QualiPSo

Quality Platform for Open Source Software

IST- FP6-IP-034763

QualiPSo – WP3.2 – Task T3.2.3

Installation and Usage Guide for SecureComm

Klaudios Kontis, European Dynamics
Kostas Lykourgiotis, European Dynamics
Panagiotis Kavalagios, European Dynamics
Dejan Kocic, European Dynamics

Version 1.2

TABLE OF CONTENTS

1 INTRODUCTION

This document provides clear and simple instructions guiding the installation of the SecureComm artefacts. It also provides concrete usage instructions by illustrating how to define a new security-mapping service in SecureComm. Next, it provides two concrete and thorough examples of defining and using a security mapping. Each example illustrates the deployment of a test client and a test back-end server (available for download from the SVN and the Files repository of the project) and the deployment of the required configuration in SecureComm for bridging the security incompatibilities between the client and the server.

Artefacts mentioned in this document (SecureComm, TelCo service and DAML2NRL examples) are described in details in the QualiPSo project work-deliverables[1]:

- wd3.2.3: "Semantic Interoperability – Open Source middleware and forge platform enhancements: software and documentation", under the chapter "SecureComm".
- wd3.2.4: "Semantic Interoperability Scenarios and Experiments", under the chapter "Secure OSS Comunication Middleware".

[1] Both deliverables are available under http://www.qualipso.org/interoperability -> "Semantics Interoperability".

2 SecureComm Project Structure

It is assumed that you have checked out – or now you will checkout the complete SecureComm from the SVN location:

http://svn.berlios.de/svnroot/repos/securecomm

The SVN directory contains the implementation of SecureComm as well as the TelCo client-server and DAML2NRL stub pair. The table below presents the most important subdirectories of SecureComm. The abbreviations given in a UNIX shell variable format are used later in this document for the sake of short and clear presentation. When an abbreviation is used in a UNIX command, replace it with the **full UNIX path** to the respective directory.

Subdirectory	**Role and Abbreviation**
SecureCommImplementation	The software implementation `${IMPLEMENT}`
SecureCommTesting	Direcotry hosting the “`TelcoService`” and the “`DAML2NR`” usage examples.
SecureCommTesting/TelcoService/client/asm-node1	Stub client implementation of the TelCo and DAML2NRL service `${TELCO_ASM_NODE}`
SecureCommTesting/TelcoService/server/telcoService_AppWebService	Stub server implementation of the TelCo service `${TELCO_AppWebService}`
SecureCommTesting/TelcoService/secureCommFiles/xsl_based	XSL based solution for implementing the bridging between the TelCo client and server in SecureComm `${XSL_BASED_EXAMPLE}`
SecureCommTesting/TelcoService/secureCommFiles/customJava_based	Java based alternative solution for implementing the bridging between the TelCo client and server in SecureComm `${JAVA_BASED_EXAMPLE}`
SecureComm/SecureCommTesting/DAML2NRL/SecureCommFiles	DAML2NRL XSLTs and ontologies files `${DAML2NRL_FILES}`
SecureComm/SecureCommTesting/DAML2NRL/client/DAMLWebClient	DAML web client implementation example `${DAML_WEB_CLIENT}`

SecureComm/SecureCommTesting/DAML2NRL/server/NRLService	NRL Web Service implemenation example `${NRL_WEB_SERVICE}`

In the following chapters all UNIX commands are prefixed by a bold dollar character followed by a space, “`$ `” denoting the shell prompt. Long commands can be wrapped in many visual lines.

3 SECURECOMM INSTALLATION

This chapter describes how to install SecureComm. It also contains references to the installation instructions of software artefacts required by SecureComm.

3.1 Setting up the runtime environment

In order to test SecureComm the runtime environment must be properly set up. This implies:

1. A working instance of Apache Servicemix (ASM)

   ASM installation instructions can be found at http://servicemix.apache.org/1-quick-start.html. It is advised to have the environmental variable ASM_HOME set to the home directory of ASM while executing the procedures described in this document. It is also advised to check and if required to change the various port settings found in "$ASM_HOME/conf/servicemix.properties" to custom values in order to avoid conflicts with ports used by other applications on your system. SecureComm was tested against Apache Servicemix 3.2.

