NIST HL7 Web Service Overview and Installation

NIST is developing a framework to test HL7 applications based on the concept of conformance profiles. The framework contains a core set of services that include message generation, message validation, profile validation, encoding transformations, and a test system. The package is written in Java and can be delivered as a set of Java APIs, applications built on top of the APIs, or web services. In this document we describe the message generation and message validation services and how they are deployed as web services. Below is an overview of the process with respect to the server and client side setup and web service installation:

Server Side:

1. Have available the set of services you wish to expose (e.g., message validation, etc).
2. Write a Java interface tailored to web services to expose a set of desired operations (e.g., MessageValidation.java)
3. Create a WSDL file for these services using *Java2WSDL* (e.g., Java2WSDL MessageValidation.class). The WSDL file describes the web service in a platform and programming neutral fashion.
4. Generate the service files using *WSDL2Java* and the WSDL file that was created in step 3 (e.g., *WSDL2Java* MessageValidation.wsdl). Among other resources this provides the Java skeleton source code files of the services.
5. Using the prewritten core set of APIs implement the skeleton files created by the *WSDL2Java* tool.
6. Using Apache Ant create the *aar* file (The *aar* is an archive file that contains all the artifacts required by the service).
7. Placed the aar file and other resources in the Axis2 service folder for deployment

(Note: The steps above have been completed by NIST; for users wishing to deploy the services locally step 7 is all that is needed—see step 5 of Installation).

Client Side:

1. Using a WSDL tool on the client for the appropriate platform (e.g., wsdl on .net) create the client side interface code (e.g., C# interface)
2. Write applications using the interface code created in the previous step.

Installation:

1. Install Apache Tomcat
2. Install Apache Axis2
3. Install Apache Ant
4. Install an Axis 2 service on Tomcat
5. Add resource files needed by the web service (e.g., aar files, libraries, etc).

Modifications

Name	Issue	Modifications	Date
Roch Bertucat	0.8	Axis 2 version 1.2 update	06.18.2007
Roch Bertucat	0.7	Files dependencies update	01.19.2007
Roch Bertucat	0.6	Add information paragraph 2.4	10.13.2006
Robert Snelick	0.5	Overview text and general editing	10.11.2006
Roch Bertucat	0.4	Methods description + link to javadoc	10.6.2006
Roch Bertucat	0.3	Session Management & C# client configuration	10.4.2006
Len Gebase	0.2	Review and completions	9.24.2006
Roch Bertucat	0.1	Server and client configuration	9.6.2006

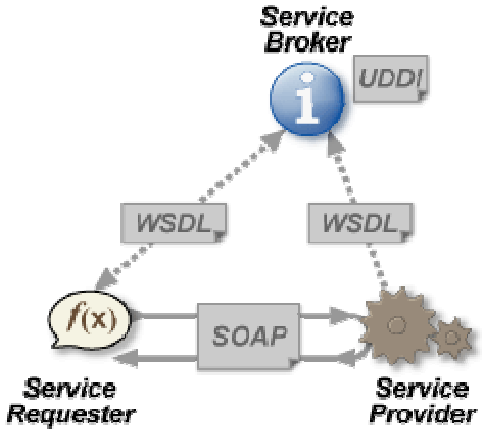
Table of Contents

1 Definitions

Web Service

The W3C defines a Web service as a software system designed to support interoperable machine-to-machine interaction over a network.

Because this definition encompasses many different systems, in common usage the term usually refers to those services that use SOAP-formatted XML envelopes and have their interfaces described by WSDL.



SOAP

An XML-based, extensible message envelope format, with "bindings" to underlying protocols (e.g., HTTP, SMTP and XMPP).

WSDL

An XML format that allows service interfaces to be described, along with the details of their bindings to specific protocols. Typically used to generate server and client code, and for configuration.

UDDI

A protocol for publishing and discovering metadata about Web services, to enable applications to find Web services, either at design time or runtime.

2 Installation

We use Java version **1.5**.

