8 I can select a higher version of java such as 1.3 or above this time I come across a different problem which maybe similar to the first problem if indeed webservices can run on java 1.1

Every time I run ANY client web service I get the following error:

Simple client

```
import org.apache.*;
import org.apache.axis.client.Service;
import org.apache.axis.utils.Options;
import javax.xml.rpc.namespace.QName;
import java.net.*;


public class HelloWorldClient
{
  public static void main(String args []) throws Exception
  {
    Service service = new Service();
    Call call = (Call)service.createCall();
    String endpoint = "http://localhost:8080/axis/HelloWorld.jws";

    call.setTargetEndpointAddress(new java.net.URL(endpoint));
    call.setPoerationName(new QName("getHelloWorld"));
    String output = (String)call.invoke(new object[]{});

    System.out.println("Got result : " + output);
  }
}
```

I get the following errors on JBuilder8.

```
"HelloWorldClient.java" : Erreur No : 302 : classe org.apache.axis.client.Service non accessible ;
java.io.IOException: classe introuvable : classe org.apache.axis.client.Service en ligne 3, colonne
31
"HelloWorldClient.java" : Erreur No : 302 : classe org.apache.axis.utils.Options non accessible ;
java.io.IOException: classe introuvable : classe org.apache.axis.utils.Options en ligne 4, colonne 30
"HelloWorldClient.java" : Erreur No : 302 : classe javax.xml.rpc.namespace.QName non accessible ;
java.io.IOException: classe introuvable : classe javax.xml.rpc.namespace.QName en ligne 5,
colonne 32
"HelloWorldClient.java" : Erreur No : 300 : classe Service introuvable  dans classe HelloWorldClient
en ligne 17, colonne 5
"HelloWorldClient.java" : Erreur No : 300 : classe Service introuvable  dans classe HelloWorldClient
en ligne 17, colonne 27
"HelloWorldClient.java" : Erreur No : 300 : classe Call introuvable  dans classe HelloWorldClient en
ligne 18, colonne 5
"HelloWorldClient.java" : Erreur No : 300 : classe Call introuvable  dans classe HelloWorldClient en
ligne 18, colonne 18
"HelloWorldClient.java" : Erreur No : 300 : classe QName introuvable  dans classe HelloWorldClient
en ligne 22, colonne 31
"HelloWorldClient.java" : Erreur No : 300 : classe object introuvable  dans classe HelloWorldClient
en ligne 23, colonne 45
```

I believe that in both cases it may be a problem stemming from the CLASSPATH this is the following code I've used in order to set the CLASSPATH: When using Java 1.1 I have no idea as of yet how to do this using JBuilde8 as I only have the free version form the internet it lacks alot of services including alot of the help files including the one concerning CLASSPATHS.

Changing the names where appropriate:

Begin by creating an environment variable for the directory we have been working with all this time, the webservices directory. Type the following at the command line. It's syntax is in accordance with the bash shell, used by Linux. (Other shells, such as tcsh, would use 'setenv' instead of 'export').

```
export WEBSERVICES_HOME="/home/yoda/webservices"
```

Now let's create an environment variable for the Axis home directory. Note how this environment variable builds on the previous one, giving you less to type. We will continue this technique going forward until we have built the entire classpath.

```
export AXIS_HOME="$WEBSERVICES_HOME/xml-axis-beta2"
```

Because we will use the lib directory within axis so much, let's create an environment variable to the lib directory itself:

```
export AXIS_LIB="$AXIS_HOME/lib"
```

Finally, we will refer at times to the OTHER lib directory--the one running in Tomcat, INSIDE of the webapp version of axis. Let's start by creating a home directory for Tomcat:

```
export TOMCAT_HOME="$WEBSERVICES_HOME/jakarta-tomcat-4.0.4-b3"
```

Now, we can create an axis webapp lib directory that derives from the TOMCAT_HOME environment variable:

```
export AXIS_WEBAPP_LIB="$TOMCAT_HOME/webapps/axis/WEB-INF/lib"
```

Clear as mud? OK, now we're ready to put these altogether into a very long classpath:

first we need a path to the Axis base directory. Why? Because all of the sample Java files we intend to compile and run derive their package roots from the axis base directory.
second, we need a path to the 6 jars found inside the Axis lib directory, as these contain the complete class libraries for Axis itself.
third, we need a path to the xerces.jar which we dropped INTO the Axis lib directory, otherwise the xml used by the web services as their standard language won't be readable
fourth, we need a path to the junit.jar so that the unit tests found in some of the sample clients will be able to run.
Later, we will be creating web services and placing them as jar files inside the Tomcat axis webapp. At that point we will ADD further information to the CLASSPATH as needed.

