**1. Introduction**

In continuing our exploration of various programming models for distributed systems, we will now consider Web services.

**Objectives.** After completing this programming assignment, you will be able to:

1. Understand the basic mechanisms by which to develop Web service clients and Services on three of the major Web services platforms:
   - Visual Studio 2005 (C#)
   - AXIS2 (Java)
   - gSOAP (C/C++)
2. Gain experience with cross-platform development
3. View raw SOAP messages and WS-Security messages

You **MUST** do this assignment by yourself. While I definitely see the value of working in groups, this particular assignment is a better learning experience if you do it by yourself. Note: if a particular step in the assignment is “not working”, you can certainly ask your colleagues for help. Of course, “help” does not mean that they tell you the answer, rather that the help you understand what’s going wrong.

Because this is in the style of a “Hands-On Lab”, I **strongly** encourage you to complete this lab in one sitting. If you are unable to complete it in one sitting, you may not be able to easily continue where you left off.

The instructions assume that you’re using a machine in 002a, although you can use your own Windows XP/Vista machine if you have Visual Studio 2005 installed (Note: this assignment requires two substantial downloads/installations, so you might just want to use the lab machines). You will also need to use a Linux machine (such as the “lava” machines – use “lava001”, for example).

Also, you are NOT expected to type in these code segments by hand (instead, cut-and-paste them); but PLEAS make sure that you understand what each step is doing! **NOTE: sometimes cut-and-paste inserts “strange characters” – make sure that the code that is inserted is the same as what appears in the PDF.**

Expected time: 3 hours.

**2. Web Services Development on the .NET Framework 3.0**

In this first part of the assignment, we will be using the .NET Framework v3.0 from Microsoft (released Nov 21, 2006). We’ll need both the .NET run-time and the SDK:

1. If you’re using machine in 002a, and the machine is not booted into Windows XP, reboot (when the “fedora” window appears, you have THREE SECONDS to hit the down arrow so that you can
select the OS to boot. Then, select Windows XP, and select the first “Windows XP” again on the next screen).

2. The .NET 3.0 run-time and the SDK should have already been installed for you. To check:
   a. **.NET 3.0 run-time**: look for "Microsoft .NET Framework 3.0" in the "Add or Remove Programs" of the Control Panel. If needed, install via the Web (google for “.net framework 3.0 redistributable”)
   b. **SDK**: look for “Microsoft Windows Software Development Kit (6000.0.0)” in the "Add or Remove Programs" of the Control Panel. If needed, install via the Web (google for “windows sdk for vista”). Make sure the version is “Orcas_March07CTP” (11/7/2006).

### 2.1 Web Services Development and Deployment on the .NET Framework using Visual Studio 2005

In this exercise, you will create a simple Web service and client using the .NET Framework 3.0 Windows Communication Foundation (WCF).

1. Log into ITC’s home directory – this is where you’ll keep your Visual Studio files.
   a. START → My computer and navigate to c:\program files\Home Directory Login v1.0.9 and run “Home” (I will assume that this drive is mapped as “J:”)

2. Boot Visual Studio 2005:
   a. START → All Programs → Microsoft Visual Studio 2005 → Microsoft Visual Studio 2005 (set “Visual C# Development Settings” as the default)

3. Create a C# Console Application:
   a. FILE → new → Project
   b. Select Visual C# Projects in the Project Types pane.
   c. Select Console Application in the Templates pane.
   d. In the Name box, enter CS451PA3Service
   e. In the Location box, enter “J:\”
   f. Click OK

   If the dialog box says “The project location is not trusted”, send me email. This should not occur.

4. Add a reference to the WCF assembly:
   a. In the Solution Explorer, right-click on the References node, under the CS451PA3Service project node, and select Add Reference.
   b. Via the “Browse” tab, navigate to c:\WINDOWS\Microsoft.NET\Framework\v3.0\Windows Communication Foundation and add System.ServiceModel.dll

5. Replace the existing code in Program.cs with the following (you can cut-and-paste from the PDF, and then just manually insert newlines):

```csharp
using System;
using System.Collections.Generic;
using System.Text;
using System.ServiceModel;
namespace HelloService
{
    class Program
    {
        static void Main(string[] args)
        {
```
Uri baseURI = new Uri("http://localhost:8099/hello");
ServiceHost HService = new ServiceHost(typeof(HelloService), baseURI);
HService.AddServiceEndpoint(typeof(HelloService),
    new BasicHttpBinding(),
    baseURI);

HService.Open();
Console.WriteLine("Service at your service.");
Console.ReadKey();
HService.Close();
}

[ServiceContract]
class HelloService
{
    [OperationContract]
    string sayHi()
    {
        return ("Hi!");
    }
}

What does this do? The class HelloService holds the functionality of the service—the [ServiceContract] attribute indicates that the class will be exposed as a service on the wire, and the attribute [OperationContract] indicates that the sayHi() function will be exposed as a method on that service.