2. Apache Maven2

   In order to create, modify and deploy examples into SecureComm the maven2 tool is necessary. Please start from http://maven.apache.org/download.html in order to install and configure maven2.

The following components are required for deploying and test client and a test server application described later in this document:

3. Apache Tomcat with Axis2

   Apache Tomcat need to be installed and properly configured as described in http://tomcat.apache.org/tomcat-6.0-doc/setup.html. We have assumed in this document that Apache Tomcat is running on port 18090. Once successful, Axis2 must be deployed in the Tomcat webapps folder as explained in http://ws.apache.org/axis2/0_94/installationguide.html#_Toc96698086

4. SOAP client

   A soap client must be installed in order to execute the Telco service example. We recommend SoapUI, available at http://www.soapui.org/

3.2 Deploying SecureComm

SecureComm consists of two separate artefacts: The "Service Administration" and the "Runtime" service-assemblies, both deployed in a single ASM instance. The Runtime is responsible for accepting incoming requests from client applications, applying the defined security mappings on the requests and dispatching the adapted requests to the server applications outside ASM.

The Service Administration provides means for a semantic expert to manage the security mappings executed by the Runtime service-assembly.

Before being able to manage security mappings using the administration interface, the ASM administrator must define into SecureComm the respective application and redeploy SecureComm in ASM. An example will be provided in section 5.

3.2.1 Description of SecureComm Runtime

The directory: `${IMPLEMENT}/SecureCommRuntime-SA/` contains the following: `securecomm-con-http`, `securecomm-camel`, `securecomm-wssec-bean`, `securecomm-prov-http` and `securecomm-sa`. In ASM terminology the first 4 are service-units while securecomm-sa is the service assembly that will contain the zip file that will be finally deployed into ASM. In brief, explaining each one:

- securecomm-con-http

  In this service unit the ASM administrator will define the entrypoints to the exposed, out of ASM, applications. This is a consumer endpoint as that is explained in http://servicemix.apache.org/servicemix-http.html. It's main job is to accept incoming request for a specific service and forward those request to the ASM chain of processing.

- securecomm-camel

  Implements the routing part inside SecureComm. It is plugged after the securecomm-con-http service-unit and it delivers the messages to the securecomm-wssec-bean and finally to the securecomm-prov-http which in turn delivers them to the real application that serves the specific client/example.

- securecomm-wssec-bean

  This component is responsible for the delegation of the ASM encapsulated information to the proper semantic authenticator module in charge of the specific service.

- securecomm-prov-http

  In this service unit, the ASM administrator will define the URL of the application that is serving this client or request. This is a consumer endpoint as that is explained in http://servicemix.apache.org/servicemix-http.html. It's main job is to push requests coming from the ASM chain of processing to the real applications serving the request.

- securecomm-sa

  The service assembly consisting of the JBI package of the above units along with any dependencies. In the end, the produced zip file of the service assembly will be uploaded into ASM.

3.2.2 Description of Service Administration

This artefact allows the administration of the services/applications attached to SecureComm. Under the directory `${IMPLEMENT}/ServiceAdministration-SA/` you will find the following subdirectories: `frontend-consumer-su`, `frontend-handler-su and frontend-sa`. In ASM terminology the first 2 are service-units while `frontend-sa` is the service assembly containing the zip file that will be finally deployed into ASM. In brief, explaining each one:

- frontend-consumer-su

  This service unit is responsible for listening to incoming administrative request. It contains XML configurations and java source code that execute the request. Unless port conflicts appear, there will never be need to modify this service-unit.

- frontend-handler-su

  In the ASM model, there are consumer and provider units. In this case, the application serving the request is ASM itself therefore this service unit only sets some internal properties and returns the message.

- frontend-sa

The service assembly consisting of the JBI package of the above units along with any dependencies. In the end, the produced zip file of the service assembly will be uploaded into ASM.

3.2.3Compilation and Deployment

For the deployment, you need the file `SecureComm_structure.tgz` which already resides in the directory `${IMPLEMENT}/SecureCommRuntime-SA/` and contains the required file-structure. Exatrct the archive in the `$ASM_HOME` directory:

```
$ cd $ASM_HOME
$ tar xzvf ${IMPLEMENT}/SecureCommRuntime-
SA/SecureComm_structure.tgz
```

Verify successful extraction by checking the existence of the `Files/` directory within the `$ASM_HOME`. To complete the configuration you have to edit two files.