2.1 Apache Tomcat

First, a web server has to be installed. We use apache tomcat. It can be downloaded at: http://tomcat.apache.org/.
Version used: 5.5.17

2.2 Apache Axis2

Next, for supporting SOAP services, axis2 must be installed from: http://ws.apache.org/axis2/. For details on how to write a web service, refer to the axis2 user guide, available at: http://ws.apache.org/axis2/1_0/userguide.html.
Version used: 1.2

You should be able to install and configure properly Axis2 before continuing. You should also know the different ways to implement and deploy a simple service (especially how to deploy a service with the archive file .aar).

2.3 Apache Ant

http://ant.apache.org/
You should know how to execute and modify a build file.

2.4 Installing the service from an .aar file

2.4.1 Copy the .aar file

Just drop the .aar file in the directory %CATALINA_HOME%\webapps\axis2\WEB-INF\services (CATALINA_HOME is the directory in which Tomcat is installed) to deploy the service.

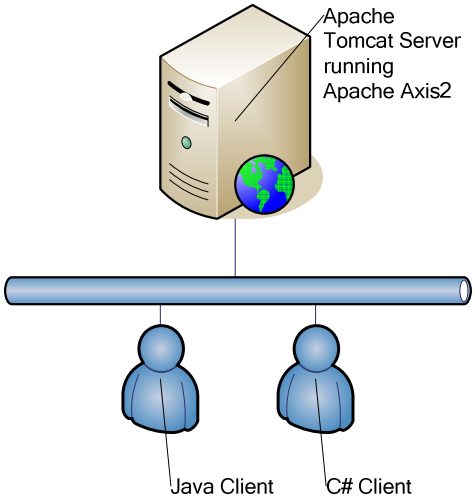
2.4.2 Copy NIST libraries and resources

Some libraries and resources required to run the NIST core module functionalities have to be added on the Axis2 module installed in the Tomcat directory:

- *core.jar, DataSource.jar, DataValidationContext.jar, hsqldb.jar, jsr173_1.0_api.jar, MessageValidationContext.jar, ProfileValidationContext.jar, Report.jar, saxon8.jar, SequenceNumber.jar, TableValue.jar,ValidationContext.jar, xalan.jar, xbean.jar, xbean_xpath.jar, xercesImpl.jar, xml-apis.jar* have to be copied in the directory %CATALINA_HOME%\webapps\axis2\WEB-INF\lib.
- Data files (contained in the directory *data*\) have to be included directly in the directory %CATALINA_HOME%\ respecting the original directory structure.

If you don't have any .aar file, you have to build the service.

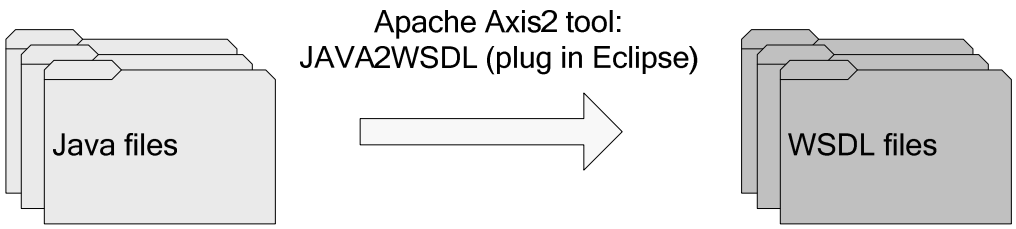
3 Building a service with Apache Axis2



We use the tools called Java2WSDL and WSDL2Java provided by Axis2. These tools can be integrated directly as a plug-in into Eclipse (follow the installation directions on: http://ws.apache.org/axis2/tools/1_2/eclipse/wsdl2java-plugin.html).

Eclipse is a Java Integrated Development Environment. If you are not an Eclipse user, you can use *Java2WSDL* and *WSDL2Java* directly using the command line (see 3.1 and 3.2).

