University of Stirling
Computing Science
Web Services

Practical: Web Service Operation with SOAP

This practical will allow you to understand how web services communicate. You will use (and change) the Approver service from Lab 2. Make a copy of the Lab2 projects, so you don't overwrite your working solutions during this lab.

Setting up SOAPMonitor

SOAPMonitor is a module that is incorporated into the Axis2 server that is used for monitoring the SOAP Requests and Replies. We will use SOAPMonitor in this practical to observe the messaging between client and service. SOAPMonitor is using an Applet which is created by a Servlet and communicates with the Servlet. Recent versions of Java are strict about the applets they will run. To run the SOAP Monitor used in this practical, we need to make some configuration changes.

Navigate to H:\tomcat8\webapps\axis2\WEB-INF\conf. Edit the file `axis2.xml` with Textpad. Add `<module ref="soapmonitor"/>` after the line `<module ref="addressing"/>`. (about line 250). Save and Exit. Then navigate to H:\tomcat8\webapps\axis2\WEB-INF and edit `web.xml`.

Uncomment the following codes (at about line 50):

```
<servlet>
    <servlet-name>SOAPMonitorService</servlet-name>
    <display-name>SOAPMonitorService</display-name>
    <servlet-class>org.apache.axis2.soapmonitor.servlet.SOAPMonitorService</servlet-class>
    <init-param>
        <param-name>SOAPMonitorPort</param-name>
        <param-value>5001</param-value>
    </init-param>
    <init-param>
        <param-name>SOAPMonitorHostName</param-name>
        <param-value>localhost</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>
```

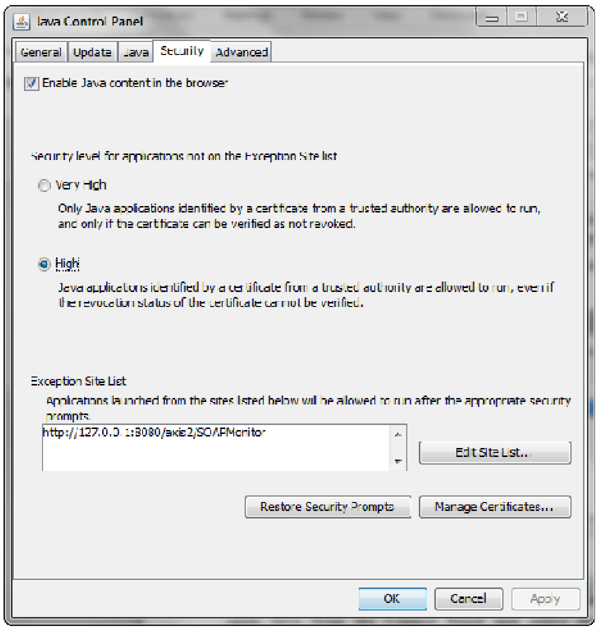
Change the port number to 8081 in the line `<param-value>5001</param-value>`

Uncomment the following lines (at about line 85).

```
<servlet-mapping>
    <servlet-name>SOAPMonitorService</servlet-name>
    <url-pattern>/SOAPMonitor</url-pattern>
</servlet-mapping>
```

Save and Exit.

Next, select Java/Configure Java from the Start menu and select the Security tab. Now click Edit Site List, enter `http://127.0.0.1:8080/axis2/SOAPMonitor` and add it (see screenshots below). Accept the security warning, and click OK until you exit the panel. When you later try the SOAP Monitor you should allow it to run if there is a Java warning.


Java Control Panel

General | Update | Java | Security | Advanced

[✓] Enable Java content in the browser

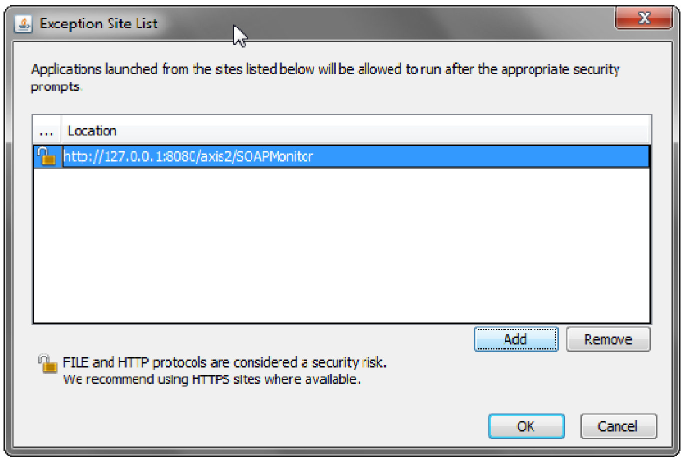
Security level for applications not on the Exception Site list

( ) Very High
Only Java applications identified by a certificate from a trusted authority are allowed to run, and only if the certificate can be verified as not revoked.