Edit `$ASM_HOME/conf/users-credentials.properties` and insert the line:

```
serveront=CN=myserverCN,OU=myserverOU,O=eurodyn,C=GR
```

Edit `$ASM_HOME/conf/users-passwords.properties` and insert the line:

```
serveront=123456
```

In order to produce the service assemblies of the above components, you must go to each of the directories the root directory of the two service assemblies and do a Maven clean install:

```
$ cd ${IMPLEMENT}/SecureCommRuntime-SA/
$ mvn clean install
$ cd ${IMPLEMENT}/ServiceAdministration-SA/
$ mvn clean install
```

This will produce 2 jar files:

- The deployable package responsible for the Service Administration:

  ```
  ${IMPLEMENT}/ServiceAdministration-SA/frontend-
  sa/target/frontend-sa-1.0-SNAPSHOT.jar
  ```

- The deployable package responsible for SecureComm Runtime:

  ```
  ${IMPLEMENT}/SecureCommRuntime-SA/securecomm-
  sa/target/securecomm-sa-1.0-SNAPSHOT.jar
  ```

Then, the produced files must be copied to the ASM hot-deploy directory renamed as zip archives. We recommend starting ASM and then doing the copy in order to see live any deployment messages:

```
$ cd $ASM_HOME
$ ./bin/servicemix &
$ cd ${IMPLEMENT}/SecureCommRuntime-SA
$ cp securecomm-sa/target/securecomm-sa-1.0-SNAPSHOT.jar
$ASM_HOME/hotdeploy/securecomm-sa-1.0-SNAPSHOT.zip
$ cd ${IMPLEMENT}/ServiceAdministration-SA
$ cp frontend-sa/target/frontend-sa-1.0-SNAPSHOT.jar
$ASM_HOME/hotdeploy/frontend-sa-1.0-SNAPSHOT.zip
```

After each copy command, you should see a success message from the ASM hot-deployer for frontend-sa, but for securecomm-sa you may safely ignore the problem that we have not defined any endpoints. Web interface will also display “ASM Administrator has not defined any services”.

4 USAGE: DEFINING A SERVICE IN SECURECOMM

Defining a service into SecureComm means: **(a)** defining the required ASM endpoint(s) for the service, **(b)** making that service available **and (c)** deploying a service-specific authenticator (security mapping) module. An ASM endpoint is where the client request will be directed. Further on, the client request will be delivered from SecureComm to the authenticator module for processing and the results forwarded to the server application. The authenticator module performs 3 distinguished actions: semantic lifting, bridging and grounding therefore besides the ontology files, the configuration may include XSL files and/or custom java code that perform any of the above operations. Actions (a), (b) and (c) are executed by different actors as listed below in details.

4.1 ASM Administrator

The steps executed by the ASM administrator are the following:

1. Define a consumer endpoint bound to the service

   In the securecomm-con-http service unit of SecureComm define an consumer endpoint as that is explained in http://servicemix.apache.org/servicemix-http.html

   Plese consult this for how to add a new endpoint or do so by copy-pasting and modifying the existing, commented out example endpoint of the TelcoService. For example:

```
File:          ${IMPLEMENT}/SecureCommRuntime-SA/securecomm-con-
http/src/main/resources/xbean.xml
```

```
<http:endpoint
    service="secCommNode2:secCommNode2ServerServiceConIn"
    endpoint="secCommNode2HttpSecConInServerEnd"
    targetService="secCommNode2:camelin"
    role="consumer"
    locationURI="http://127.0.0.1:8291/secConIn/"
    defaultMep="http://www.w3.org/2004/08/wsdl/in-out"
    soap="true"
    soapVersion="1.1"
/>
```

   In the above example the properties in bold must not change since from it depends the internal routing within the SecureComm module. The other attributes like **service**, **locationURI** etc. must change according to the settings of your example (e.g. put the hostname, port where your client will be connected, etc).