3.1 Generating a WSDL file



We want to generate the WSDL file for the MessageValidation.java interface shown below (note that the WSDL file is only a standard representation of the data types used, so the implementation of the methods doesn't matter for now). We'll use *Java2WSDL* to generate the file.

```
/*
 * NIST HL7 Web Service
 * MessageValidation.java Aug 29, 2006
 *
 * This code was produced by the National Institute of Standards and
 * Technology (NIST). See the "nist.disclaimer" file given in the
distribution
```

```
 * for information on the use and redistribution of this software.
 */

/**
 *
 */
package gov.nist.hl7.ws.validation;

/**
 * This interface provides a simple interface for validating a message
against a profile.
 * This interface is intended for use in a web-service environment.
Parameters and return
 * values are intentionally made to be simple objects.
 * <p>
 * Currently only basic validation functionality is provided. The
capibability for
 * advanced validation services is planned. For example, the user will be
able to set the
 * context in which the message is validated. This functionality already
exists but will
 * need to be exposed in this interface.
 *
 * @author Robert Snelick (NIST)
 *
 */
public interface MessageValidation {

    /**
     * Set the Profile
     * @param xmlProfile a Profile encoded as an XML String
     * @param profileId an id to identify the Profile (value used in the
report)
     * @return true if the Profile has been set
     */
    public boolean setProfile(String xmlProfile, String profileId);

      /**
       * Set the Message
       * @param message a Message encoded as an XML String. The message
can be an
       * XML or ER7 String. An ER7 message will be converted into its XML
representation for
       * processing.
     * @param isXML Set to true if the <code>String</code> is an XML
representation of the
     * message; false if the <code>String</code> is an ER7 representation
of the message.
       * @return true if the message was successfully read and processed;
false otherwise.
       */
      public boolean setMessage(String message, boolean isXML);

    /**
     * Set the MessageValidationContext
     * @param xmlMessageValidationContext a MessageValidationContext
encoded as an XML String
```

```
     * @return true if the MessageValidationContext has been set
     */
    public boolean setMessageValidationContext(String
xmlMessageValidationContext);

     /**
      * Validate the message against a profile. The return value is true
if the message is
      * valid with respect to the profile and the validation context. The
validation context
      * describes the validation tests that are performed. The return
value of false is returned if
      * the validation fails or if the profile or message is not set
correctly.
      * <p>
      * @return true if the message is valid with respect to the profile
and the validation
      * context; return false otherwise.
      */
     public boolean validate();

    /**
     * Get the MessageValidationReport associated with the previous
validation
     * @return a String object that contains an XML report of the message
validation results
     */
    public String getMessageValidationReport();

    /**
     * Get the messsage of the last exception caught
     * @return a String containing the last exception message
     */
    public String getLastExceptionMessage();
}
```

To generate the WSDL file, we need the file named MessageValidation.java. The file has to be placed in the directory gov\nist\hl7\ws\validation (on unix platforms replace \ with /) under your Java class path directory. The file must then be compiled.

After the MessageValidation.java file has been compiled, the *Java2WSDL* tool is run on the class file (*Java2WSDL* MessageValidation.class creates MessageValidation.wsdl). With Eclipse, the WSDL plug-in can be found by selecting File->New->Other (further details can be found by following the link above for installing the plug-ins).

For non Eclipse users, you can use the script files java2wsdl.bat or java2wsdl.sh. These files are located in the bin\ directory under the directory where Axis2 was installed. To run either script the environment variables AXIS2_HOME and JAVA_HOME must be set correctly. The shell script can be run from the Axis2 bin directory as provided, but to run it elsewhere the variables must be set. An example invocation of the scripts, along with a description of the script options, is provided in the Axis2 user guide referenced above.

