Final Project Report

BTech 450 DT

Arun Reddy

Table of Contents

Introduction

Background

This project has been offered by Kiwiplan (www.kiwiplan.com), which specialises in software servicing the corrugating industry. Kiwiplan is a well established company in this sector for over 20 years. Their core competency has been in the sophistication of their scheduling solutions. Kiwiplan software is in use in over 350 sites in 27 countries, and its customer base is rapidly increasing.

ESP is an order-processing, estimation and costing product written in Microsoft's Visual Basic 6. This application is deployed on machines running Microsoft Windows 2000. ESP currently integrates with legacy Kiwiplan products written in FORTRAN via a TCP/IP protocol. They foresee a 'small steps' migration where equivalent Java modules replace FORTRAN code until all FORTRAN code is supplanted. Kiwiplan are also in the process of developing new Java products that integrate with legacy code via the Java Native Interface (JNI). These products have a distributed architecture with a thin-client Java Swing and Web user interfaces.

As more code migrates from FORTRAN to Java, fewer services may become available to ESP through the socket interface. And so, to ensure smooth migration ESP needs access to the Java version of these services.

Benefits

Java being a multi-platform and object oriented programming language will be easier to maintain. Java will also allow addition of improved and enhanced services to the manufacturing products. These services may need to be made available to ESP, so integration is critical.

Integrating new server-side Java products with ESP would enable reporting across all enterprise objects, from the Java or the ESP side. This would create opportunities to develop powerful decision support analysis tools tailored to Kiwiplan software in such a way that they could effectively compete against competitors offering OLAP and ERP solutions that currently dominate this kind of reporting.

The Project

The project is effectively a feasibility study of various approaches of integrating Java and Visual Basic 6. No work has yet been done so far towards integrating the new Java products and ESP, and Kiwiplan hope that the results of this project will be useful in the integration process.

The feasibility study hopes to use simple prototype implementations to establish proof-of-concept. The possible benefits, drawbacks and other factors of each approach will also be discussed.

A number of approaches have been suggested for the integration of Java and VB6 including COM Bridges, SOAP RPC and CORBA. In the first half of this year, I have decided to investigate the new technology of web services as a possible solution.

Considerations

Certain considerations must also be taken into account when evaluating the various approaches. Kiwiplan products are deployed on a variety of platform and configurations. ESP is always deployed on a Microsoft Windows 2000 machine but other products may be deployed on different platforms. The technology being considered for integration must work across platforms.

One of the key goals for Kiwiplan products is good performance over a Wide Area Network. The integration solution should not significantly affect performance of products.

Finally, the development effort required to implement the solution should also be taken into account. The skills and strengths of the company in various technologies must be taken into consideration when making a recommendation.

Acknowledgements

I would like to thank Mr. Van Bellen, Managing Director of Kiwiplan NZ Ltd., for offering this project. I would also like to thank my supervisor Sameer Kalidas for his help, encouragement and support throughout the year. And finally I would like to thank Mr. Gareth Cronin for his support and help through trying times with the AXIS Toolkit and JORAM.

1 Web Services

> A Web Service is a piece of software functionality that is accessible over a network and built on technologies that are independent of platform, programming language and component model.
>
> - **AXIS: Next Generation of Java SOAP (J. Basha & R. Irani)**

Web Services as a new technology has been accepted and hailed by the software industry as an important concept which will influence development in cross platform communications in much the same way as TCP/IP did. More importantly, it has been embraced by all sides of the software divide. Implementations for most of the major platforms have been available for almost three years now, and have been continually improved.

The focus in this project will be on Web Service Implementations for Java and VB6. Sun's Java Web Services Toolkit provides a good implementation. However, the latest from the Apache XML Project, the Apache AXIS SOAP Server, is gaining ground. On the VB6 side, Microsoft's MSSOAP Toolkit 3.0 makes exposing certain methods as Web Services an easy task. Following sections discuss AXIS and MSSOAP 3.0 in detail.

Service Oriented Architecture

Web services are based on a Service Oriented Architecture (SOA) where software functionality is distributed as a set of services.

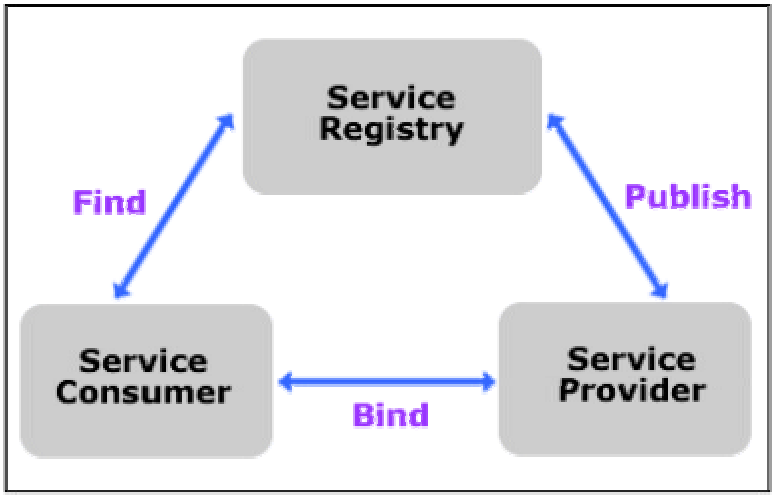
An SOA describes three basic roles:

Service Provider: The service provider implements the web service, describes the interface to this service and publishes this interface to a registry for service consumers to find the web service.

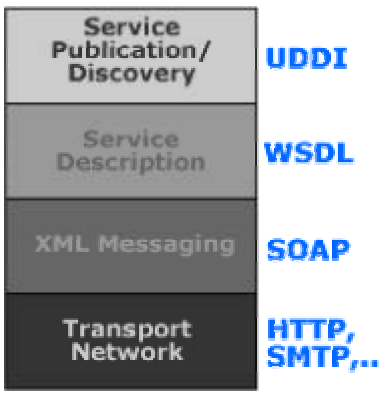
Service Registry: The service registry is a repository of web services to which service providers publish their web service definitions to. Web service consumers use the service registry to find web services and also necessary information in order to bind to and invoke the web services.

Service Consumer: The service consumer makes use of the web service created by the service provider.

A typical Service Oriented Architecture:



To achieve the above mentioned functions of finding a web service and binding to it, if you are a service consumer, or publishing your web service, if you are a service provider, one requires a platform and vendor neutral implementation. Several companies like IBM, Microsoft among others have described a web services stack to achieve this interoperability.



UDDI : A standard mechanism for publishing and discovering web services

WSDL : A standard mechanism for describing web services

SOAP : A standard mechanism for invoking web services

Transport Network: The network responsible for communication between endpoints.

SOAP

SOAP is an XML based communications protocol and encoding format for inter-application development. It was originally conceived by Microsoft and Userland and has steadily gained approval.

Simple Object Access Protocol (SOAP) version 1.0 was released in 1999. Version 1.1 incorporated XML Schema Data Types instead of a native type system. Following this change SOAP has been widely accepted, and is now the standard communications protocol in use with web services. The W3C's XML Protocol working group (http://www.w3.org/2000/xp/Group/) is in the process of turning SOAP into a true open standard, and is in the process of formulating version 1.2 of the SOAP protocol. It is hoped that 1.2 will clear some grey areas in the SOAP 1.1 spec.

Structure of a SOAP Message

```
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
SOAP-
ENV:encodingStyle=”http://schemas.xmlsoap.org/soap/encoding
/”>

   <SOAP-ENV:Header>
   ....
   </SOAP-ENV:Header>

   <SOAP-ENV:Body>
   ....
   </SOAP-ENV:Body>

</SOAP-ENV:Envelope>
```

The SOAP Envelope

The SOAP envelope is a mandatory element which encapsulates a SOAP message. It must contain a SOAP body element although a SOAP header is optional.

The SOAP Header

The SOAP header is an optional element which can contain information needed to successfully process the data in the SOAP body. For example, the SOAP header could convey authentication, security or versioning information, which would otherwise be inappropriate to be included in the SOAP Body.

The SOAP Body
The SOAP Body contains the information being conveyed by the SOAP Message. It could be an RPC Call or just a simple message. SOAP Faults, which are exceptions thrown by the service provider, are also conveyed through the SOAP Body. The SOAP Specification describes how an RPC call is mapped into the XML Structure of a SOAP Message.

SOAP Faults

SOAP Faults are a specially formatted SOAP message as described in the SOAP Specification. SOAP Faults can occur due to a number of factors. For example, the server side can return a SOAP Fault message if the received SOAP Message has not been properly formatted or if it is not understood, or if there was an error on the server-side while the message was being processed, etc.

```
<SOAP-ENV:Body>
  <Fault>
    <faultcode> ... <\faultcode>
    <faultactor> ... <\faultactor>
    <faultstring> ... <\faultstring>
    <detail> ... <\detail>
  </Fault>
</SOAP-ENV:Body>
```

The **FaultCode** identifies the source of the error. A faultcode of *VersionMismatch* is returned when the recipient of the message failed to understand the namespace attribute of the Envelope element. A *MustUnderstand* faultcode is returned when the recipient failed to understand an attribute in the SOAP header marked with the 'mustUnderstand = 1' attribute. A *Client* faultcode id returned when the recipient failed to receive all necessary information for processing the request. And finally a *Server* faultcode indicates a server-side problem.

The **FaultActor** identifies the service that caused the fault.

The **FaultString** describes the problem in detail.

Detail provides application specific information for the cause of the SOAP Fault.

Benefits of Using SOAP

- SOAP is an XML protocol, hence is character based and consequently cross-platform.
- A mechanism for error handling through SOAP fault messages using which error and error-diagnostic information can be exchanged between participants.
- SOAP is easily extensible.
- A flexible mechanism for representing data already serialized in some format (text, XML, etc).
- A convention for representing Remote Procedure Calls and responses as SOAP messages.
- SOAP binds to HTTP, the most common communication protocol on the internet.
- SOAP is an infrastructure technology. One need not know the intricacies of SOAP to be able to program or use Web-services.

AXIS: Apache's SOAP Solution

AXIS is an open source web service toolkit for Java. It is a successor to the Apache Soap Toolkit. It fully supports SOAP version 1.1, and support for version 1.2 is being continually added as the specification is finalised. The current version of AXIS is written in Java and runs as a servlet in a servlet container (most commonly deployed on Tomcat 4.0.1 and above), but a C++ version is being developed. AXIS is supported by major players like IBM, Macromedia and Computer Associates in its development and is reputed to be the best and most compliant implementation of SOAP for Java.

A Simple AXIS Web Service

Writing a web service for the AXIS SOAP Server is a trivial task as it requires no special AXIS specific code. To demonstrate this fact, I have written a simple web service, the code for which can be found in Appendix A. The Web Service is a MathsHelper service which simply returns the addition of two integers supplied. The example has been intentionally kept simple to illustrate the communications between the client and service.

A Simple AXIS Web Client

Writing a client for the web service in Java requires more effort, as the client opens a socket connection between itself and the AXIS. But the complexities of network programming in Java have been abstracted away, and invoking a service is kept simple. The AXIS API includes helper classes Call and Service (in the *org.apache.axis.client* package) which facilitate this.

Supported Data Types

AXIS supports all the data types specified in the SOAP Specification. The following Table from the AXIS User Guide shows the standard mapping from WSDL to Java.