The Program class holds the infrastructure—the ServiceHost class spins up all the necessary hosting infrastructure for the service. The service endpoint offers a way to communicate with the service—it’s analogous to how an interface works on an object.

6. Replace the line “Service at your service” with “YYYY Service at your service”, where “YYYY” is replaced by your name (“Humphrey”, etc.)

7. Build the Web service:
   A. From the Visual Studio, hit "Build ➔ Build Solution", and correct any errors.
   B. Confirm that the Web service application runs via right-click on “CS451PA3Service” in the Solution Explorer, and select “Debug ➔ Start new instance”. If this is successful, a console application should pop up and “Service at your service” should appear.

8. This application is listening for messages at the address “http://localhost:8099/hello”. To test this, aim Internet Explorer at this address (while the app is running). If everything is working, Internet Explorer should display a page entitled “Service” and should say that “Metadata publishing for this service is currently disabled”. This indicates that the Service is NOT yet exposing its service interface to the world. Let’s fix that now, roughly according to the instructions on the Web page displayed in Internet Explorer.
   A. Select this Service console window (the “group service at your service.” Window), and then hit “Enter” to make the application terminate.
   B. Comment out the (programmatic) endpoint declaration in Program.cs of CS451PA3Service:
// HService.AddServiceEndpoint(typeof(HelloService),
// new BasicHttpBinding(), baseURI);

C. Instead of programmatically declaring the service properties, we’re going to use the WCF declarative mechanism: Right-click on the CS451PA3Service in the Solution Explorer, and “Add → New Item” and select “Application Configuration File” (the default name of “App.config” should be used). Replace the entire contents with the following code:

```xml
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <services>
      <service name="HelloService.HelloService"
        behaviorConfiguration="MyServiceTypeBehaviors">
        <endpoint address="http://localhost:8099/hello"
          binding="basicHttpBinding"
          contract="HelloService.HelloService" />
        <endpoint contract="IMetadataExchange"
          binding="mexHttpBinding"
          address="mex" />
      </service>
    </services>
    <behaviors>
      <serviceBehaviors>
        <behavior name="MyServiceTypeBehaviors">
          <serviceMetadata httpGetEnabled="true" />
        </behavior>
      </serviceBehaviors>
    </behaviors>
  </system.serviceModel>
</configuration>
```

D. Re-compile and re-run the application. If you are successful, then a console app should appear and again display “YYYY Service at your service” (Note: you might have to shut down Visual Studio and restart it to get this step to work).

E. With the service running, refresh the IE page. If you are successful, the page should mention something about “svcutil.exe”. This is indicating that your service is publishing its service interface.

To hand in for this section:

1. Start an MS Word Document (via START → All Programs → Microsoft Office → Microsoft Office Word 2003”
2. Type in “CS 451 PA3” followed by your name(s).
3. Select the Service console window and hit the “Print Screen” button (upper-right of the keyboard). Note: yes, you must capture the entire screen, not just the console window.
4. Select the Microsoft Office doc, type in “Part 2.1”, hit “enter”, and then hit “shift-insert”.
5. Save this doc on J:

2.2 Writing the .NET Client that consumes the Web Service

In this exercise, you will create a simple .NET client that consumes the Web service created in Exercise 2.1
9. Halt the service by selecting the Service console window and hitting “enter”
10. Add a “TestClient” project to the CS451PA3Service solution by right-clicking on the CS451PA3Service solution node in the Visual Studio Solution Explorer window, select Add, New Project, C# "Console Application”… Name it Client, hit return
11. In the Client project, add a reference “System.ServiceModel.dll” (click on the “Recent” tab)
12. Creating the proxy code for the CS451PA3Service:
   A. Start → Microsoft Windows SDK → CMD Shell
   B. Cd to your J:\CS451PA3Service\Client directory, and execute the following (note: the service must be running):

   

   This will generate two files for you in this directory: “app.config” (which is a configuration file for the client that indicates how to interact with the service), and “HelloService.cs” (which is the proxy code for the service).
   