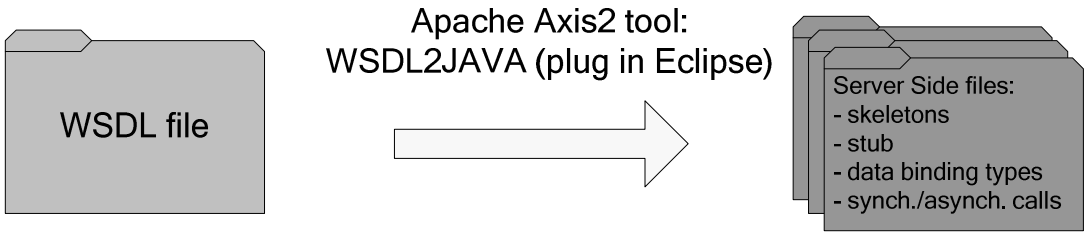
3.2 Generating the Service Files from the WSDL file

When the WSDL file is generated, we extract the service files with the tool *WSDL2Java*.

This can be done through Eclipse or with the script files wsdl2java.bat or wsdl2java.sh (See 3.1).

Invoking the script as shown in the example with our WSDL file replacing the one in the example, results in the creation of a number of files. The files contain configuration information, a build file and Java classes to access the web service, create client and server side *adapter* code, and additional resources.

The Java source files that are generated will have to be moved before they will compile. The package name at the top of the files should be used to determine the correct location for the files. The build file and the resources directory should be moved above the root of the source code directory. The exact location is determined by the value of the property named "src" in the build file.



The Java class files allow the user to configure, describe the service and access it synchronously or asynchronously. We describe the service in a file called skeleton.

3.2.1 Implementing the Skeleton file

The tool *WSDL2Java* generates several files. One of them is called the skeleton and implements the service executed on the server side.

```
/**
 * MessageValidationSkeleton.java This file was auto-generated from WSDL
by the Apache Axis2 version: 1.0 May 04, 2006 (09:21:04 IST)
 */
package gov.nist.hl7.testframework.testws.messagevalidation;

/**
 * MessageValidationSkeleton java skeleton for the axisService
 */
public class MessageValidationSkeleton
    implements MessageValidationSkeletonInterface {
    /**
```

```
     * @param param8
     */
    public
gov.nist.hl7.testframework.testws.messagevalidation.types.SetMessageRespon
se setMessage(

gov.nist.hl7.testframework.testws.messagevalidation.types.SetMessage
param8) {
        //Todo fill this with the necessary business logic
        throw new java.lang.UnsupportedOperationException();
    }

    /**
     * @param param10
     */
    public
gov.nist.hl7.testframework.testws.messagevalidation.types.ValidateResponse
validate(
        gov.nist.hl7.testframework.testws.messagevalidation.types.Validate
param10) {
        //Todo fill this with the necessary business logic
        throw new java.lang.UnsupportedOperationException();
    }

    /**
     * @param param12
     */
    public
gov.nist.hl7.testframework.testws.messagevalidation.types.GetValidationRep
ortResponse getValidationReport(

gov.nist.hl7.testframework.testws.messagevalidation.types.GetValidationRep
ort param12) {
        //Todo fill this with the necessary business logic
        throw new java.lang.UnsupportedOperationException();
    }

    /**
     * @param param14
     */
    public
gov.nist.hl7.testframework.testws.messagevalidation.types.SetProfileRespon
se setProfile(

gov.nist.hl7.testframework.testws.messagevalidation.types.SetProfile
param14) {
        //Todo fill this with the necessary business logic
        throw new java.lang.UnsupportedOperationException();
    }
}
```

The MessageValidationSkeleton.java file shown above should have been generated under the src directory when *WSDL2Java* was run, and it should have been moved to the appropriate directory. The user has to explicitly move all the files (created by *WSDL2Java*) to

the directory with the same package named indicated in the web service interface file (MessageValidation.java; in this case the directory is gov\nist\hl7\ws\validation).

3.2.2 Modifying the Axis2 Session Management

Axis2 uses "services.xml" to hold the configurations for a particular web service deployed in the Axis2 engine. This file is generated by the *WSDL2Java* tool and placed in the directory resources.