WSDL Mapping	Java data type
`xsd:base64Binary`	`byte[]`
`xsd:boolean`	`boolean`
`xsd:byte`	`byte`
`xsd:dateTime`	`java.util.Calendar`

xsd:decimal	java.math.BigDecimal
xsd:double	double
xsd:float	float
xsd:hexBinary	byte[]
xsd:int	int
xsd:integer	java.math.BigInteger
xsd:long	long
xsd:QName	javax.xml.namespace.QName
xsd:short	short
xsd:string	java.lang.String

SOAP Messaging between the simple Client and Service

The Client sends a properly formatted SOAP Message to the SOAP Server hosting the service. The parameters are sent in the <soapenv:Body> </soapenv:Body> of the message. In this instance, the client requests the addition of two integers (1 and 2).

Client → Service

```
                        <----- HTTP Headers ----->
POST /axis/MathsHelper.jws HTTP/1.0
Content-Type: text/xml; charset=utf-8
Accept: application/soap+xml,application/dime, multipart/related,
text/*
User-Agent: Axis/1.1RC2
Host: localhost
Cache-Control: no-cache
Pragma: no-cache
SOAPAction: ""
Content-Length: 428

                        <----- SOAP Document ----->
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
       xmlns:xsd="http://www.w3.org/2001/XMLSchema"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <soapenv:Body>
  <add
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
   <int_1 xsi:type="xsd:int">1</int_1>
```

```
   <int_2 xsi:type="xsd:int">2</int_2>
  </add>
 </soapenv:Body>
</soapenv:Envelope>
```

The **`SOAPAction`** HTTP header helps the server identify the message as a SOAP Message and specifies the intent of the message. Usually the value of the header is the URI of the web service. An empty string means the intent of the SOAP Message is provided by the HTTP POST request URI (in this case, */axis/MathsHelper.jws*).

Service → Client

The Web Server passes the clients SOAP Message to the SOAP Server for processing. The SOAP Server (AXIS) parses the message and invokes the appropriate web service with the parameters supplied by the client. The web service's response, if any, is packaged into a new SOAP Message and sent back to the Client.

In this example, the result of the addition (1 + 2 = 3) is sent back to the client within the `<addReturn xsi:type="xsd:int">3</addReturn>` tag, where **xsd:int** indicates that the returned value is of type **int**.

```
                        <----- HTTP Headers ----->
HTTP/1.1 200 OK
Set-Cookie: JSESSIONID=0679FB7C0CB503710E27E0BC63E95824;Path=/axis
Content-Type: text/xml; charset=utf-8
Date: Thu, 15 May 2003 23:48:50 GMT
Server: Apache Coyote/1.0
Connection: close

                        <----- SOAP Document ----->
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
       xmlns:xsd="http://www.w3.org/2001/XMLSchema"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <soapenv:Body>
  <addResponse
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
   <addReturn xsi:type="xsd:int">3</addReturn>
  </addResponse>
 </soapenv:Body>
</soapenv:Envelope>
```

Deployment of a Web Service on AXIS

There are two methods of deploying a web service on AXIS.

Drop-in Deployment

This is the simpler of the two methods, and is practical when one has the source code for the web service to be deployed. The source file's extension needs to be changed from `.java` to `.jws` (Java Web Service) and put in the servlet container. There is no requirement to write any additional configuration files. The source file is automatically compiled and run when a client requests the web service.

Web Service Descriptor (WSDD) Deployment

This method of deployment is useful when we do not have the source files for the web service. It also works in instances when the necessary files are packaged as part of a `.jar` file.

A Web Services Descriptor (WSDD) is a n XML based file which includes configuration information for deploying a web service on AXIS. The following is a simple WSDD with minimal configuration information.

```
<deployment xmlns="http://xml.apache.org/axis/wsdd/"
        xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
 <service name="" Web Service Name" provider="  Provider Type ">
  <parameter name="className" value=" Java Class Name "/>
  <parameter name="allowedMethods" value=" method Name "/>
 </service>
</deployment>
```

Web Service Name: is a unique name provided to the web service.

Provider Type: A provider could be thought about as an AXIS module responsible for invoking a web service. AXIS provides two basic providers, an RPC based provider, java:RPC, and a message based provider, java:MSG. There is also an EJB Provider and according to the book *AXIS: Next Generation of Java SOAP*, a COM provider is currently being written.

Class Name: The fully qualified class name of the web service (`packageName.className`).

Allowed Methods: A whitespace-delimited list of methods available to the client. Care should be taken here of not including private and other methods the author wishes not to expose.

Deploying web services on AXIS is made easy using the AdminClient tool which is discussed in the following section on AXIS tools. If successfully deployed, the

web service appears as being deployed on AXIS administration servlet's '*deployed services list*'.

AXIS maintains a list of deployed web services in the **server-config.wsdd** file. So suppose we used the WSDD deployment to deploy our example MathsHelper web service, the following is added to the server-config.wsdd file. …..

```
<service name="MathsHelper" provider="java:RPC">
  <parameter name="allowedMethods" value="add"/>
  <parameter name="className" value="MathsHelper"/>
</service>
```

Undeploying a Web Service on AXIS

Undeploying a deployed web service on AXIS is fairly straightforward and requires another WSDD file, this time a much simpler version with only the name of the web service to be undeployed required. Here is a sample undeploy script.

```
<undeployment xmlns="http://xml.apache.org/axis/wsdd">
    <service name="service name"/>
</undeployment>
```

Running the AdminClient tool to undeploy the web service has the effect of removing the appropriate <service> </service> configuration details from the server-config.wsdd file.

AXIS Tools

In this section I briefly discuss three important tools provided as part of the AXIS package which help in administering, creating, deploying and undeploying web services on AXIS.

Java2WSDL (java org.apache.axis.wsdl.Java2WSDL)

Java2WSDL is a utility which creates a WSDL file given a Java source file. One would require a `.wsdl` file when dealing with the problem of interoperability. MSSOAP requires a wsdl file of the web service it is trying to invoke.

WSDL2Java (java org.apache.axis.wsdl.WSDL2Java)

WSDL2Java has a dual purpose.

- Given a WSDL file, it can create a client stub (a Java class with the same interface as the web service) which can be used to access the service.
- Given a WSDL file, it can generate server-side skeleton code and also additionally create WSDD files for deploying and undeploying the web service.

The first purpose, that of generating client stubs is useful when we wish to create a client for a web service whose WSDL interface we have access to. This situation might arise when the web service has been created by another organisation and they expose the service on the web by providing a URL to the WSDL file of that service.

The second purpose, that of generating server-side skeleton files, given a WSDL of the service seems a bit strange at first. This involves writing a WSDL file for the web service we wish to create first and then allowing the utility to generate server-side skeleton classes for the service. This method of creating a web service might be useful if one is not daunted by the verbosity of a typical WSDL file and can write such a file in reasonable time without errors. This does seem a long-winded way of creating a web service, especially if its functionality is limited.

AdminClient (java.or.apache.axis.client.AdminClient)

As we have seen in a previous section, AdminClient is useful in deploying and undeploying a web service on AXIS using a WSDD file. The simple effect of using AdminClient is that it add configuration details of the web service to the `server-config.wsdd` file typically found in the `/WEB-INF` folder of the AXIS installation.

TCP Monitor (org.apache.axis.utils.tcpmon)

TCP Monitor monitors HTTP traffic between the client and web service displaying the messages being passed. All the HTTP headers and SOAP message is made available. This utility is useful when trying to debug errors and also for demonstrating the communication between the client and the service.
TCP Monitor was used to capture the SOAP transactions between the simple client and the MathsHelper web service.

Benefits of using AXIS

- AXIS is open source. So there are no user agreements to sign, no product activating and no license fee to pay. Also since the source is freely available it can be compiled for platform's for which the binaries aren't available yet. One can, at least in theory, modify the code to suit company needs.
- AXIS is fully compliant with SOAP 1.1 and is being continually updated to meet the v1.2 specifications as they are finalised by the W3C.
- AXIS team comprises of web service experts from companies like HP, IBM and Macromedia. AXIS started off as an IBM product and members of the above mentioned companies are actively involved in coding and testing.
- AXIS is highly extensible. One of the shortcomings of IBM's SOAP4J was that it was inflexible. AXIS was designed specifically with extensibility and flexibility in mind. This means that AXIS can be tuned to work for the specific job at hand

- AXIS can run on a simple servlet engine such as Tomcat or on a full-featured J2EE application server.

MSSOAP Toolkit 3.0

The MSSOAP Toolkit is an 'add-on' package for Visual Basic 6, enabling web services and clients to be written in VB6. SOAP Toolkit 3.0 supports SOAP version 1.1 and WSDL version 1.1. A web service written in VB6 typically runs on the Internet Information Server version 5.x on the Windows 2000 and XP Professional Platforms, and on IIS Version 4.x on Windows NT 4.0.

On the Server side, a server-side component is required which maps invoked XML web service operations to COM object method calls as described by the WSDL and Web Services Meta Language (WSML) files.

WSML files are a Microsoft invention and required only for using the toolkit, whereas WSDL files are a W3C specification and part of the official web services infrastructure.

The Toolkit offers a **high-level** as well as a **low-level** API for creating SOAP Messages. The high-level API makes programming easier by abstracting away the underlying details of creating the message and requires just a few lines of code to generate a SOAP message. But one does not have much control on the formatting of the SOAP Message itself. The low-level API offers several interfaces for low-level interactions. These low-level interfaces allow the client and server to generate, build, exchange, and process SOAP messages.

Data Mapping Visual Basic data Types to XSD Types

The following is the data type mapping from visual basic to XSD types in the WSDL files generated by the WSDLGen3 utility. One will need to take into consideration the data types understood by AXIS when dealing with interoperability issues.

WSDL Mapping	Visual Basic data type
xsd:base64Binary	Byte()
xsd:string	String
xsd:Boolean	Boolean
xsd:dateTime	Date

xsd:decimal	Variant
xsd:short	Integer
xsd:int	Long
xsd:unsignedInt	Long
xsd:float	Single
xsd:double	Double
xsd:anyType	Variant

SOAP Messaging between the Simple Client and Service

Client → Service (Request)

```
POST /MathsHelper/VB/Server/MathsHelper.WSDL HTTP/1.1
SOAPAction: "http://tempuri.org/MathsHelper/action/Class1.add"
Content-Type: text/xml; charset="UTF-8"
User-Agent: SOAP Toolkit 3.0
Host: 127.0.0.1
Content-Length: 536
Connection: Keep-Alive
Cache-Control: no-cache
Pragma: no-cache

<?xml version="1.0" encoding="UTF-8" standalone="no"?>
 <SOAP-ENV:Envelope xmlns:SOAPSDK1="http://www.w3.org/2001/XMLSchema"
xmlns:SOAPSDK2="http://www.w3.org/2001/XMLSchema-instance"
xmlns:SOAPSDK3="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body SOAP-
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
   <SOAPSDK4:add
xmlns:SOAPSDK4="http://tempuri.org/MathsHelper/message/">
    <int_1>2</int_1>
    <int_2>3</int_2>
   </SOAPSDK4:add>
  </SOAP-ENV:Body>
 </SOAP-ENV:Envelope>
```

The SOAP Toolkit makes use of the SOAPAction HTTP header, however it is upto the server side to make use of the header or ignore it. The AXIS SOAP Server currently ignores the SOAPAction header.