[● High]
Java applications identified by a certificate from a trusted authority are allowed to run, even if the revocation status of the certificate cannot be verified.

Exception Site List
Applications launched from the sites listed below will be allowed to run after the appropriate security prompts.
http://127.0.0.1:8080/axis2/SOAPMonitor

Edit Site List... | Restore Security Prompts | Manage Certificates... | OK | Cancel | Apply


Exception Site List

Applications launched from the sites listed below will be allowed to run after the appropriate security prompts

...	Location
	http://127.0.0.1:8080/axis2/SOAPMonitor

Add | Remove

FILE and HTTP protocols are considered a security risk.
We recommend using HTTPS sites where available.

OK | Cancel

The change you have just made allows the servlet to launch the applet. Next we need to create a user specific java.policy file to allow the applet to connect back to the servlet.

Using Textpad create a text file named .java.policy (watch out for the initial ‘.’) in the directory C:\Users\xyz (where xyz is your CS username) with the following content:

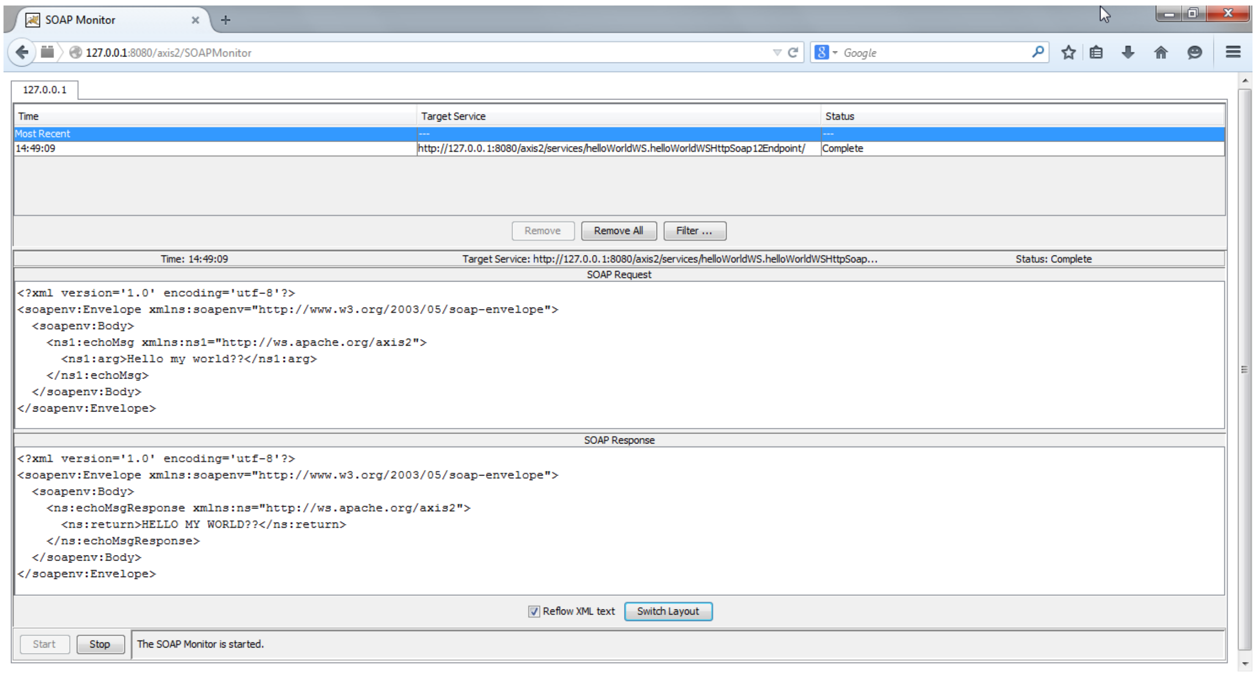
```
grant {
  permission java.net.SocketPermission "127.0.0.1:8081", "accept,connect,listen,resolve";
};
```

This is a user specific java policy file (see the following web page http://docs.oracle.com/javase/7/docs/technotes/guides/security/PolicyFiles.html for more detail).

Unfortunately, the use of the user specific policy file is disabled by default. This is another configuration change: go to C:\Users\mko\AppData\LocalLow\Sun\Java\Deployment and edit the file `deployment.properties` which should exist. Add a line near the top:

```
deployment.security.use.user.home.java.policy=true
```

Now start Tomcat and then Internet Explorer or Firefox browser (Chrome does not work for this.). Your previous web services should still be deployed. The client and the service are exchanging SOAP messages. What do they look like? Open *http://127.0.0.1:8080/axis2/SOAPMonitor* in a browser (not Chrome!). You may see some security warnings and pleas to update the Java browser plugin. Ignore the warning and allow the applet to run. Execute the web service client (from within eclipse). The top portion of the SOAP monitor shows each exchange. Click on one, and view the SOAP request and response. To see the XML neatly laid out, click the ‘Reflow XML Text’ option at the bottom. See what ‘Switch Layout’ does. You should see a screen similar to the screenshot below.