By design, Web services are said to be stateless. This means that local class attributes set when a web service is called are not persistent, i.e., if a subsequent call is made, values set in the previous call are lost. Even though web services are stateless, to support some applications, maintaining some state information is necessary. To address this problem, Axis2 supports session management. With this approach local variables still cannot be used to maintain state, but the global state of the service can be maintained.

Listed below are the different levels of session that provides Axis2:

1. Request
   This is the default type of session by Axis2. The lifetime of this session is limited to the method invocation's lifetime, or the request processing time.

2. SOAPSession
   Managing a SOAP session requires both the client and service to be aware of the sessions; in other words, the client has to send the session-related data if he wants to access the same session and the service has to validate the user by using session-related data. So both clients and server have to configure the SOAP message to add the session information.

   This scope allows the data on the server side to be persistent.

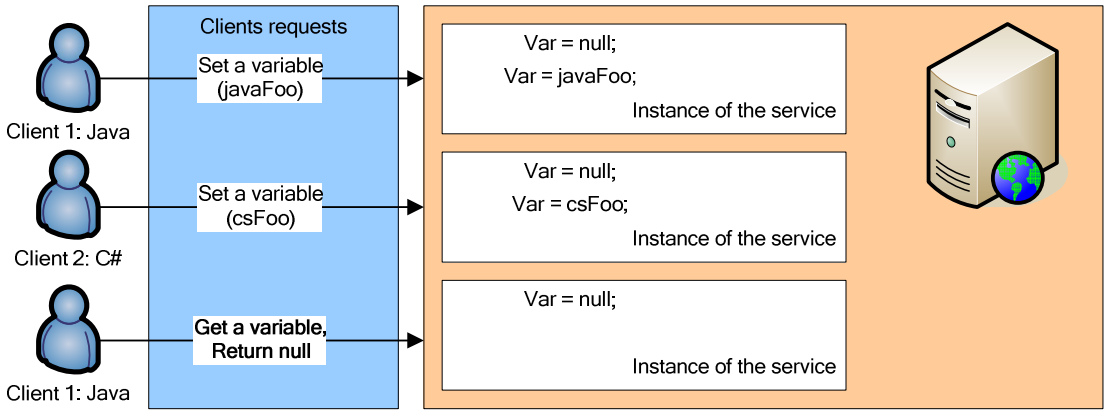
3. Transport
   In the case of a Transport session, Axis2 uses transport-related session management techniques to manage session. As an example, in the case of HTTP, it uses HTTP cookies to manage the session. The lifetime of the session is controlled by the transport, not by Axis2.

   This scope allows the data on the server side to be persistent. This type seems to be the one we need either to manage the users or to be interoperable with other environment (only a cookie to manage).

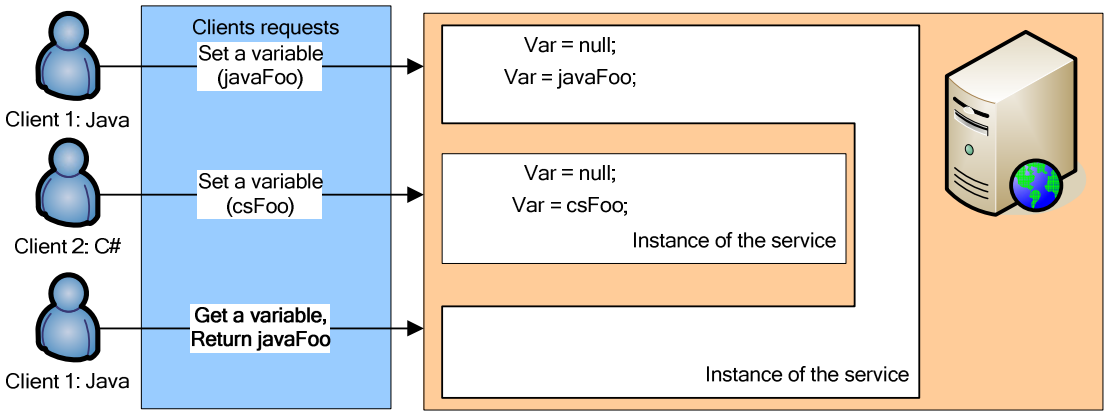
4. Application
   Application scope has the longest lifetime compared to others; the lifetime of the application session is equal to the lifetime of the system. If you deploy a service in application scope, there will be only one instance of that service.