   C. Select the Service console window, and then hit “Enter” to make the application terminate.
   D. Add these two files to the Client project (right-click on the “Client” project, and select “Add → Existing Item”; change “Files of type” to be “All Files”, and add “app” and “HelloService”, one at a time).
   E. Change the “Program.cs” of the Client project to be the following:

   ```
   using System;
   using System.Collections.Generic;
   using System.Text;
   using System.ServiceModel;
   namespace Client
   {
       class Program
       {
           static void Main(string[] args)
           {
               HelloServiceClient p = new HelloServiceClient();
               Console.WriteLine("Humphrey .NET client running.");
               Console.WriteLine(p.sayHi());
           }
       }
   }
   ```
   
   F. Replace the “Humphrey” in the “Console.WriteLine” statement with your name.
   G. Rebuild the solution
   H. Re-run the service (note that the window might be buried)
   I. Run the client by doing the following: Open a new command prompt (start → all programs → accessories → command prompt), and cd into the CS451PA3Service\Client\bin\Debug directory, and execute “Client.exe”. If you’re successful, “Hi!” should be printed on the screen.

To hand in for this section:

1. Select the Client console window (Command Prompt) and hit the “Print Screen” button.
2. Add this as “Part 2.2” of your MS Word doc.
2.3 Reconfiguring the Service to Log messages

In this section, we’ll get the service to log a subset of its messages.

1. Select the Service console window, and then hit “Enter” to make the application terminate.
2. We could edit the service’s “App.config” directly, but Microsoft provides a tool, which we’ll use instead. First, close Visual Studio. Then, in the “Windows SDK CMD window”, execute “svcconfigeditor”
3. File → Open → Config file, to open the service’s “App.config”
   a. Click on the “Diagnostics” folder and click on the link that says “Enable MessageLogging”
   b. Then, open up the “Diagnostics” folder, and look at the “Message Logging”. Click on the “False” next to “LogEntireMessage” and change it to “True”.
   c. File → Save (note: you’ll have to save it as “App_new.config” because of file permissions issues)
   d. Exit
   e. In a command prompt, navigate to J:\CS451PA3Service\CS451PA3Service and execute “move App_new.config App.config”
4. Re-start Visual Studio by double-clicking on the solution.
5. Re-run the service and re-run the client to confirm that it still works. (Note: there is no need to recompile anything).
6. Select the Service console window, and then hit “Enter” to make the application terminate.
7. In the “Windows SDK CMD window”, execute “svctraceviewer”
   a. File → Open and then open “App_messages.svclog” in the service’s folder.
   b. Click on the “000000000000” entry under the “Activity” title on the left of the screen.
   c. Click on the first message, and then click on the “XML” tab.

To hand in for this section:

1. Select the “Microsoft Service Trace Viewer” window and hit “ALT-Print Screen” button.
2. Add this as “Part 2.3” of your MS Word doc.

2.4 Securing the message exchange

In this section, we’ll reconfigure the service and client to secure the message exchange.

1. If the “Microsoft Service Trace Viewer” is still running, kill it.
2. Delete the App_messages.svclog in the service directory.
3. In the service’s App.config, change binding=”basicHttpBinding” to binding=”wsHttpBinding”. There is no need to recompile, because the app reads this config file every time it runs.
4. Now the client must be updated to match the new service configuration. It’s easiest to re-run the “svcutil” on the client side to update the client.
   a. Delete “HelloService.cs” and “app.config” from the Client’s project.
   b. Start the service
   c. Re-run the “svcutil” command as specified above.
   d. Stop the service
   e. Re-add “HelloService.cs” and “app.config” to the Client’s project.
5. Recompile the solution, restart the service, and re-run the client.
6. Stop the service.(Note that this might appear to hang – be patient).
7. Re-inspect the log via the technique described above.

To hand in for this section:

1. From the log, how many messages are exchanged?
2. What is the purpose of each message?(that is, what specification does it conform to?)
3. Is this message-level security, transport-level security, or both?
4. What is the encryption method used?
5. What is the value that is finally returned from the service to the client?

Select the “Microsoft Service Trace Viewer” window. Select the LAST message logged, show the XML tag, and scroll to the point where your answer to question 5 is shown, and hit “ALT-Print Screen” button. Add this as “Part 2.4” of your MS Word doc, after your answers to the 5 questions above.

3. Web Services Development on Linux with Apache Axis2

In the second part of the assignment, we'll compare the development experience gained in the first part on Windows XP via Visual Studio with a development experience on Linux using Apache Axis2. Note that to keep things a little simpler, we're not going to use an IDE on Linux (although there are many available).