2. Define a provider endpoint

   In the securecomm-prov-http service unit of SecureComm define a provider endpoint as that is explained in http://servicemix.apache.org/servicemix-http.html

Plese consult this for how to add a new endpoint or do so by copy-pasting and modifying the existing, commented out example endpoint of the TelcoService. For example:

```
File:             ${IMPLEMENT}/SecureCommRuntime-SA/securecomm-prov-
http/src/main/resources/xbean.xml
```

```
<http:endpoint
    service="secCommNode2:secCommNode2ServerServiceProvIn"
    endpoint="secCommNode2HttpPlainProvInServerEnd"
    role="provider"
    locationURI=
         "http://127.0.0.1:18090/axis2/services/TelcoServiceXSLT"
    defaultMep="http://www.w3.org/2004/08/wsdl/in-only"
    wsdlResource=
       "http://127.0.0.1:18090/axis2/services/TelcoServiceXSLT?wsdl"
    soap="true"
    soapVersion="1.1"
/>
```

In the above example, besides, the properties defined in black, all other should be modified accordingly to your example settings (e.g. put the hostname, port where your server application listens, etc).

3. Redeploy SecureComm Runtime

   Redeploy SecureComm for the new application to be visible within SecureComm. Please stop ASM if running and execute the commands:

```
$ cd ${IMPLEMENT}/SecureCommRuntime-SA
$ mvn clean install
$ rm $ASM_HOME/hotdeploy/securecomm-sa-1.0-SNAPSHOT.zip
$ cd $ASM_HOME && ./bin/servicemix &
$ cd ${IMPLEMENT}/SecureCommRuntime-SA
$ cp securecomm-sa/1.0-SNAPSHOT/securecomm-sa-1.0-
SNAPSHOT.zip $ASM_HOME/hotdeploy
```

   You should see a undeploy message when starting ASM and finally a success message when performing the copy of the new SecureComm service assembly.

4. Make Visible

   In order for the service to be manageable from the Administration interface and therefore workable, it must first be declared. From the ASM home, edit the file:

   `$ASM_HOME/Files/Common/Properties/service_endpoints.properties`

   There you need to add a `service-name=endpoint-name` pair; service-name is something that you can freely choose. The endpoint-name is defined in step 2 as the value of the "`endpoint`" XML attribute. Check existing commented out example in this file for reference.

4.2 Semantic expert

Once a service is available the needed data for it to function properly must be made available by the Semantic expert. This is done through the administration interface.

Therefore, the steps to be executed at this stage are:

1. Upload an authenticator module for available services by accessing http://your-host.domain:8989/frontend/?operation=uploadService where your-host.domain and 8989 must be replaced with the hostname and port of your ASM.
2. You should see the "Deploy Security Mapping" page. Select the name of the service in the "Choose applicable services" drop down.
3. Select method of Semantic Lifting. If XSL is chosen an XSL stylesheet must be uploaded as well. If custom java is selected then it must be provided in the JAR file field down below.
4. In the Semantic Bridging field provide the ontology files as request. The source, bridge and target ontology are mandatory while other ontologies can be uploaded as well in the respective field; other ontologies can only be provided in the form of a single ZIP file containing 1 or more OWL files. Select the bridging method: In "Generic" bridging is performed by the generic built-in SecureComm Bridger. In "Custom" bridging is performed by some custom java code that you must provide in the JAR file field down below.
5. Select method of Semantic Grounding. If XSL is chosen an XSL stylesheet must be uploaded as well. If custom java is selected then grounding code must be provided in the JAR file field down below.
6. The "Custom Java Code for bridging and processing" needs to be completed only if some custom option is selected above. In the "Java Class" field, the full package and name of the class responsible for the bridging, lifting or grounding processes must be inserted. The class files must be provided as jars in the "JAR file" field. In the "Other Jars" field other necessary jars for the authenticator can be uploaded (if many in zip format) such as for example the class code for a custom SWRL built-in. The interface implemented by the JAR file is described in Appendix: Custom Java Code API.
7. Upload the data and verify its existence by accessing the http://your-host.domain:8989/frontend/?operation=listServices and clicking on the desired name. You should see all the files you inserted as a list.
8. The service will be ready to use immediately if no custom java code is used or after the administrator has restarted ASM if any custom java code is provided.