The next figures below show the different scope from the client side.

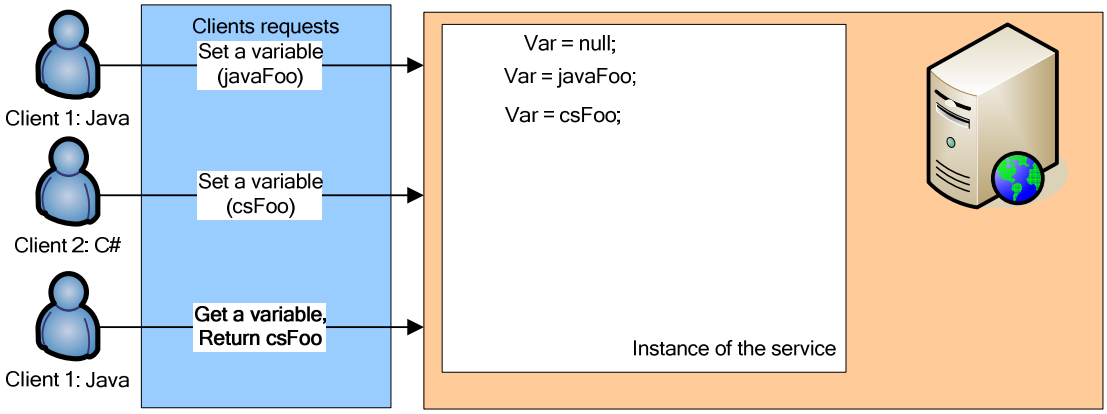
① Request



② SOAP Session / Transport Session



③ Application



To change the scope of the service, modify the service.xml file. Setting the scope of the service element as shown below allows the same service to be provided to all users when more than one user simultaneously accesses the service.

```
<!-- This file was auto-generated from WSDL -->
<!-- by the Apache Axis2 version: #axisVersion# #today# -->
<serviceGroup>
<service name="MessageValidation" scope="transportsession">
...
</service>
</serviceGroup>
```

3.2.3 Configuring the Build File

The default build file generated by *WSDL2Java* has to be changed to include source files and libraries used in the core package. This can be done by copying the following XML elements and pasting them in the build file.

```
<property name="hl7.base.dir" value="./.. "/>
<property name="project.base.dir" value="."/>
<property name="maven.class.path" value=""/>
<property name="name" value="MessageValidation"/>
<property name="src" value="${project.base.dir}/src"/>
<property name="test"
value="${project.base.dir}/test"/>
<property name="build"
value="${project.base.dir}/build"/>
<property name="classes" value="${build}/classes"/>
<property name="lib" value="${build}/lib"/>
<property name="lib2" value="${hl7.base.dir}/lib"/>
...
<path id="axis2.class.path">
<pathelement path="${java.class.path}"/>
<pathelement path="${maven.class.path}"/>
<fileset dir="${axis2.home}">
<include name="lib/*.jar"/>
</fileset>
<fileset dir="${lib2}">
<include name="*.jar"/>
</fileset>
</path>
...
<javac debug="on" destdir="${classes}">
<src path="${src}"/>
<classpath refid="axis2.class.path"/>
</javac>
...
```

hl7.base.dir is the root directory of the package.

lib2 is the path to the libraries.

The included path has to include the core libraries.

3.2.4 Building the core.jar

The file core.jar has to be built and added to the class path of the working project. With *ant*, we execute the build file with the option *jar* on the directory *\hl7\core*. The core jar is created into the folder *jar/*.

3.2.5 Executing the Build File

With *ant*, we execute the build file. Invoking *ant* without any arguments in the directory where the build file is located results in *ant* executing the default target. This produces several output files in the directory *build/*.