Service → Client (Response)

```
HTTP/1.1 100 Continue
Server: Microsoft-IIS/5.1
Date: Tue, 27 May 2003 19:16:10 GMT
HTTP/1.1 200 OK
Server: Microsoft-IIS/5.1
Date: Tue, 27 May 2003 19:16:10 GMT
Content-Type: text/xml; charset="UTF-8"
Content-Length: 538
Expires: -1;

<?xml version="1.0" encoding="UTF-8" standalone="no"?>
 <SOAP-ENV:Envelope xmlns:SOAPSDK1="http://www.w3.org/2001/XMLSchema"
xmlns:SOAPSDK2="http://www.w3.org/2001/XMLSchema-instance"
xmlns:SOAPSDK3="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body SOAP-
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <SOAPSDK4:addResponse
xmlns:SOAPSDK4="http://tempuri.org/MathsHelper/message/">
      <Result>5</Result>
    </SOAPSDK4:addResponse>
  </SOAP-ENV:Body>
 </SOAP-ENV:Envelope>
```

Using the Low Level API

The Low Level API gives more control to the programmer in terms of generating, building, exchanging and processing SOAP messages.

The Low Level API should be used when the WSDL file of the service is not available but information such as the port number and message formatting is known. Programming with the Low Level API is similar to the 'Simple Client for Java' example for the AXIS SOAP Server, where configuration information such as port number, method name and parameter types was hard coded.

The Low Level API also comes in handy when we want to manually create the WSDL file rather than use the WSDLGen3 utility (described later).

MSSOAP 3.0 Toolkit Tools

There are three tools packages with the toolkit to help in web service development. They are quite similar in nature to what comes packaged with AXIS.

WSDL/WSML Generator (WSDLgen3.exe)

This utility generates WSDL and WSML files for a given web service. The WSDL file generated by this utility can be copied over to the client side if the client has

been programmed using the high level API. The WSML file stays with the service and is a toolkit specific file.

SOAP Tracer (MsSoapT3.exe)
The SOAP Tracer is similar to the TCP Monitor utility of AXIS. It monitors the SOAP messages being passed between the client and service and displays the requests and responses. I prefer to use the TCP Monitor utility rather than SOAP Tracer, firstly because it's easier to use and the interface is more user friendly and secondly, I have found TCP Monitor to be more stable than SOAP Tracer. On one occasion, an incorrectly formatted SOAP Message sent SOAP Tracer into an infinite loop.

SOAPVDIR.CMD
This is a batch file running a visual basic script which configures a virtual directory on IIS to use SOAP Toolkit 3 ISAPI. SOAP requests are sent by the client as POST requests. Not running this batch file when configuring a virtual directory on IIS causes IIS to refuse HTTP POST requests and IIS replies with a `405 Method Not Allowed`. It is easy to forget to run this file. On two occasions, I lost almost an hour each time trying different virtual directory setups as IIS was refusing POST requests. So it is most important to run this batch file.

Interoperability

After introducing the various pieces of software and tools needed to solve the problem of interoperability, we finally get down to making it happen. I decided to first focus on getting a VB client to interoperate with a Java Service.

Client: VB ← → **Service:** Java

I am using the same example (the MathsHelper service). So the code for the Client remains unchanged. If we decide to use the high level API to code the client we have to supply the WSDL file for the service, so that the client can actually locate the service and format the query it sends to the service.

To generate the WSDL file for the service, I used the Java2WSDL utility from the AXIS package. The following command generates a WSDL file where

```
java org.apache.axis.wsdl.Java2WSDL -o MathsHelper.wsdl  -l
"http://localhost:8080/axis/services/MathsHelper" -n
"urn:MathsHelper"  MathsHelper
```

Where, **-o** is the name of the WSDL file to be created, **-l** is the URL of the web service and **–n** is the target namespace.

Java2WSDL creates a properly formatted WSDL file, but in the **<service>** section of the file, which defines the name of the web service, it appends "Service" to the name of the java class.

For example, on running the above command, we get the following in our WSDL file

```
<wsdl:service name="MathsHelperService">
 <wsdl:port binding="impl:MathsHelperSoapBinding"
name="MathsHelper">
  <wsdlsoap:address
location="http://localhost:8080/axis/services/MathsHelper"/
>
 </wsdl:port>
</wsdl:service>
```

Using this WSDL file with the VB client will obviously not work as the VB client is looking for the service names MathsHelper in WSDL file. This is because the following line in the VB6 source file defines the name of the service…..

```
Call SoapClient3.mssoapinit("MathsHelper.wsdl",
"MathsHelper", "MathsHelper")
```

where the first parameter of the mssoapinit method refers to the **wsdl** file, the second to the name of the **service** and the third to the **port**.

So the two alternatives are to change the second parameter from MathsHelper to MathsHelperService
OR
To edit the name of the service in the WSDL file from MathsHelperService to MathsHelper. Either alternative will solve this problem.

There are some further observations which can be made form looking at the SOAP Request ……

```
POST /axis/services/MathsHelper HTTP/1.1
SOAPAction: ""
Content-Type: text/xml; charset="UTF-8"
User-Agent: SOAP Toolkit 3.0
Host: 127.0.0.1
Content-Length: 500
Connection: Keep-Alive
Cache-Control: no-cache
Pragma: no-cache

<?xml version="1.0" encoding="UTF-8" standalone="no"?>
   <SOAP-ENV:Envelope xmlns:SOAPSDK1="http://www.w3.org/2001/XMLSchema"
xmlns:SOAPSDK2="http://www.w3.org/2001/XMLSchema-instance"
xmlns:SOAPSDK3="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
      <SOAP-ENV:Body SOAP-
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
         <SOAPSDK4:add xmlns:SOAPSDK4="MathsHelper">
            <in0>2</in0>
            <in1>3</in1>
         </SOAPSDK4:add>
      </SOAP-ENV:Body>
   </SOAP-ENV:Envelope>
```

Notice that the parameters haven't been XSD typed. Ideally we would have liked the following so that the receiving server can properly interpret the message.

```
                    <in0 xsi:type="xsd:int">2</in0>
                    <in1 xsi:type="xsd:int">3</in1>
```

MSSOAP Toolkit doesn't type the parameters being passed, however it does check that the proper parameters are being sent. For example, if we sent a String in place of an integer for either of the parameters, the toolkit will throw an exception and will not forward the message.

This may not always work especially in instances where VB considers something to be of one type, but Java another. So care should be exercised here and proper testing needs to be carried out before the client and web service are trusted to carry out critical calculations.

Client: Java ← → **Service:** VB

The Interoperation between a Java client and a VB6 service proved slightly tricky. The AXIS SOAP Server disregards the SOAPAction HTTP header and by default sets the value to a null string in its requests.

```
SOAPAction: ""
```

The VB6 SOAP Toolkit however strictly requires the value of the SOAPAction header to be set to the name of the service parameter in the WSDL Document and refuses to accept a null string. Also the namespace for the message body needs to be set for the toolkit to interpret the message correctly. The following lines of Java code set the soapaction and body namespace.

```
   //Requests the use of a SOAPAction Header in the request
call.setUseSOAPAction(true);
   //Sets the header value
call.setSOAPActionURI("SOAPAction Header value");
   //sets the SOAP Body Namespace and method to invoke
call.setOperationName(new QName("Namespace","MethodName"));
```

It is strange that the methods **`call.setUseSOAPAction(boolean)`** and **`call.setSOAPActionURI(String)`** are not in the API documentation released as part of the AXIS package. I luckily found references to them on a web posting by one of the AXIS Developers who works for Macromedia.

An issue previously discussed is the fact that MSSOAP Toolkit 3.0 fails to specify the type of parameters being passed. Here is part of the response from the VB service to the Java client which shows the untyped result which AXIS does not have a problem accepting as we have specified in the client code that the return parameter is of primitive int type.

```
<SOAPSDK4:addResponse
xmlns:SOAPSDK4="http://amalthea.org/MathsHelper/message/">
            <Result>3</Result>
        </SOAPSDK4:addResponse>
```

2 Mid Semester Conclusions

Considerations
Achieving interoperability between VB6 and Java certainly works using the relatively new technology of web services as was demonstrated in the previous sections. However, how does it weigh up in the face of the considerations for a solution discussed in the introduction? To recap, some of the considerations were,

- The solution should work across hardware and software platforms.
- Good performance across a Wide Area Network.
- Minimal development effort.

Web Services are based on XML based protocols which are character based protocols and work across software and hardware platforms. The messages being passed around are text messages which are structured in a human-readable form and debugging is vastly simplified due to this fact. There are SOAP implementations for most platforms, be they Windows or some flavour of UNIX.

Web Services as a concept was developed with wide area networks in mind. Services can be invoked over a number of different protocols including HTTP and SMTP, though HTTP is the predominate protocol for obvious reasons. During my experiments with AXIS and the SOAP Toolkit, I didn't find any significant delays in client-server communications. Communication between VB6 client and service was almost instantaneous. AXIS did take a few milliseconds at times to create a SOAP message or interpret a received message, though this may be due to the fact the AXIS currently runs as a servlet in a servlet container. This may change when a C++ implementation of AXIS arrives which may run as a CGI process or have its own built-in server. But having a Java implementation certainly makes it accessible to more platforms.

Incorporating any integration technology would involve some development effort. The goal is to keep this to a minimum. As can be seen from the examples, there is almost no additional effort needed to expose methods as services on the server side other than creating a WSDL and a WSML files. A VB client requires

just a couple of lines of code when programming in the high-level API. The Java client is also easy to program once one understands the AXIS API.
As mentioned earlier, debugging during development is made easier due to the nature of SOAP messages. SOAP Faults thrown by both the VB Toolkit and AXIS are helpful in recognising and fixing the problem.

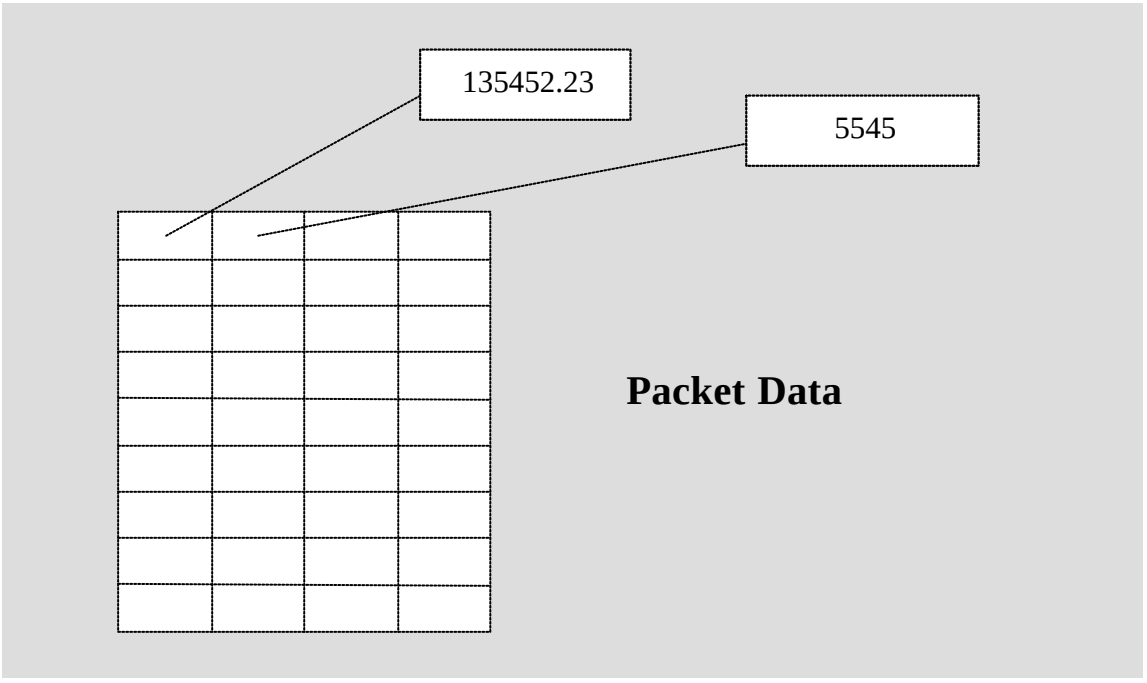
Other Effects

The system at present is a one-way communication from ESP to the FORTRAN side. Web Services would allow for communications to run both ways leading to a better integrated product. Also one has the option of pursuing traditional RPC style calls or Messaging calls, whichever suits the situation.