5 THE TELCO SERVICE USAGE EXAMPLE

In this section we provide a practical example of the concrete actions needed to add a functioning service in SecureComm. The TelCo (Telecommunication Company) service is a stump service supporting a set of simple operations. The test client and the test server have different representations for credentials; these differences are bridged in SecureComm. The full description of the service and the security bridging performed by SecureComm is documented in wd3.2.3.

In the following sections we describe how to set-up the back-end server and the client of the Telco service and how to add the service bridging in the context of Apache Servicemix and the SecureComm.

5.1 Back-end Server Setup

By server setup we mean the real application to which the processed requests are sent from ASM. In the case of the Telco-Service this is a Web Service providing two simple operations. This sub-section assumes the necessary software described in § 3.1 is installed. In order to prepare the server-part of the following steps must be executed:

1. Make sure that you have checked-out the web service under the directory `${TELCO_AppWebService}`.
2. Into the `${TELCO_AppWebService}` directory edit the `build.xml` file and change the values of the `AXIS2_HOME` and `TOMCAT_HOME` properties to your server values.
3. Recompile and deploy into Tomcat by executing the command from the checkout dir.

   ```
   $ ant tomcatDeploy
   ```

4. Start Tomcat and verify correct installation by accessing the WSDL of the service from http://your-host.domain:18090/axis2/services/listServices

The URL of the WSDL will be used further on when defining the endpoint in the `securecomm-prov-http xbean.xml` configuration file, explained further on.

5.2 ASM Administration required actions

In order to add support for the Telco-Service into SecureComm, the following things must be done, in accordance with the procedures described in section 4.1[2].

1. Define a consumer endpoint.

[2] You will notice that the artefacts downloaded from the project repository already contain the required configuration for the TelCo-Service example, commented out.

In the file `${IMPLEMENT}/SecureCommRuntime-SA/securecomm-con-http/src/main/resources/xbean.xml` uncomments the content in between the `<##### TELCOSERVICE Example ..>` tags.

2. Define a provider endpoint.

In the file `${IMPLEMENT}/SecureCommRuntime-SA/securecomm-prov-http/src/main/resources/xbean.xml` uncomment the content in between the `<##### TELCOSERVICE Example ..>` tags.

If needed, please adjust the host, port of the Web-Service to your preferred values.

3. Redeploy SecureComm as stated in step 3 of § 4.1.
4. Make the service available as stated in step 4 § of 4.1.

5.3 Actions required for Semantic mapping definition

The authenticator module for the Telco-Service can be based on XSL style-sheets or custom Java code. We shall present both alternatives.

5.3.1XSLT based solution

1. Make sure you have already the OWL and XSL files under the directory `${XSL_BASED_EXAMPLE}`. Files below are relevant to this path.
2. Execute step 1 of § 4.2
3. Select the Telco-service in the step 2 of § 4.2
4. In step 3 of § 4.2 select XSL and upload the file `XSLT/lift.xsl`
5. Upload the ontologies found in the `ONTOLOGY/` directory as indicated by their names into the respective fields e.g. `sourceOntology.owl` is to be uploaded in the “source ontology” form field. Select “Generic” bridging.
6. In step 6 of § 4.2 select XSL and upload the file `XSLT/full_ground.xsl`
7. Submit the form and verify success as told in step 7 of § 4.2.

5.3.2Custom Java Based Solution

1. Make sure you have already the OWL and XSL files under the directory `${JAVA_BASED_EXAMPLE}`. Files below are relevant to this path.
2. Go inside the directory and execute the following command in order to build the needed jar file:

```
$ cd ${JAVA_BASED_EXAMPLE}
$ mvn clean install
$ ls -lt  target/telcoService-1.0-SNAPSHOT.jar
```

   Verify the creation of the jar will be uploaded into SecureComm.

3. Execute step 1 of § 4.2
4. Select the Telco-service in the step 2 of § 4.2
5. In step 3 of § 4.2 select the option “Based on Java code provided below”.