The aar file contained in the folder *build/lib* is the one describing the service. This file has to be copied in the directory %CATALINA_HOME%\webapps\axis2\WEB-INF\services.

You can now deploy the service file (see 2.4).

3.2.6 Debugging with Axis2 and Tomcat

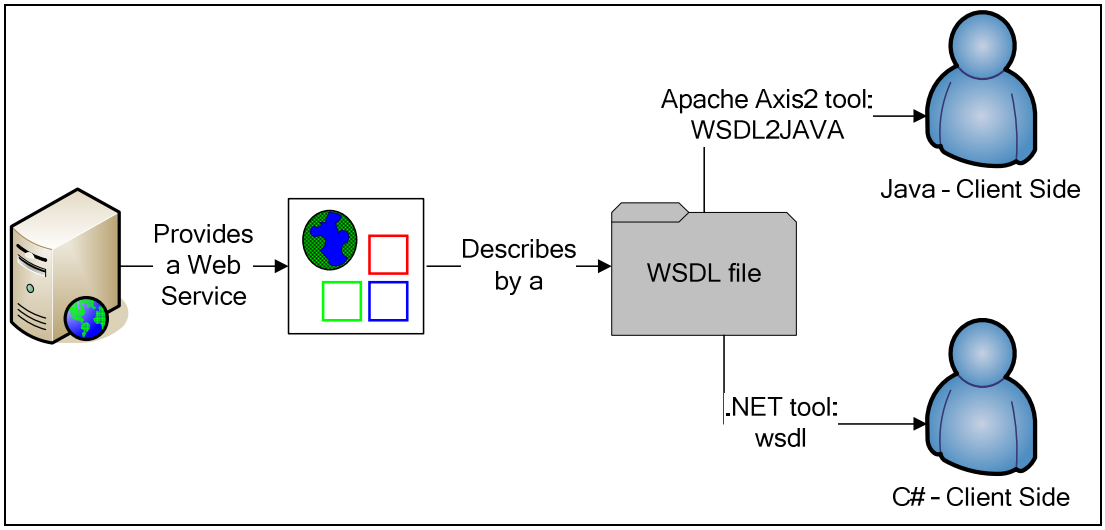
You can use two very useful tools to debug your service with Tomcat, Eclipse and Axis2.

A tool called SOAP Monitor provides a way for service developers to monitor the SOAP messages being used to invoke Web services along with the results of those messages. This tool is provided by Axis2.

Tomcat can be configured in debug mode. If you configure Eclipse properly, you can debug your running service directly with the very powerful debug interface provided by Eclipse.

4 Connecting Clients to an Axis2 Service

4.1 Overview and Available Methods



Currently the NIST web service exposes Message Validation and Profile Validation functionality. The features for these services are scaled down in the current version. The interface for these services use simple data types (e.g., String) and we avoid complex operations. Multiple calls are required to perform a desired action. These simplifications are intentional to make testing and deployment easier. Additional functionality can be added later. For complete details of the services currently offered, see the javadoc.

4.1.1 Message Validation

Available operations

- setMessageValidationContext
- getLastExceptionMessage
- setMessage
- getMessageValidationReport
- validate
- setProfile

4.1.2 Profile Validation

Available operations

- getLastExceptionMessage
- validate
- setProfileValidationContext
- setProfile

- getProfileValidationReport

4.2 Java Client

From a WSDL file, we can produce the files required to access the service. For example, the tool *WSDL2Java* generates java files that can be added directly to an existing project.

4.3 C# Client

4.3.1 Generalities

From a WSDL file, we can produce the files required to access the service. The tool 'wsdl' provided by Microsoft .NET generates C#, C++ or VB files that can be added directly to an existing project.