From investigations done upto this point, it is clear that Java and VB6 communicate seamlessly as long as the parameters being passed are primitive data or strings.

Further investigations will have to be undertaken to evaluate the effectiveness of this system if user defined types are used.

ESP currently sends packet data over to the FORTRAN side which is a bundled set of parameters consisting of primitive data.



Any change to the structure of the packet leads to changing the pieces of code which construct and interpret the packet. With web services one will be able to

retain this type of communications or move to a more manageable system wherein only requested data is passed along the wire.

Kiwiplan as a company are moving towards a cross-platform and open source approach of software development, reducing their dependence on a single vendor. The web service solution, I have suggested above involves AXIS which is an open source implementation of the JAX-RPC API and runs in any servlet container including the Catalina container, a part of the Tomcat server, which is also an Apache project. The VB Toolkit is a free add-on for SOAP development on VB6. The good news is that Microsoft seems to be showing continued interest in developing the toolkit even after releasing the .NET development platform which seems to be the preferred platform for web service development (MSSOAP Toolkit version 3 was released in March 2003).

Web Services has not been around for very long but it is already in wide use in the industry. *Microsoft's Windows Update* site, for example uses a web service which the Internet Explorer browser connects to determine which updates to download.

The German IT magazine tecChannel, an IDG publication, recently revealed this fact, that the site uses a web service and information is passed through SOAP messages. Also revealed was the fact that MS seems to be gathering personal information which the site denies, but allows itself the right to gather personal information through a paragraph in the Win XP SP1 License agreement. ☺

The only possible drawback could be the fact that AXIS requires JDK 1.4.0. This would be a problem in which the JDK version is for some reason not available for that platform.

I conclude with a few comments from some columnists at the IT publication Computerweek from both sides of the argument on what they think about the new technology…..

> Web services will make it easier to integrate internal systems. If nothing else happens, that will be a success story. Web services are likely to replace what we now call electronic data interchange. They're a powerful tool for automating supply chain activity among trusted partners.
>
> **Mitch Betts, features editor**
>
> The Web services Tsunami: already, Web services are being used internally and externally. IT may never be the same.
> Computerworld, May 19, 2003 v37 i20 p25(1).

> With the current economic slowdown, it is easy to see why the industry is so desperate for web services to succeed…..Call me a cynic, but the idea that web services will change the way we conduct business seems an awfully familiar message.
>
> **James Rogers**,
> Web services; Cynics hear familiar ring to the next big thing,
> Computer Weekly, May 13, 2003 p21.

> Because of the global scope of Web-based services, the potential financial upside is extremely attractive…..Web services also promise improved collaboration with customers, partners and suppliers….. However, the Web services vision is still new and not without risk. Despite the significant potential, it remains to be seen how Web services will play out on a large scale.
>
> **Frank P. Coyle**,
> (Author of XML, Web Services, and the Data Revolution, Addison-Wesley, 2002)

Web services, simply put: here's an executive guide that explains XML, SOAP and web services -- in plain English

3 Pilot Implementation

Introduction
The second part of this project dealt with a pilot implementation of the interoperability ideas expressed during the first semester. Most of this work was carried out at Kiwiplan using their present systems as a reference. The primary objective was the transfer of business information from ESP (VB6) to Java applications which would require the use of this information.

To demonstrate this, Customer information such as Address and Contact information among others was chosen for the pilot implementation. A customer could have a number of addresses with each address having a unique '*AddressNo*'. Customers are identified with a '*CustomerNo*'. Thus the combination of a CustomerNo and AddressNo uniquely identified a set of address, contact and related information.

Before discussing the details, I would first like to explain how the AXIS Toolkit and MSSOAP Toolkit deal with structured XML information. Although we are only dealing with character based information here, this information needs to pass through the network in a structured self-explanatory manner. For example, we would like to see the following type of structure …..

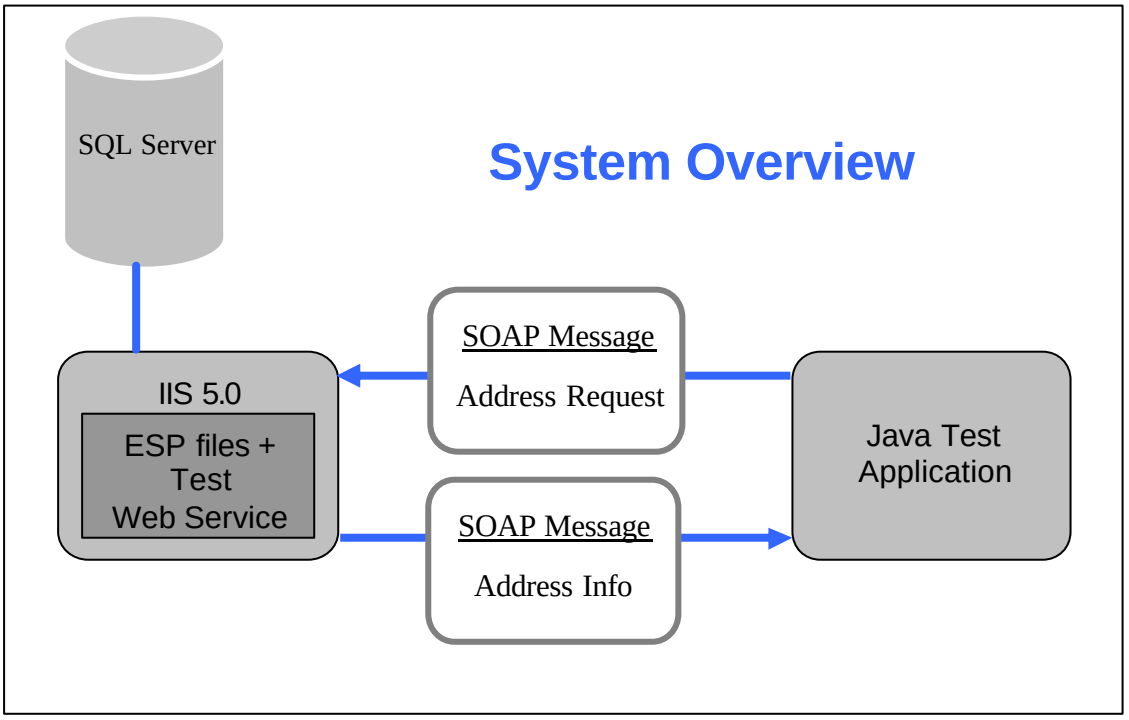
```
<Address>
      <Street>…</Street>
      <City>…</City>
      ……
      <Telephone>…</Telephone>
      <Email>…</Email>
</Address>
```

The MSSOAP Toolkit works heavily off the WSDL file describing the web service. There are a number of ways to have the Toolkit read and write structured XML, but one of the easiest is to define a new data type (In the case of the above example, an Address type). The WSDLGen utility will then create appropriate WSDL files which the Toolkit uses for serializing and deserializing the data type. The *User-Defined Type Mapper*, a part of the toolkit will perform the serializing and deserializing functions in the background. One could also achieve this effect programmatically by creating an IXMLDOMList, but this approach involves a lot

more work which seems unnecessary when the WSDLGen utility accomplishes the same task without additional effort. Creating an IXMLDOMList is much more appropriate when one is dealing with COM Data types.

The AXIS Toolkit has a BeanSerializer class which serializes and deserializes any class which follows the JavaBean pattern of accessor and mutator methods. This is similar in functionality to the User-Defined Type Mapper of the MSSOAP Toolkit. The other alternative would be to write a custom serializer/deserializer. This approach gives total control of the serializing and deserializing processes. AXIS implements the JAX-RPC specification and so the custom serializers and deserializers follow the pattern defined in the specification.

System Overview



The test system was a PC running Win2K with IIS 5.0. The business information (address information, in this case) is held in a SQL Server database running on a remote machine. The relevant ESP files dealing with database access and the web service serving the address information ran on IIS 5.0. The Java Test Application utilized the AXIS API.

Implementation
I thought it would be best to describe the evolution of the implementation through a series of versions. Each successive version describes an improvement over previous implementations.

Version One:

- Java Test Application requests Address information by sending across CompanyNo and AddressNo.

```
HTTP Headers and SOAP Envelope/Body declarations
<AddressRequest>
     <CompanyNo>12454</CompanyNo>
     <AddressNo>1</AddressNo>
</AddressRequest>
……..
```

- The web service looks up the database for the requested information and creates an AddressInfo user defined variable type. This type is automatically serialized and returned to the requesting Java Application in the form of a SOAP message.

```
HTTP Headers and SOAP Envelope/Body declarations
<AddressInfo>
     <Street>10 La Rascasse</Street>
     <City> Monte Carlo</City>
     ……..
     <Telephone>123-456789</Telephone>
     <Email>jt@ferrari.it</Email>
</AddressInfo>
……..
```

- The AddressInfo deserializer on the Java side takes care of deserializing the Address information and creating an appropriate object.

Version Two:

New feature: Ability to send and receive multiple requests.

- To enable the application to request multiple addresses in just a single call, an AddressRequest class was created and corresponding serializer/deserializer pair written.
- The Java Application now creates an array of AddressRequest objects and calls the remote web service with this array as a parameter. The AXIS Toolkit has inbuilt functionality to handle arrays, so no extra effort is needed here.

```
HTTP Headers + Envelope/Body declaration
<ns1:getAddresses namespace declarations>
      <AddressBatch xsi:type="soapenc:Array"
soapenc:arrayType="ns2:AddressRequest [3]" namespace
declarations>
      <item href="#id0"/>
      <item href="#id1"/>
      <item href="#id2"/>
      </AddressBatch>
</ns1:getAddresses>
<multiRef id="id2" soapenc:root="0"
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xsi:type="ns3:AddressRequest" namespace declarations >
      <CompanyNo xsi:type="xsd:string" xsi:nil="true"/>
      <AddressNo xsi:type="xsd:int">0</AddressNo>
</multiRef>
<multiRef id="id0" soapenc:root="0"
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xsi:type="ns4:AddressRequest" namespace declarations >
      <CompanyNo xsi:type="xsd:string">108436</CompanyNo>
      <AddressNo xsi:type="xsd:int">1</AddressNo>
</multiRef>
<multiRef id="id1" …… >
      ……
</multiRef>
……
```

- Similarly on the VB side, the MSSOAP Toolkit handles arrays without any additional effort. The web service then makes a series of database calls for the necessary information which it returns in the form of an array of AddressInfo structures.
- Back on the Java side, this is deserialized by AXIS and an array of AddressInfo objects in created.