6. Upload the ontologies found in the `ONTOLOGY/` directory as indicated by their names into the respective fields e.g. `sourceOntology.owl` is to be uploaded in the “source ontology” form field. Select “Based on Java code provided below” concerning bridging.
7. In step 6 of 4.2 select the option “Based on Java code provided below”
8. In the “Java Class implementing Processing and Bridging” field insert the following class name as an entry point to the custom java “`org.qualipso.interop.semantics.securecomm.telcoService.QualipsoSemanticMapper`”. In the “JAR file implementing the class” field upload the jar file created at step 2.
9. Submit the form and verify success as told in step 7 of § 4.2.

5.4 Client Setup

On the client’s part we need a soap GUI client in order to do the testing. Furthermore, since all incoming request are expected to be digitally signed by X509 we need to sign the client requests before pushing them to SecureComm.

5.4.1Signing the request

We chose to sign the request by using another instance of ASM and uploading a service assembly. This instance will be put in between the client tool and the ASM instance containing SecureComm and its only job is to accept the clients requests sign them and dispatch them to the ASM holding SecureComm. The steps to do the above are:

1. Install another instance of ASM as told in § 3.1 in a directory further referred to as `$ASM2_HOME`.
2. Extract the file `${TELCO_ASM_NODE}/asm2_test_structure.tgz` in the `$ASM2_HOME` directory.

```
$ cd $ASM2_HOME
$ tar xzvf ${TELCO_ASM_NODE}/asm2_test_structure.tgz
```

Edit `$ASM2_HOME/conf/users-credentials.properties` and insert the line:

```
serveront=CN=myserverCN,OU=myserverOU,O=eurodyn,C=GR
```

Edit `$ASM2_HOME/conf/users-passwords.properties` and insert the line:

```
serveront=123456
```

3. Edit the XML file: `${TELCO_ASM_NODE}/asm1-http-con-su/src/main/resources/xbean.xml` and uncomment the content in between the `<##### TELCOSERVICE Example ..>` tags. This is where the client will send its request.
4. Edit the XML file: `${TELCO_ASM_NODE}/asm1-http-prov-su/src/main/resources/xbean.xml` and uncomment the content in between the `<##### TELCOSERVICE Example ..>` tags. Please notice

that this provider must push the signed request to the SecureComm ASM consumer. Therefore the endpoint URL must point to the exposed SecureComm consumer endpoint URL as defined in step 1 of § 4.1.

5. Compile the results by executing the following command from the directory `${TELCO_ASM_NODE}`:

   ```
   $ cd ${TELCO_ASM_NODE}
   $ mvn clean install
   $ cd $ASM2_HOME && ./bin/servicemix &
   $ cd ${TELCO_ASM_NODE}
   $ cp asm1-sa/target/asm1-sa-1.0-SNAPSHOT.jar
   $ASM2_HOME/hotdeploy/asm1-sa-1.0-SNAPSHOT.zip
   ```

   This will produce the `asm1-sa/1.0-SNAPSHOT/asm1-sa-1.0-SNAPSHOT.jar` file that must be deployed in the ASM instance of step 1. You should see a successful deployment message after the copy command.

5.4.2Testing with soapUI

Before testing please make sure that both instances of ASM are running. Also make sure that Tomcat is functional. Familiarity with the SoapUI tool is a must in order to easily execute the test. The steps in brief are:

1. Create a new WSDL project for the test. Provide the URL to the WSDL of step 4 of § 5.1. The tool will automatically create the request. These are the requests as expected by the Web-Service. These we will replace further on with other content that will be transformed from the authenticator module to what the Web-Service expects.
2. Select the `getAccountInfo` request. In to top row replace the existing URL with the URL of the `http:endpoint` of the consumer as defined in step 3 of § 5.4.1
3. Replace the content of the request with the one provided below:

```
<soap:Envelope  xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
xmlns:xsd="http://qualipso.authenticators/xsd">
   <soap:Header/>
   <soap:Body>
      <xsd:getAccountInfo service="TelcoServiceXSLT">
        <xsd:operation>getAccountInfo</xsd:operation>
        <xsd:entryPoint>someEntryPoint</xsd:entryPoint>
        <xsd:phoneNumber>123456789</xsd:phoneNumber>
        <xsd:uname>kkontis</xsd:uname>
        <xsd:password>yoyo_pass</xsd:password>
        <xsd:startTimeFrame>3/8/2008</xsd:startTimeFrame>
        <xsd:endTimeFrame>5/8/2008</xsd:endTimeFrame>
      </xsd:getAccountInfo>
```

```
   </soap:Body>
</soap:Envelope>
```