3.1 Web Services Deployment on AXIS

1. Log onto one of the Lava machines (you were given an account on these at the beginning of the semester). START → All Programs → SecureCRT 5.0
2. mkdir PA3; chmod 700 PA3; cd PA3
3. wget http://www.cs.virginia.edu/~humphrey/cs451/Assignments/Assignment3/axis2-1.1.zip
4. unzip axis2-1.1.zip
5. create the following Java file, and name it SimpleService.java:

```java
/**
 * The service implementation class
 */
public class SimpleService {

    /**
     * The echo method which will be exposed as the
     * echo operation of the web service
     */
    public String echo(String value) {
        return “from Humphrey service” + value;
    }
}
```

6. Replace the “Humphrey” in the “return” statement with your name
7. Each Axis2 service must have a services.xml file which will inform Axis2 about the service. Following is the services.xml file contents for the SimpleService Web service:
<service>
  <parameter name="ServiceClass" locked="false">
    SimpleService</parameter>
  <operation name="echo">
    <messageReceiver class="org.apache.axis2.rpc.receivers.RPCMessageReceiver"/>
  </operation>
</service>

8. Axis2 expects services to be packaged in a certain format. Let’s do that now.
   a. export JAVA_HOME=/usr/cs/jdk-latest
   b. export PATH=$JAVA_HOME/bin:$PATH
   c. mkdir temp
   d. javac SimpleService.java -d temp/
   e. mkdir temp/META-INF
   f. cp services.xml temp/META-INF/
   g. cd temp
   h. jar -cvf SimpleService.aar *

9. There are two primary hosting environments available for AXIS2 services: [1] Using the SimpleHTTPServer that is available in the Apache Axis2 distribution, or [2] Using Axis2 with Tomcat. We will use the first option. Let’s deploy your new Web service by adding it to the sample Web service directory and telling Axis2 about it:
   a. export AXIS2_HOME=$HOME/PA3/axis2-1.1
   b. cp $HOME/PA3/temp/SimpleService.aar $AXIS2_HOME/repository/services/
   c. edit (emacs, e.g.) $AXIS2_HOME/repository/services/services.list to include your new service

10. We need to make sure that you use a different port than your colleagues:
    a. Edit (emacs, e.g.) $AXIS2_HOME/conf/axis2.xml to use a port other than the default, which is 8080 (suggestion: use “80XY”, where X and Y are two random digits)

11. Start up Axis by: sh $AXIS2_HOME/bin/axis2server.sh

12. Now test it by aiming IE from the Windows machine at the lava machine and port, e.g.,
    http://lava001.cs.virginia.edu:80XY/
    a. If it’s working, you should see a list of deployed services. Click on the hyperlink for “SimpleService” and you’ll see the WSDL.

To hand in for this section:

1. Select the “Internet Explorer” window, click on “SimpleService”, scroll down to the bottom of the WSDL, and hit “ALT-Print Screen” button.
2. Add this as “Part 3.1” of your MS Word doc.

3.2 Writing the AXIS2 Client that consumes the Web Service

13. Start up a new SecureCRT terminal to the chosen lava machine.
14. mkdir PA3/Client
15. cd PA3/Client
16. export JAVA_HOME=/usr/cs/jdk-latest
17. export PATH=$JAVA_HOME/bin:$PATH
18. export AXIS2_HOME=$HOME/PA3/axis2-1.1
20. Now let’s build some code that uses this proxy code. For simplicity, we’ll put it in the same package as the generated code (org.apache.ws.axis2). Create the following file Client.java in src/org/apache/ws/axis2/:

```java
package org.apache.ws.axis2;

import org.apache.ws.axis2.SimpleServiceStub.EchoResponse;

public class Client {
    public static void main(String[] args) throws Exception {
        SimpleServiceStub stub = new SimpleServiceStub();

        //Create the request
        SimpleServiceStub.Echo request = new SimpleServiceStub.Echo();
        request.setParam0("Hello from Humphrey client!");

        //Invoke the service
        EchoResponse response = stub.echo(request);

        System.out.println("Response : " + response.get_return());
    }
}
```

21. Replace the “Humphrey” in the “request.setParam0” statement with your name

22. We can now compile everything:
   a. `cd $HOME/PA3/Client`
   b. `mkdir temp`
   c. `javac -extdirs $AXIS2_HOME/lib/ src/org/apache/ws/axis2/*.java`
      (you can ignore the warning about using “unchecked or unsafe operations”)