Version Three

New Feature: Modification of the AddressInfo structure to include a better defined ‘OpenDays’ tag.

- The ‘OpenDays’ tag in the AddressInfo structure defined the days on which the address was open for accepting deliveries. ESP stores this information a seven character string. For example, “NYYYYYN” means that the business is open from Monday – Friday and closed on the Weekends, with ‘N’ signifying a day when the business was closed and ‘Y’ representing a day open for business. ESP convention dictated that the week always began on a Sunday and ended on a Saturday.
- It was suggested that having a string like “NYYYYYN” could be better expressed as an XML Structure.

```
HTTP Headers and SOAP Envelope/Body declarations + Other Data
<OpenDays>
      <Sunday>false</Sunday>
      <Monday>true</Monday>
      <Tuesday>true</Tuesday>
      <Wednesday>true</Wednesday>
      <Thursday>true</Thursday>
      <Friday>true</Friday>
      <Saturday>false</Saturday>
</OpenDays>
……….
```

- This modification involved defining a new VB6 type called OpenDays on the VB side and a new OpenDays class with corresponding serializer/deserializer on the Java side. This produces the following standards compliant XML structure which AXIS has no problem in deserializing.

Version Four

New Feature: Error Handling

- Error Handling through SOAP Fault messages is considered as one of the important features of SOAP. SOAP Faults are a specially formatted SOAP message as described in the SOAP Specification. I'll skip over the finer details as they have already been described in a previous section.

Error Handling in the MSSOAP Toolkit

The Toolkit will always respond with a SOAP Fault message when an error occurs in the processing chain. The message will largely be of a default nature with certain error specific information. The structure and contents of this default Fault message is described in the *SOAP Toolkit 3.0 User Guide*.

However to respond to error with a custom SOAP Fault message, the object must implement the **ISoapError** interface. The ISoapError interface basically has five properties. The following is a reduced reproduction of the interface from the SOAP Reference guide.

Property Name	Description
Detail	(read-only) Provides the value of the <detail> element.
FaultActor	(read-only) Provides the value of the <faultactor> element.
FaultCode	(read-only) Provides the value of the <faultcode> element.
FaultCodeNamespace	(read-only) Provides the namespace URI used to qualify the value of the <faultcode> element of the SOAP <Fault> element.

FaultString (read-only) Provides the value of the <faultstring> element.

The above properties may be customized for the error. The Toolkit will then use these values when creating the Fault Message.

Error Handling in AXIS

The AXIS system throws an *org.apache.axis.AxisFault* whenever an exception occurs. This class contains all the information pertaining to a SOAP Fault and also methods for setting the various fault detail values, which can be customised for the occasion.

- The Implementation returned an appropriate Fault Message whenever an exception occurred. For example, when the request was for a non-existent *CompanyNo* or *AddressNo*.

Version Five

New Feature: Modification to the way the 'Open Days' information was handled.

- As defined in version three, the string specifying the days on which a particular address was open for accepting deliveries was not intuitive and hence a self-explanatory XML Structure was implemented.
- This conversion from a String → OpenDays type was being carried out by the web service and then passed back to the Java end. It was rightly pointed out, that the web service should have the right to serve data in its native format and the consumers of the service should deal with format conversion at their end.
- So this new version refrains from performing any function on the OpenDays string. The conversion is now carried out in the AddressInfo deserializer where the string is intercepted and an OpenDays object created from the information.

Further work

This section detail some further work done relating to data translation on the information before it reaches the client. These methods would be applicable in situations such as converting the OpenDays string to a self-explanatory OpenDays structure.

Custom Handlers

The concept of SOAP Message Handlers is part of the JAX-RPC specifications. A handler basically performs some pre-processing or post-processing on the message travelling to or from a client or service. Examples listed in the specifications for possible handler functions are encryption, logging and caching.

I investigated the use of handlers for data format conversion such as converting the OpenDays string to a self-explanatory OpenDays structure. In theory, this should be possible as a function like encryption also involves reading data from a stream and writing back. However, I was unsuccessful using the AXIS API to perform the data conversion.

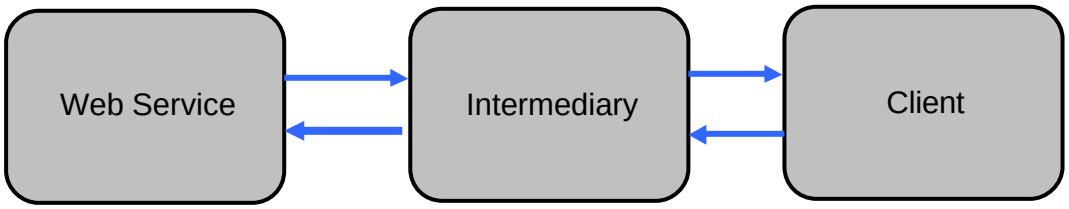
SOAP Intermediaries

> The semantics of one or more SOAP blocks in a SOAP message, or the SOAP message exchange pattern used MAY request that the SOAP message be forwarded to another SOAP node on behalf of the initiator of the inbound SOAP message. In this case, the processing SOAP node acts in the role of a SOAP intermediary.
>
> http://www.w3.org/2000/xp/Group/2/02/27-SOAPIntermediaries.html

The concept of SOAP Intermediaries provides an excellent opportunity to undertake processing on the message before it reaches its final destination. In our case, we could have the *AddressInfo* structures pass through a SOAP Intermediary which could have those structures converted to a format acceptable to the client.

Unfortunately AXIS does not support SOAP Intermediaries at present. It is expected that a future release will provide support as it is part of the SOAP Specification. It is not clear whether the MSSOAP Toolkit supports this either. Further experimentation will be required in this case.

However, one can always hard code the path followed by a message and achieve the same effect. But of course this is not an ideal solution and will be hard to manage.



In the above proposed scenario, the intermediary would let the request pass through to the service, but would perform some processing on the information returned by the service before passing it on to the client.

Notes on the Implementation:

1. The implementation was fairly basic in its form, the main idea being to prove the feasibility of using web service as medium for information transfer between Java and VB6. And so it did not contain any advanced database access functions. When requesting multiple addresses, the web service merely made a series of database calls, each time requesting a single address. Fetching multiple addresses with a single call would be a much more efficient method than making multiple calls.

2. When requesting multiple addresses from the service, the array of address requests needs to be one element longer (for instance, if I want to request 50 addresses, I should send the service an array of address requests of length 51. The final element should be of type *AddressRequest*, but the contents of which can be null). The web service will return an array 51 addresses with one of the addresses (usually the last address) set to null. The VB web service, it seems, receives only an array of 50 requests, which it processes and sends back 50 addresses. The MSSOAP Toolkit for some reason seems to be adding the null address to the last element. If the Java client sends an array of 50 address requests expecting a similar number of addresses back, it in fact receives only 49, with one of the addresses (usually the last address) set to null. The workaround is to send one extra request. I couldn't find a mention of this feature in either the manual, published literature or on the internet.

3. The following affects WSDL files created using the WSDL Generator utility of the SOAP toolkit for services involving user-defined types. The WSDL file by default instructs the web service to return the request as part of the response.

For example, consider the following…..

SOAP Request

HTTP Headers

```
<?xml version="1.0" encoding="UTF-8"?>
  <soapenv:Envelope namespace info>
    <soapenv:Body>
      <ns1:getAddress namespace info>
        <AddressRequest href="#id0"/>
      </ns1:getAddress>
      <multiRef id="id0" namespace info>
        <CompanyNo xsi:type="xsd:string">108436</CompanyNo>
        <AddressNo xsi:type="xsd:int">2</AddressNo>
      </multiRef>
    </soapenv:Body>
  </soapenv:Envelope>
```

An AddressRequest user defined type is sent across in the request

SOAP Response

HTTP Headers

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
  <SOAP-ENV: namespace info>
    <SOAP-ENV:Body namespace info >
      <SOAPSDK4:getAddressResponse namespace info>
        <Result href="#id1"/>
        <AddressRequest href="#id3"/>
      </SOAPSDK4:getAddressResponse>

            ##Address  information##

      <SOAPSDK7:AddressRequest namespace info>
        <CompanyNo>108436</CompanyNo>
        <AddressNo>2</AddressNo>
      </SOAPSDK7:AddressRequest>
    </SOAP-ENV:Body>
  </SOAP-ENV:Envelope>
```

The AddressRequest user defined type is sent back with the response by default. Since we do not require this information and as it can be s ubstantial when a request size is large or when we make multiple requests, we can make editions to the .WSDL file to instruct the web service not to send the request back.

To avoid having the response sent back, editing the parts of the WSDL file which instruct the web service to return the response is the solution…
The following are relevant parts from the WSDL file for the web service. The highlighted portions need to be deleted from the file.

```
<?xml version="1.0" encoding="UTF-8" ?>
<definitions name="AddressAgent" namespace info >
……
<message name="clsAddressAgent.getAddressResponse">
<part name="Result" type="typens:AddressInfo" />
<part name="AddressRequest" type="typens:AddressRequest" />
</message>
……
<message name="clsAddressAgent.getAddressesResponse">
<part name="Result" type="typens:ArrayOfAddress" />
<part name="AddressBatch"
type="typens:ArrayOfAddressRequest" />
</message>
……
```