Enter the CLASSPATH environment variable now:

```
export CLASSPATH="$AXIS_HOME:$AXIS_LIB/axis.jar:\
$AXIS_LIB/commons-logging.jar:$AXIS_LIB/jaxrpc.jar:\
$AXIS_LIB/log4j-core.jar:$AXIS_LIB/tt-bytecode.jar:\
$AXIS_LIB/wsdl4j.jar:$AXIS_LIB/xerces.jar:\
$WEBSERVICES_HOME/junit3.7/junit.jar"
```

You are now ready to begin creating and running webservice examples! We're going to look at 3 examples:
The first uses the most simple deployment method: drop an uncompiled java file in the Axis directory with a .jws extension and start running it.
The second uses special axis utilities called Java2WSDL and WSDL2Java to create the complete classes needed to run a more sophisticated webservice.
The third is one of the existing sample webservices provided by Axis in its samples subdirectory.

Still no sucess.

This has taken up much of my time as it is the most important element of last week's work another task I was asked to perform was to write a simple program which would be able to calculate the area of a Circle and eploy this as a web service

This again I have deployed but cannot run the client side until I have solved my other problems.

```
import java.io.*;
public class Circle
{
  public static void main(String args[])
  {
    double pie = 3.14159;
    double ans = 0.0;
    double radius = 0.0;
    double diameter=0.0;
    String userName= null;
    String choice=null;

    System.out.println("Please choose either Diameter or Radius by typing either radius or diameter");

    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

    try
    {
     choice=br.readLine();
    }

   catch(IOException ioe)
   {
     System.out.println("Error choice not radius or diameter");
     System.exit(1);
   }



   if((choice.equals("diameter"))||(choice.equals("Diameter")) )
   {
     System.out.println("Please enter a diameter");

          try
           {
             userName=br.readLine();
             diameter=Double.parseDouble(userName);
             radius =diameter/2;
           }

           catch(IOException ioe)
           {
             System.out.println("Error");
             System.exit(1);
           }

   }

else
   if((choice.equals("radius"))||(choice.equals("Radius")))
   {
     System.out.println("Please enter a radius");

     try
       {
        userName=br.readLine();
        radius=Double.parseDouble(userName);

        }

        catch(IOException ioe)
        {
        System.out.println("Error");
        System.exit(1);
        }

      }


    ans=(radius*radius)*pie;
    System.out.println("The area of this circle is approx "+ans);

  }
```

}// end of class circle

The WSDL for this file is

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="http://localhost:8080/axis/Circle.jws"
xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:apachesoap="http://xml.apache.org/xml-soap"
xmlns:impl="http://localhost:8080/axis/Circle.jws" xmlns:intf="http://localhost:8080/axis/Circle.jws"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"><wsdl:types><schema
targetNamespace="http://localhost:8080/axis/Circle.jws"
xmlns="http://www.w3.org/2001/XMLSchema"><import
namespace="http://schemas.xmlsoap.org/soap/encoding/"/><complexType
name="ArrayOf_xsd_string"><complexContent><restriction base="soapenc:Array"><attribute
ref="soapenc:arrayType"
wsdl:arrayType="xsd:string[]"/></restriction></complexContent></complexType></schema></wsdl:typ
es>
  <wsdl:message name="mainResponse">
  </wsdl:message>
  <wsdl:message name="mainRequest">
    <wsdl:part name="args" type="intf:ArrayOf_xsd_string"/>
  </wsdl:message>
  <wsdl:portType name="Circle">
    <wsdl:operation name="main" parameterOrder="args">
      <wsdl:input message="intf:mainRequest" name="mainRequest"/>
      <wsdl:output message="intf:mainResponse" name="mainResponse"/>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="CircleSoapBinding" type="intf:Circle">
    <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="main">
      <wsdlsoap:operation soapAction=""/>
      <wsdl:input name="mainRequest">
        <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://localhost:8080/axis/Circle.jws" use="encoded"/>
      </wsdl:input>
      <wsdl:output name="mainResponse">
        <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://localhost:8080/axis/Circle.jws" use="encoded"/>
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="CircleService">
    <wsdl:port binding="intf:CircleSoapBinding" name="Circle">
      <wsdlsoap:address location="http://localhost:8080/axis/Circle.jws"/>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

I will continue as fast as possible to finaly solve my web service client problems as soon as possible
I fully realise how urgently this must be resolved.

Damien Cummins 24/05/2002