You should get the following reply from the application

```
<soap:Envelope        xmlns:soap="http://www.w3.org/2003/05/soap-
envelope">
   <soap:Body>
      <xsd:getAccountInfoResponse
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://qualipso.authenticators/xsd">
         <xsd:call xsd:date="4/8/2008">595907741</xsd:call>
         <xsd:call xsd:date="5/8/2008">1732723979</xsd:call>
      </xsd:getAccountInfoResponse>
   </soap:Body>
</soap:Envelope>
```

4. Select the `resetPin` request. Replace the content of the request with the one provided below:

```
<soap:Envelope  xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
xmlns:xsd="http://qualipso.authenticators/xsd">
   <soap:Header/>
   <soap:Body>
      <xsd:resetPin service="TelcoServiceXSLT">
         <xsd:operation>resetPin</xsd:operation>
         <xsd:entryPoint>someEntryPoint</xsd:entryPoint>
         <xsd:phoneNumber>123456789</xsd:phoneNumber>
        <xsd:uname>kkontis</xsd:uname>
         <xsd:password>yoyo_pass</xsd:password>
      <xsd:newPin>someNewPin</xsd:newPin>
      </xsd:resetPin>
   </soap:Body>
</soap:Envelope>
```

You should get the following reply from the application

```
<soap:Envelope        xmlns:soap="http://www.w3.org/2003/05/soap-
envelope">
   <soap:Body>
      <xsd:resetPinResponse
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://qualipso.authenticators/xsd">
         <xsd:success>someNewPin</xsd:success>
      </xsd:resetPinResponse>
   </soap:Body>
```

```
</soap:Envelope>
```

Try to change the value of the password field. You should get an error message of “Permission denied”.

THE DAML2NRL USAGE EXAMPLE

In this section we provide a practical example of the concrete actions needed to add another functioning service in SecureComm. The DAML2NRL service is a stump service supporting a set of simple operations. The test client and the test server have different representations for credentials; the former follows DAML security schema and the latter the NRL credential and security. Those differences are bridged in SecureComm. The full description of the service and the security bridging performed by SecureComm is documented in wd3.2.4.

In the following sections we describe how to set-up the back-end server and the client of the DAML2NRL service and how to add the service bridging in the context of Apache Servicemix and the SecureComm.

5.5 Back-end Server Setup

By server setup we mean the real application to which the processed requests are sent from ASM. In the case of the DAML2NRL is again, like in Telco Service example, a Web Service providing a simple operation that understands NRL syntax security methods. This sub-section assumes the necessary software described in § 3.1 is installed. In order to prepare the server-part of the following steps must be executed:

1. Make sure that you have checked-out the web service under the directory `${NRL_WEB_SERVICE}`.
2. Into the `${NRL_WEB_SERVICE}` directory edit the `build.xml` file and change the value of the `TOMCAT_HOME` properties to your server values.
3. Recompile and deploy into Tomcat by executing the command from the checkout dir.

   ```
   $ ant tomcat.deploy
   ```
4. Copy endorsed libraries under your Tomcat installation

   Tomcat 5.X:

   ```
   $ cp -R ${DAML2NRL_FILES}/endorsed ${TOMCAT_HOME}/commons/
   ```

   Tomcat 6.X:

   ```
   $ cp -R ${DAML2NRL_FILES}/endorsed ${TOMCAT_HOME}/
   ```
5. Start Tomcat and verify correct installation by accessing the WSDL of the service from http://your-host.domain:18090/axis2/services/listServices

The URL of the WSDL will be used further on when defining the endpoint in the `securecomm-prov-http xbean.xml` configuration file, explained further on.

5.6 ASM Administration required actions

In order to add support for the DAML2NRL into SecureComm, the following things must be done, in accordance with the procedures described in section 4.1[3].

1. Define a consumer endpoint.

   In the file `${IMPLEMENT}/SecureCommRuntime-SA/securecomm-con-http/src/main/resources/xbean.xml` uncomments the content in between the `<##### DAML2NRL Example ..>` tags.