```
<message name="clsAddressAgent.getFakeAddressesResponse">
<part name="Result" type="typens:ArrayOfAddress" />
<part name="AddressBatch"
type="typens:ArrayOfAddressRequest" />
</message>
……
<operation name="getAddress">
……
<output>
<soap:body use="encoded"
namespace="http://tempuri.org/AddressAgent/message/"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
parts="Result AddressRequest" />
</output>
</operation>
<operation name="getAddresses">
……
<output>
<soap:body use="encoded"
namespace="http://tempuri.org/AddressAgent/message/"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
parts="Result AddressBatch" />
</output>
</operation>
<operation name="getFakeAddresses">
……
<output>
<soap:body use="encoded"
namespace="http://tempuri.org/AddressAgent/message/"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
parts="Result AddressBatch" />
……
……
</definitions>
```

4 Messaging

This section details the results of an investigation into possible architectures for a messaging system. The MSSOAP Toolkit version 3 only supports RPC style calls. The manual suggests editing the WSDL file of the service, to remove the lines which list the return type for the call. This forces the toolkit to close the connection on the VB side, which seems like a rather crude way. In addition, there is no reliability mechanism for these types of calls. AXIS also supports something similar (Described as a one-way input/outbound message pattern in Jeelani[1]). It is presently unclear whether it will ever support reliable asynchronous messaging. While first investigating this matter the AXIS developers seemed to be concentrating on ironing out the bugs and adding new features to RPC style communication. Since then, there has not been any movement on this front.
So I focused my investigations into enabling reliable asynchronous communications in this type of a scenario using either of two Java APIs, which support asynchronous messaging.

JAXM (Java API for XML Messaging)

> The Java API for XML Messaging (JAXM) Optional Package enables applications to send and receive document oriented XML messages using a pure Java API. JAXM implements Simple Object Access Protocol (SOAP) 1.1 with Attachments messaging so that developers can focus on building, sending, receiving, and decomposing messages for their applications instead of programming low level XML communications routines
>
> http://java.sun.com/xml/jaxm/

JAXM is one among the many Java APIs for XML Processing. Unfortunately, JAXM seems to be loosing favour with the developer community and even within the team which created the specifications. The reasons include existence of a similar standard (JMS) for messaging which is reportedly easier to use and provides better features.

JAXM which was part of the JWSDP version 1.1 (Java Web Services Development Pack) has been dropped in version 1.2 and is offered as an additional download.

The Reference Implementation for the latest JAXM specification (1.1.2) requires all participating clients to run on the same servlet or J2EE container (in our case, Tomcat). This does not help the situation as we require a message queue, which accepts SOAP Messages from an external source (ESP) and then forwards it onto a Java Client.

JMS (Java Messaging Service)

JMS is a strategic technology for J2EE. JMS will work in concert with other technologies to provide reliable, asynchronous communication between components in a distributed computing environment.

http://java.sun.com/products/jms/

JMS was developed by Sun working in close cooperation with leading enterprise-messaging vendors including IBM, Hewlett-Packard and Sonic Software.
JMS is a part of the J2EE platform since the release of version 1.3. The J2EE platform now has a reference implementation of the JMS standard. In addition there are a number of commercial as well as open-source implementations.

One such open-source implementation is JORAM (Java Open Reliable Asynchronous Messaging) by the ObjectWeb consortium.

JORAM incorporates a 100% pure Java implementation of JMS (Java Message Service API released by Sun Microsystems, Inc.). It provides access to a MOM (Message Oriented Middleware), built on top of the ScalAgent agents (http://www.scalagent.com) based distributed platform. JORAM is a free, open source initiative.

http://joram.objectweb.org/index.html

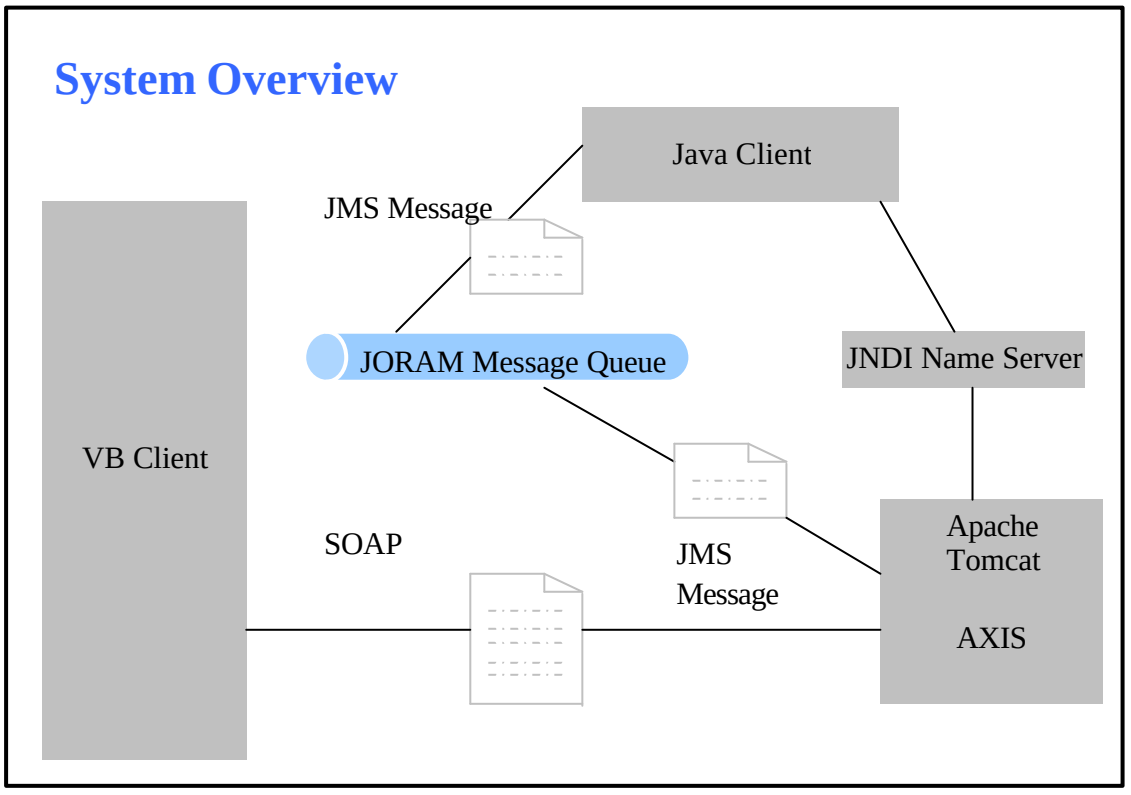
JMS clients and the Message Queue communicate using a JMS specific protocol, which is unknown to VB. So we need a mechanism to get the message from VB and push it onto the JMS queue for a Java client to retrieve the message at a later time. This could be done by a web service running on AXIS.

There exists a reliable connection between the JORAM Message Queue and AXIS and between the Message Queue and the Java Client. The JMS specification calls for this reliability and if the JORAM implementation adheres strictly to this specification, which it claims, we should have no problems there.

So the system at present operates in the following manner:

- VB sends a message wrapped in a SOAP Package to AXIS
- AXIS unwraps the message and pushes it onto the message queue (The service utilises the JORAM API for this).
- AXIS informs VB of the success or failure of this operation.
- A Java client at a later time can connect to the queue to receive the message.

In this manner messaging can be reliably carried out between VB and Java. I have managed to successfully send and receive Text Messages. In theory, one should also be able to send and receive objects (or in this case, AddressInfo with updated address information). However, I have been unsuccessful in my attempts at passing AddressInfo objects. After much investigation, I am unable to track down the problem. At first, it appeared to be a case of WSDL files generated by AXIS being unacceptable to the MSSOAP Toolkit.

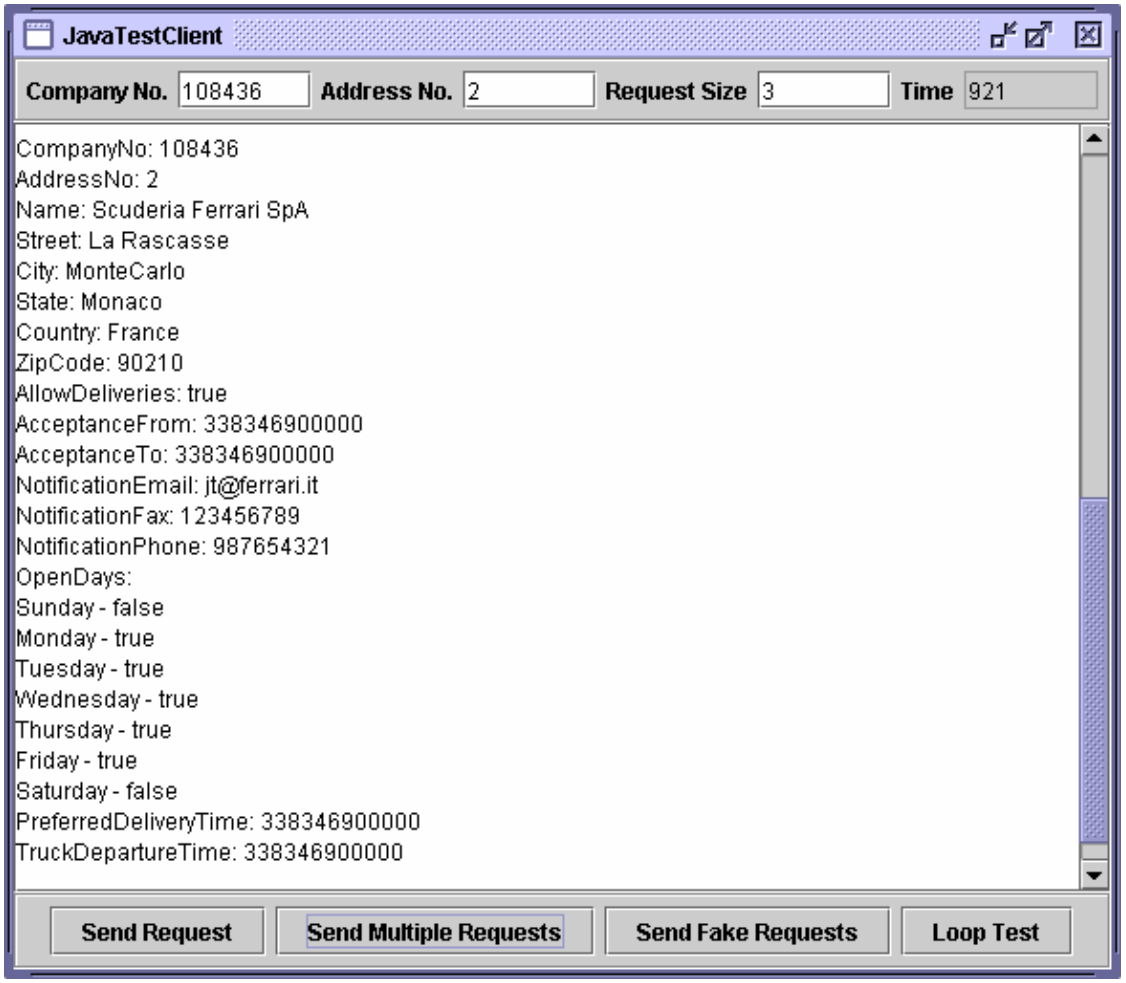


5 Screenshots

A few screenshots of some of the pilot implementations for demonstrating interoperability and messaging

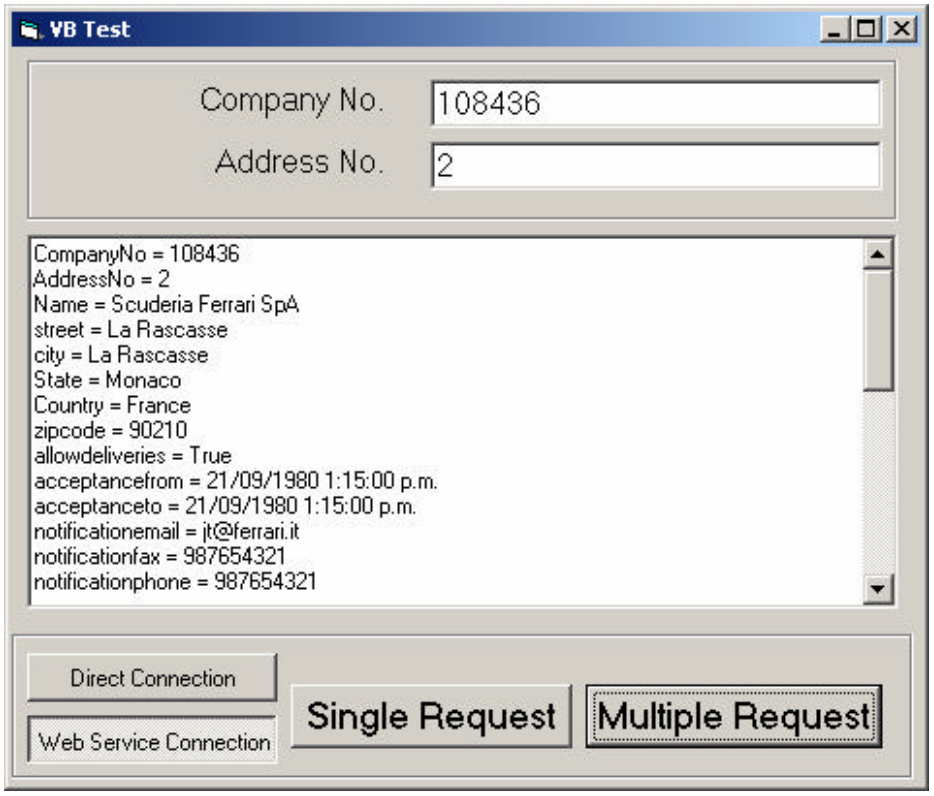
Interoperability

Java Test Client



- Able to send single and multiple requests.
- Multiple requests in the implementation were for the same company-address information. So the round trip time may not be an accurate estimate if the database is caching recently served information.
- Able to send a number of fake requests (Fake Requests are replied to with a response hard-coded response in the web service with no database querying involved). This was to gauge the impact of having to access the database.

VB Test Client



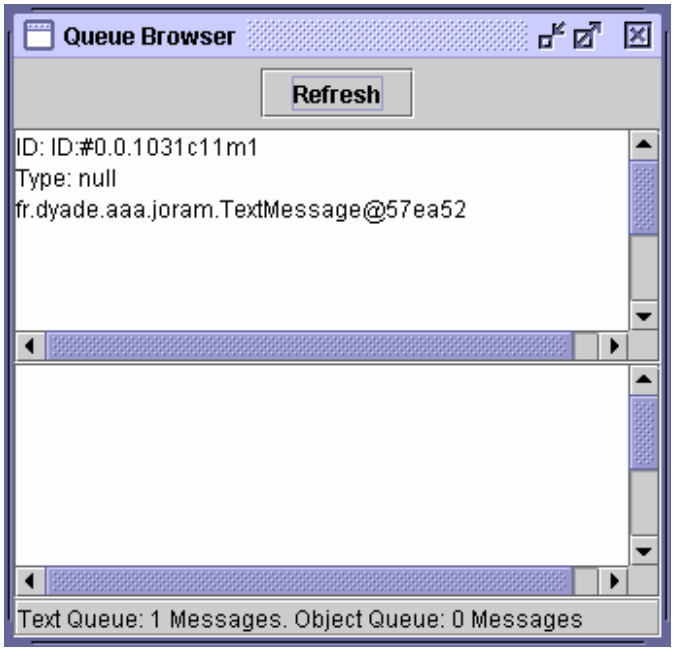
- The VB Test Client is the VB6 equivalent of the Java Test Client which can access the service through both its web service interface and COM interface. This feature immensely helped during debugging (For instance, if there was a problem with the web server or the web service interface, the web service mode of retrieving information would fail, but the COM path would still be open.)
- Additionally the code for this VB6 version was much simpler than the Java version. This is because the SOAP Toolkit works mostly off the WSDL file whereas AXIS prefers it hard-coded or through a config file.

Messaging

MessageSender

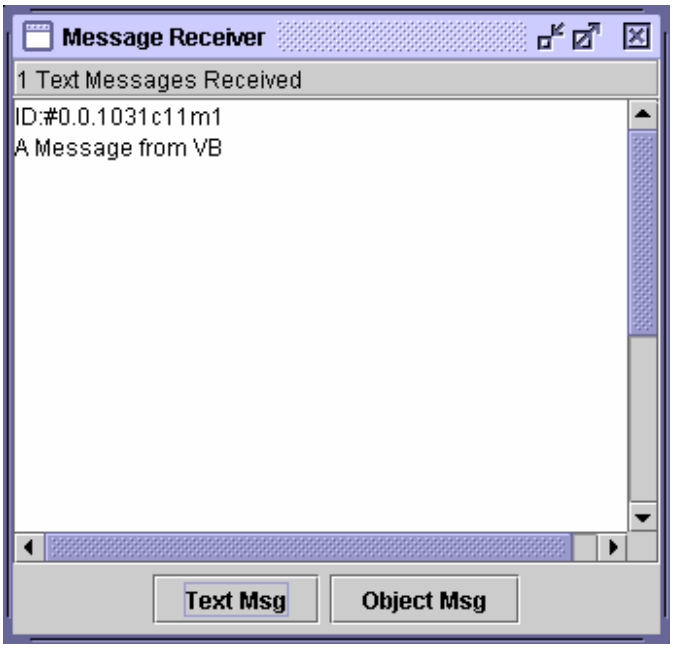
- The Message Sender sends asynchronous message calls. The application has provision for sending both text messages and object (structured XML) messages.
- The TextField displays the status of the last sent message, i.e whether the message was successfully put on the Message Queue or not.

Queue Browser



- The Queue Browser provides a snapshot of the messages currently in the queue. The upper TextArea lists the text messages in queue and the lower lists the object messages.
- The utility uses the JORAM implementation of the JMS API.

Message Receiver



- The Message Receiver is the Java Client Application which is the ultimate destination for all messages.

Note:
All the applications listed in this section have provision to send and receive object message calls. But this is not possible at the present moment due to the problem described in the Messaging section of this report.

6 Conclusion

The work done in Semester One concentrated on the theoretical aspects of interoperability and web services in particular. Building on this, the work done in Semester Two dealt in evaluating the application of web services in solving a 'real world' problem. A pilot implementation revealed a number of interesting caveats. I would like to again list the goals set for a good interoperability solution at the start of this project and comment on the success of the web services solution in meeting those goals.

Solution should work across hardware and software platforms:
Web services were shown to successfully work across software platforms (and in theory across hardware platforms due to their character based nature). Previous attempts at achieving interoperability have mostly been systems which passed binary information making it difficult to migrate software to platforms other than the one on which they were developed.

Good performance across a Wide Area Network:
Measurement of round-trip times (time spent between issuing a request to a web service and receiving a response) seemed to indicate that the bottleneck was in database access. Communication between the client and web service was instantaneous, although it must be noted here that both ran on the test machine and there was no external network to negotiate. Admittedly, web services are a verbose form of communication and further steps will have to be taken to reduce the sizes of requests and responses when on a low bandwidth network. In such cases, one alternative could be to compress the messages. But this will have to be at the cost of clarity and/or processing time.

Minimal development effort:
The development effort hasn't all been smooth sailing. Although SOAP and WSDL are strict standards, the MSSOAP Toolkit and the AXIS Toolkit implement certain features differently (while still remaining within the bounds of the standard). This does present some challenges to start with, but shouldn't trouble the experienced developer. Overall I do believe that the web services solution offers good value for the time spent.

I have worked throughout the year evaluating the potential of web services as an interoperability technology. As long as character based information is being exchanged, web services should serve as a good and easy-to-implement solution.

7 References

Books

1. AXIS - Next Generation Java SOAP
   Romin Irani, S. Jeelani Basha
   May 2002, Wrox Press Ltd.
   ISBN 1-861007-15-9

2. Building Web Services with Java™ - Making Sense of XML, SOAP, WSDL and UDDI
   Steve Graham, Simeon Simenov, Toufic Boubez, Doug Davis, Glen Daniels, Yuichi Nakamura, Ryo Neyama
   March 2002, SAMS Publishing
   ISBN 0-672-32181-5

Other Resources

3. **The Java™ Tutorial**
   A complete and definitive guide to the Java programming language.
4. **Apache AXIS Documentation**
   Documentation for the Apache AXIS SOAP Server.
5. **MSSOAP Toolkit 3.0 Documentation**
   Documentation for the SOAP Toolkit.

8 Appendices

The Appendix includes code I have used to establish the findings made. It includes VB code, java code, WSDL and WSML files.
The code successfully ran on the following machine and software configuration.

Machine Configuration:

CPU	Pentium PIII 600 MHz
RAM	192 MB
Operating System	Windows XP Professional SP1

Software Configuration:

Java side:

JDK version	1.4.1_01
JRE version	1.4.1_01
AXIS version	1.1
JORAM version	3.5.0
Tomcat version	4.1.27

VB Side:

Visual Basic version	6.0
SOAP Toolkit version	3.0
Internet Information Server version (IIS)	5.1

The appendix has been sub-divided into the following sections:

- **Section One**: VB6 Client and a VB6 Service
- **Section Two**: Java Client and a Java Service
- **Section Three**: Java Service and VB Client
- **Section Four**: VB6 Client and Java Service

Section One: VB6 Client ←→ VB6 Service

Client: (MathsHelperClient.vbs)