23. We can now run the client:
   a. `cd temp`
   b. `java -Djava.ext.dirs=$AXIS2_HOME/lib/ org.apache.ws.axis2.Client`

---

3.3 Consuming the AXIS2 Service via a .NET client


25. In Program.cs for the Client, replace the body of the “Main” routine with the following:

```java
edu.virginia.cs.lava001.echo request = new Client.edu.virginia.cs.lava001.echo();
request.param0 = "Hello from Humphrey .NET client";
edu.virginia.cs.lava001.echoResponse resp = ss.echo(request);
Console.WriteLine(resp.@return);
```
26. Replace the “Humphrey” in the “request[param0]” statement with your name. Also, you might need to replace “lava001”.
27. Re-compile the Client project and re-run the client.

To hand in for this section:
   1. Select the Command Prompt that you ran the client from and hit “ALT-Print Screen” button.
   2. Add this as “Part 3.3” of your MS Word doc.

4. Web Services Development on Linux with gSOAP (C)

4.1 Web Services Deployment on gSOAP

1. Use the second terminal onto the lava machine (leave the AXIS2 service running in the other window).
2. cd $HOME/PA3
3. wget http://www.cs.virginia.edu/~humphrey/cs451/Assignments/Assignment3/gsoap_linux_2.7.9d.tar.gz
4. tar xfz gsoap_linux_2.7.9d.tar.gz
5. ln –s gsoap-linux-2.7 gsoap-install
6. Let’s test the client-side to make sure it worked:
   a. cd $HOME/PA3/gsoap-install/samples/quote
   b. make
   c. ./quote IBM (if it works, this returns the current stock price of IBM)
7. Now let’s test the server-side to make sure it worked:
   a. cd $HOME/PA3/gsoap-install/samples/webserver
   b. make
   c. ./webserver 8190 & (Note: this must be an even-numbered port)

To hand in for this section:
   1. Select the Internet Explorer showing “webserver.wsdl” and hit “ALT-Print Screen” button.
   2. Add this as “Part 4.1” of your MS Word doc.

4.2 Consuming the AXIS2 Service via gSOAP client

28. cd $HOME/PA3
29. mkdir gSOAP-client
30. cd gSOAP-client
31. export PATH=$HOME/PA3/gsoap-install/bin:$PATH
33. soapcpp2 -c SimpleServiceQuery.h
34. create the following C file, and name it SimpleServiceQuery.c

#include "soapH.h" // obtain the generated stub
#include "SimpleServiceSOAP11Binding.nsmap" // obtain the generated XML namespace mapping
// table for the Simple Service

main()
```c
struct soap *soap = soap_new();
struct _ns2__echo ns2__echo;
struct _ns2__echoResponse ns2__echoResponse;

ns2__echo.param0 = "Hello there from Humphrey gSOAP client!";

if (soap_call___ns1__echo(soap, NULL, NULL, &ns2__echo, &ns2__echoResponse) == SOAP_OK)
    printf("Response is %s\n", ns2__echoResponse.return_);
else // an error occurred
    soap_print_fault(soap, stderr); // display the SOAP fault on the stderr stream
```

35. Replace the “Humphrey” in the “ns2__echo.param0” statement with your name.

36. cp $HOME/PA3/gsoap-install/stdsoap2.c.
37. cp $HOME/PA3/gsoap-install/stdsoap2.h.
38. gcc -o SimpleServiceQuery SimpleServiceQuery.c soapC.c soapClient.c stdsoap2.c
39. execute the client app by invoking: ./SimpleServiceQuery

To hand in for this section:
1. Select the Command Prompt that you ran the client from and hit “ALT-Print Screen” button.
2. Add this as “Part 4.2” of your MS Word doc.

What to hand in AND email

A few additional questions that you must answer:
1. What are the pros/cons of each of the three Web service environments (.NET, AXIS, gSOAP)?
2. What are the pros/cons of using XML as a message format for distributed systems?
3. What parts of the assignment did you particularly like?
4. What parts of the assignment are particularly dislike or were confusing (which you would like me to explain better)?
5. How would you improve the assignment?

The due date for this assignment is 12:30pm on Thursday Mar 1. (Note that the submission requirements are basically the same as last time, except that there is no “part 1”.)
1. Hand in a paper copy of your submission.
2. Email an electronic copy of the team’s submission to humphrey@cs.virginia.edu with the subject line “CS451 Assignment 3 (your name)”