2. Define a provider endpoint.

   In the file `${IMPLEMENT}/SecureCommRuntime-SA/securecomm-prov-http/src/main/resources/xbean.xml` uncomment the content in between the `<##### DAML2NRL Example ..>` tags.

   If needed, please adjust the host, port of the Web-Service to your preferred values.

3. Redeploy SecureComm as stated in step 3 of § 4.1.
4. Make the service available as stated in step 4 § of 4.1.

5.7 Actions required for Semantic mapping definition

The protocol module for the DAML2NRL service is expressed on XSL style-sheets. The configuration is provided below.

5.7.1XSLT based solution

1. Make sure you have already the OWL and XSL files under the directory `${DAML2NRL_FILES}`. Files below are relevant to this path.
2. Execute step 1 of § 4.2
3. Select the DAML2NRL service in the step 2 of § 4.2
4. In step 3 of § 4.2 select XSL and upload the file XSLT/lifting-daml.xsl
5. Upload the ontologies found in the `ontologies/` directory. Ontology file `credential-security-daml.owl` is the source ontology, `bridgesecuritymodels-daml2nrl.owl` is the bridging ontology and `credential-security-nrl.owl` is the target ontology. Leave “Other ontologies” field empty. Select “Generic” bridging.
6. In step 6 of § 4.2 select XSL and upload the file XSLT/grounding-nrl.xsl
7. Submit the form and verify success as told in step 7 of § 4.2.

[3] You will notice that the artefacts downloaded from the project repository already contain the required configuration for the DAML2NRL Service example in comments.

5.8 Client Setup

On the client's part we need a tomcat client in order to do the testing. Furthermore, since all incoming request are expected to be digitally signed by X509 we need to sign the client requests before pushing them to SecureComm.

5.8.1 Signing the request

Repeat the configuration steps of paragraph 5.4.1, with the only exceptions of steps 3 and 4, where you have to uncomment the lines between the `<##### DAML2NRL Example ..>` tags.

5.8.2 Deploying and testing DAML web client

Before testing please make sure that both instances of ASM are running. Also make sure that Tomcat is functional.

1. Make sure that you have checked-out the web service under the directory `${DAML_WEB_CLIENT}`.
2. Into the `${DAML_WEB_CLIENT}` directory edit the `build.xml` file and change the value of the `TOMCAT_HOME` properties to your server values.
3. Recompile and deploy into Tomcat by executing the command from the checkout dir.

   ```
   $ ant tomcat.deploy
   ```
4. Start Tomcat and verify correct installation by accessing the service from http://your-host.domain:18090/DAMLWebClient/

Select one of the three categories request, choose the components that your request will carry and press the submit button. You will be presented with a page containing your formed DAML request and the response from the NRL server.

APPENDIX: CUSTOM JAVA CODE API

When the service has been deployed as described in § 4.2 and if any of the custom java code options has been selected, the respective Java method must have been implemented for SecureComm to pass the control.

Name	semanticGrounding	
Signature	public MessageExchange semanticGrounding(MessageExchange msgExchange, String outstr) throws Exception	
Input	msgExchange	The whole ASM normalised message
	outstr	Full path to the briding result file that will be used to parse and lift the results in order to construct the updated outgoing message
Output	MessageExchange	The updated message exchange. I now must contain the results of the semantic mappings as part of the body of the outgoing message.

Name	semanticBridging	
Signature	public void semanticBridging (String sourceOnto, String targetOnto, String bridgeOnto, String bridgeRepository, List otherOnto, String resultOfBridging) throws Exception	
Input	sourceOnto	Full path to the source ontology file
	targetOnto	Full path to the target ontology file
	bridgeOnto	Full path to the bridge ontology file
	bridgeRepository	Full path to the bridge ontology repository file
	otherOnto	A List containing full paths to other ontology files that are imported in the above ontologies.
	resultOfBridging	Full path to where will the result of bridging be saved.

Method Name	semanticLifting
Signature	public void semanticLifting(MessageExchange msgExchange, String sourceOntology, String

	sourceInstance) throws Exception	
Input	msgExchange	The whole incomming ASM message
	sourceOntology	Full path to the source ontology file.
	sourceInstance	Full path to where to save the new ontology file. It is now populated with the data parsed from the ASM incoming message holding the client request.

--- END of DOCUMENT ---