```
Option Explicit

//Creating a SOAP Toolkit 3.0 Client which encapsulates all web service
//behaviours and also formats a SOAP message to send to the service
Dim soapClient3
set soapclient3 = CreateObject("MSSOAP.SoapClient30")
On Error Resume Next
//Calling the mssoapinit method on the MathsHelper service defined in
//MathsHelper.wsdl file
Call SoapClient3.mssoapinit("MathsHelper.wsdl",  "MathsHelper",
"MathsHelperSoapPort")
if err <> 0 then
  wscript.echo "initialization failed " + err.description
end if

//invoking the add method with the int parameters valued 2 & 3
wscript.echo  SoapClient3.add(2, 3)

//print the SOAP fault message if a fault has occurred
if err <> 0 then
  wscript.echo   err.description
  wscript.echo   "faultcode=" + SoapClient3.faultcode
  wscript.echo   "faultstring=" + SoapClient3.faultstring
  wscript.echo   "faultactor=" + SoapClient3.faultactor
  wscript.echo   "detail=" + SoapClient3.detail
end if
```

Server: (MathsHelper.cls)

```
//A simple function which returns the addition of two integers
Public Function add(ByVal int_1 As Long, ByVal int_2 As Long) As Long
    add = int_1 + int_2
End Function
```

WSDL: (MathsHelper.wsdl created by WSDLGen3.exe)

```
<?xml version='1.0' encoding='UTF-8' ?>
<!-- Generated 05/28/03 by Microsoft SOAP Toolkit WSDL File Generator,
Version 3.00.1325.0 -->
<definitions
      name='MathsHelper'
      targetNamespace='http://amalthea.org/MathsHelper/wsdl/'
      xmlns:wsdlns='http://amalthea.org/MathsHelper/wsdl/'
      xmlns:typens='http://amalthea.org/MathsHelper/type/'
      xmlns:soap='http://schemas.xmlsoap.org/wsdl/soap/'
      xmlns:xsd='http://www.w3.org/2001/XMLSchema'
      xmlns:stk='http://schemas.microsoft.com/soap-toolkit/wsdl-
extension'
```