Time	Target Service	Status
Most Recent	---	---
14:49:09	http://127.0.0.1:8080/axis2/services/helloWorldWS.helloWorldWSHttpSoap12Endpoint/	Complete

Time: 14:49:09 Target Service: http://127.0.0.1:8080/axis2/services/helloWorldWS.helloWorldWSHttpSoap... Status: Complete

SOAP Request

```
<?xml version='1.0' encoding='utf-8'?>
<soapenv:Envelope xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope">
  <soapenv:Body>
    <ns1:echoMsg xmlns:ns1="http://ws.apache.org/axis2">
      <ns1:arg>Hello my world??</ns1:arg>
    </ns1:echoMsg>
  </soapenv:Body>
</soapenv:Envelope>
```

SOAP Response

```
<?xml version='1.0' encoding='utf-8'?>
<soapenv:Envelope xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope">
  <soapenv:Body>
    <ns:echoMsgResponse xmlns:ns="http://ws.apache.org/axis2">
      <ns:return>HELLO MY WORLD??</ns:return>
    </ns:echoMsgResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

Reflow XML text: ✓

The SOAP Monitor is started.

Investigating SOAP messaging

By the time you have finished this practical, you should have a different SOAP request/response pairs for different SOAP styles and literal encoding (SOAP encoded styles are not supported by Axis2).

At the moment, both the client and the service are using *rpc/literal*. Copy a typical SOAP request/ response from the SOAP monitor. Use Alt-Print Screen to copy the window and paste it into MS Word. Click 'Remove All' to clear the SOAP monitor messages.

The next step would be to change the client and service to use *document/literal* instead. However, before you do so, as the *document* style requires to have a single element encapsulating multiple parameters in the request message, let's include the three parameters in a structure (WSDL complexType). For this you will need a types section in the WSDL. See below for a start of this type definition.

```
<types>
   <schema elementFormDefault="qualified" targetNamespace="urn:Approver"
     xmlns="http://www.w3.org/2001/XMLSchema">

   <complexType name="propType">
     <sequence>
         …
     </sequence>
   </complexType>

  </schema>
 </types>
```

You can then use this new type with your request message. Afterwards, you will need to regenerate the skeleton and stubs. Also your service and client code will need to change a bit to take into account the new data structure. Deploy the new version of the service and use the client to call the service. Capture the message with SOAPMonitor. Do you see a difference?

Now, let's change to *document/literal* style. Open *Approver.wsdl* in a text editor. Where it says *soap:binding style="rpc"*, change this to use *"document"* instead. One additional requirement for the document style is that the message parts are declared using WSDL elements, rather than types. You will need the following element specifications as part of the schema definition introduced above:

```
<element name="proposal" type="app:propType"/>
<element name="rate" type="xsd:float"/>
```

You can then use the proposal and rate elements with the parts in your request and response messages.

Then regenerate the skeletons and stubs and adapt your service and client code. Redeploy the service and use client to execute service while using SOAPMonitor to capture the message exchange. Compare the results you just got from *document/literal* with those you got earlier with *rpc/literal*. What is different?

By sharing the definitions in *Approver.wsdl*, the client and server use a common SOAP format. Suppose, however, that the client were coded to use *rpc/literal* and the service to use *document/literal*. What do you think would happen when you tried to run the client?

Checkpoint

Show a lab demonstrator the samples of SOAP requests and responses you copied. Show that you have understood the general principles of each format. Explain what would happen with using *rpc/literal* in the client but *document/literal* in the service.

Extra Activity:

Try introducing a fault message in your WSDL and change your service to raise a fault for a certain set of parameters (e.g. requested amount too large).

A fault message in WSDL can be defined as follows:

```
<message name="errorFault">
  <part name="errorPart" element="…"/>
</message>
```

This message can then be used as part of operation definitions like other request/response messages, e.g.

```
<operation name="approveOperation">
  <input message="app:proposalMessage"/>
  <output message="app:rateMessage"/>
  <fault message="app:errorMessage"/>
</operation>
```

You can throw an exception in Java using code such as:

```
throw (new ErrorFault(errorMessage, exception));
```

where ErrorFault is the fault message as defined in the WSDL. errorMessage maps to the WSDL element used to define the fault message and exception a Java exception object.

Demonstrate the fault for a **Bonus Checkpoint**.