Here is an extract of the C# file produced by the tool. This file can be used to access our service.

```
//------------------------------------------------------------------------------
-
// <autogenerated>
//     This code was generated by a tool.
//     Runtime Version: 1.1.4322.2032
//
//     Changes to this file may cause incorrect behavior and will be lost if
//     the code is regenerated.
// </autogenerated>
//------------------------------------------------------------------------------
-

//
// This source code was auto-generated by wsdl, Version=1.1.4322.2032.
//
using System.Diagnostics;
using System.Xml.Serialization;
using System;
using System.Web.Services.Protocols;
using System.ComponentModel;
using System.Web.Services;


/// <remarks/>
[System.Diagnostics.DebuggerStepThroughAttribute()]
[System.ComponentModel.DesignerCategoryAttribute("code")]
[System.Web.Services.WebServiceBindingAttribute(Name="MessageValidationSOAP11Bi
nding",
Namespace="http://MessageValidation.testws.testframework.hl7.nist.gov")]
public class MessageValidation :
System.Web.Services.Protocols.SoapHttpClientProtocol {

    /// <remarks/>
    public MessageValidation() {
        this.Url = "http://127.0.0.1:8080/axis2/services/MessageValidation";
```

```
    }

    /// <remarks/>

[System.Web.Services.Protocols.SoapDocumentMethodAttribute("urn:setMessage",
Use=System.Web.Services.Description.SoapBindingUse.Literal,
ParameterStyle=System.Web.Services.Protocols.SoapParameterStyle.Bare)]
    [return: System.Xml.Serialization.XmlElementAttribute("setMessageResponse",
Namespace="http://MessageValidation.testws.testframework.hl7.nist.gov/types")]
    public setMessageResponse
setMessage([System.Xml.Serialization.XmlElementAttribute("setMessage",
Namespace="")] setMessage setMessage1) {
        object[] results = this.Invoke("setMessage", new object[] {
                    setMessage1});
        return ((setMessageResponse)(results[0]));
    }

...

}
```

4.3.2 How to solve some .NET interoperability issues with Tomcat?

Code generated from the Microsoft .NET's "wsdl" tool

By default, the wsdl tool will generate some C# file used to connect to the remote server. It appears that Apache Axis2 won't handle some "Namespace" attributes that .NET provides in its SOAP message requests.

The solution is to remove the namespaces in the parameters declarations `(System.Xml.Serialization.XmlElementAttribute)`.

Session Management

As described above in the document, we use the transport session to manage the users accessing our service. The .NET client has to be aware that it needs to collect a cookie and send it back to the service.

A simple modification can be done when we declare the instance of the connector with the service.

```
/// <summary>
/// Class constructor.
/// </summary>
/// <param name="validationWebServiceUrl">NIST URL</param>
public NistWebServiceClient(System.String validationWebServiceUrl)
{
      _hl7MessageValidation = new
Dvtk.IheActors.Hl7.WebService.Validation.MessageValidation(validationWebServiceUrl);
      _hl7MessageGeneration = new
Dvtk.IheActors.Hl7.WebService.Generation.MessageGeneration(validationWebServiceUrl);

      _hl7MessageValidation.CookieContainer = new CookieContainer();
      _hl7MessageGeneration.CookieContainer = new CookieContainer();
}
```

4.4 Other Clients

Theoretically, from any WSDL file, a user can generate access files in any language.

5 References

http://tomcat.apache.org/

http://ws.apache.org/axis2/

http://ant.apache.org/

http://www.w3.org/2002/ws/

http://en.wikipedia.org/wiki/Web_service

http://ws.apache.org/axis2/tools/1_2/eclipse/wsdl2java-plugin.html
Eclipse plugging JAVA2WSDL and WSDL2JAVA

http://www.developer.com/java/web/article.php/3620661
Axis2 Session Management

http://wso2.org/library/225
Tomcat setup for debugging & Eclipse configuration for debugging an axis2 webservice

http://ws.apache.org/axis2/1_2/soapmonitor-module.html
SoapMonitor configuration

http://msdn2.microsoft.com/en-us/library/7h3ystb6(VS.71).aspx
Microsoft .NET's WSDL tool