```
      xmlns:dime='http://schemas.xmlsoap.org/ws/2002/04/dime/wsdl/'
      xmlns:ref='http://schemas.xmlsoap.org/ws/2002/04/reference/'
      xmlns:content='http://schemas.xmlsoap.org/ws/2002/04/content-
type/'
      xmlns:wsdl='http://schemas.xmlsoap.org/wsdl/'
      xmlns='http://schemas.xmlsoap.org/wsdl/'>
<types>
            <schema

      targetNamespace='http://amalthea.org/MathsHelper/type/'
                  xmlns='http://www.w3.org/2001/XMLSchema'
                  xmlns:SOAP-
ENC='http://schemas.xmlsoap.org/soap/encoding/'
                  xmlns:wsdl='http://schemas.xmlsoap.org/wsdl/'
                  elementFormDefault='qualified'>

                  <import
namespace='http://schemas.xmlsoap.org/soap/encoding/'/>
                  <import
namespace='http://schemas.xmlsoap.org/wsdl/'/>
                  <import
namespace='http://schemas.xmlsoap.org/ws/2002/04/reference/'/>
                  <import
namespace='http://schemas.xmlsoap.org/ws/2002/04/content-type/'/>

            </schema>
      </types>

      <message name='MathsHelper.add'>
            <part name='int_1' type='xsd:int'/>
            <part name='int_2' type='xsd:int'/>
      </message>

      <message name='MathsHelper.addResponse'>
            <part name='Result' type='xsd:int'/>
      </message>

      <portType name='MathsHelperSoapPort'>

            <operation name='add' parameterOrder='int_1 int_2'>
                  <input message='wsdlns:MathsHelper.add'/>
                  <output message='wsdlns:MathsHelper.addResponse'/>
            </operation>

      </portType>

      <binding name='MathsHelperSoapBinding'
type='wsdlns:MathsHelperSoapPort' >

            <stk:binding preferredEncoding='UTF-8'/>
            <soap:binding style='rpc'
transport='http://schemas.xmlsoap.org/soap/http'/>

            <operation name='add'>
                  <soap:operation
soapAction='http://amalthea.org/MathsHelper/action/MathsHelper.add'/>
                  <input>
```

```
                        <soap:body
                              use='encoded'

      namespace='http://amalthea.org/MathsHelper/message/'

      encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'
                              parts='int_1 int_2'/>
                  </input>
                  <output>
                        <soap:body
                              use='encoded'

      namespace='http://amalthea.org/MathsHelper/message/'
      encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'
                              parts='Result'/>
                  </output>
            </operation>

      </binding>
      <service name='MathsHelper' >
            <port name='MathsHelperSoapPort'
binding='wsdlns:MathsHelperSoapBinding' >
                  <soap:address
location='http://localhost:8000/MathsHelper/VB/Server/MathsHelper.WSDL'
/>
            </port>
      </service>

</definitions>
```

WSML (MathsHelper.wsml created by WSDLGen3.exe)

```
<?xml version='1.0' encoding='UTF-8' ?>
 <!-- Generated 05/28/03 by Microsoft SOAP Toolkit WSDL File Generator,
Version 3.00.1325.0 -->
  <servicemapping name='MathsHelper'
xmlns:dime='http://schemas.xmlsoap.org/ws/2002/04/dime/wsdl/'>
  <service name='MathsHelper'>
  </service>
</servicemapping>
```

SOAP Request:

```
POST /MathsHelper/VB/Server/MathsHelper.WSDL HTTP/1.1
SOAPAction: "http://amalthea.org/MathsHelper/action/MathsHelper.add"
Content-Type: text/xml; charset="UTF-8"
User-Agent: SOAP Toolkit 3.0
Host: 127.0.0.1
Content-Length: 537
Connection: Keep-Alive
Cache-Control: no-cache
Pragma: no-cache

<?xml version="1.0" encoding="UTF-8" standalone="no"?>
```

```
   <SOAP-ENV:Envelope xmlns:SOAPSDK1="http://www.w3.org/2001/XMLSchema"
xmlns:SOAPSDK2="http://www.w3.org/2001/XMLSchema-instance"
xmlns:SOAPSDK3="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
      <SOAP-ENV:Body SOAP-
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
         <SOAPSDK4:add
xmlns:SOAPSDK4="http://amalthea.org/MathsHelper/message/">
            <int_1>2</int_1>
            <int_2>3</int_2>
         </SOAPSDK4:add>
      </SOAP-ENV:Body>
   </SOAP-ENV:Envelope>
```

SOAP Response

```
HTTP/1.1 100 Continue
Server: Microsoft-IIS/5.1
Date: Wed, 28 May 2003 22:11:30 GMT

HTTP/1.1 200 OK
Server: Microsoft-IIS/5.1
Date: Wed, 28 May 2003 22:11:30 GMT
Content-Type: text/xml; charset="UTF-8"
Content-Length: 539
Expires: -1;

<?xml version="1.0" encoding="UTF-8" standalone="no"?>
   <SOAP-ENV:Envelope xmlns:SOAPSDK1="http://www.w3.org/2001/XMLSchema"
xmlns:SOAPSDK2="http://www.w3.org/2001/XMLSchema-instance"
xmlns:SOAPSDK3="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
      <SOAP-ENV:Body SOAP-
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
         <SOAPSDK4:addResponse
xmlns:SOAPSDK4="http://amalthea.org/MathsHelper/message/">
            <Result>5</Result>
         </SOAPSDK4:addResponse>
      </SOAP-ENV:Body>
   </SOAP-ENV:Envelope>
```

Section Two: Java Client ←→ Java Service

Client: (MathsHelperClient.java)

```
//MathsHelperClient.java

import java.net.URL;
import org.apache.axis.client.Service;
import org.apache.axis.client.Call;
import org.apache.axis.encoding.XMLType;
import javax.xml.rpc.ParameterMode;
import javax.xml.namespace.QName;

public class MathsHelperClient{

  public static void main(String [] args){
    try{
        //Endpoint for the VB Webservice
      /**String endpointURL =
"http://localhost:8002/MathsHelper/VB/Server/MathsHelper.WSDL";**/
        //Endpoint for Java Webservice
      String endpointURL =
"http://localhost:8001/axis/services/MathsHelper";
        //Method name
      String addMethodName = "add";
        //SOAPAction uri
      String uriProperty =
"http://amalthea.org/MathsHelper/action/MathsHelper.add";
      Service service = new Service();
        //Invoking the Addition method of the MathsHelper Web Service
      Call call = (Call) service.createCall();
        //Set the endpoint URL
      call.setTargetEndpointAddress(new java.net.URL(endpointURL));
        //setting the SOAPAction header in the request
      call.setUseSOAPAction(true);
      call.setSOAPActionURI(uriProperty);
        //set the methodName to invoke - add
      call.setOperationName(new
QName("http://amalthea.org/MathsHelper/message/",addMethodName));
        //Seting the Parameters to be passed as to the Web Service
      call.addParameter("int_1",XMLType.XSD_INT,ParameterMode.IN);
      call.addParameter("int_2",XMLType.XSD_INT,ParameterMode.IN);
      call.setReturnType(XMLType.XSD_INT);
      Integer result1 = (Integer) call.invoke(new Object[] {new
Integer(1), new Integer(2)});
        //Print out the result
      System.out.println("Addition: " + result1.intValue());
    }
    catch(Exception e){
      System.err.println(e.toString());
    }
  }
}
```

Service: (MathsHelper.java)

```
public class MathsHelper{
    //constructor
  public MathsHelper(){
  }

    //returns the addition of two numbers
  public int add(int a, int b){
    return a+b;
  }

}
```

WSDL (creted by Java2WSDL)

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="MathsHelper"
xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:apachesoap="http://xml.apache.org/xml-soap"
xmlns:impl="MathsHelper" xmlns:intf="MathsHelper"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">

   <wsdl:message name="addRequest">
      <wsdl:part name="in0" type="xsd:int"/>
      <wsdl:part name="in1" type="xsd:int"/>
   </wsdl:message>

   <wsdl:message name="addResponse">
      <wsdl:part name="addReturn" type="xsd:int"/>
   </wsdl:message>

   <wsdl:portType name="MathsHelper1">
      <wsdl:operation name="add" parameterOrder="in0 in1">
         <wsdl:input message="impl:addRequest" name="addRequest"/>
         <wsdl:output message="impl:addResponse" name="addResponse"/>
      </wsdl:operation>
   </wsdl:portType>

   <wsdl:binding name="MathsHelperSoapBinding"
type="impl:MathsHelper1">

    <wsdlsoap:binding style="rpc"
transport="http://schemas.xmlsoap.org/soap/http"/>
      <wsdl:operation name="add">
         <wsdlsoap:operation soapAction=""/>
         <wsdl:input name="addRequest">
            <wsdlsoap:body
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="MathsHelper" use="encoded"/>
         </wsdl:input>
```

```
         <wsdl:output name="addResponse">
  <wsdl:service name="MathsHelperService">
      <wsdl:port binding="impl:MathsHelperSoapBinding"
name="MathsHelper">
         <wsdlsoap:address
location="http://localhost:8001/axis/services/MathsHelper"/>
      </wsdl:port>
   </wsdl:service>

</wsdl:definitions>
```

SOAP Request

```
POST /axis/services/MathsHelper HTTP/1.0
Content-Type: text/xml; charset=utf-8
Accept: application/soap+xml, application/dime, multipart/related,
text/*
User-Agent: Axis/1.1RC2
Host: 127.0.0.1
Cache-Control: no-cache
Pragma: no-cache
SOAPAction: "http://amalthea.org/MathsHelper/action/MathsHelper.add"
Content-Length: 489

<?xml version="1.0" encoding="UTF-8"?>
   <soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <soapenv:Body>
         <ns1:add
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:ns1="http://amalthea.org/MathsHelper/message/">
            <int_1 xsi:type="xsd:int">1</int_1>
            <int_2 xsi:type="xsd:int">2</int_2>
         </ns1:add>
      </soapenv:Body>
   </soapenv:Envelope>
```

SOAP Response

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Date: Wed, 28 May 2003 22:21:24 GMT
Server: Apache Coyote/1.0
Connection: close

<?xml version="1.0" encoding="UTF-8"?>
   <soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <soapenv:Body>
```

```
         <ns1:addResponse
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:ns1="http://amalthea.org/MathsHelper/message/">
            <addReturn xsi:type="xsd:int">3</addReturn>
         </ns1:addResponse>
      </soapenv:Body>
   </soapenv:Envelope>
```

Section Three: Java Client ←→ VB6 Service

Client (MathsHelperClient.java)

```
Same as MathsHelperClient.java from Section Two
```

Service (MathsHelper.cls)

```
Same as MathsHelper.cls from Section One
```

WSDL (MathsHelper.wsdl)

```
Same as MathsHelper.wsdl from Section One
```

SOAP Request

```
POST /MathsHelper/VB/Server/MathsHelper.WSDL HTTP/1.0
Content-Type: text/xml; charset=utf-8
Accept: application/soap+xml, application/dime, multipart/related,
text/*
User-Agent: Axis/1.1RC2
Host: 127.0.0.1
Cache-Control: no-cache
Pragma: no-cache
SOAPAction: "http://amalthea.org/MathsHelper/action/MathsHelper.add"
Content-Length: 489

<?xml version="1.0" encoding="UTF-8"?>
   <soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <soapenv:Body>
         <ns1:add
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:ns1="http://amalthea.org/MathsHelper/message/">
            <int_1 xsi:type="xsd:int">1</int_1>
            <int_2 xsi:type="xsd:int">2</int_2>
         </ns1:add>
      </soapenv:Body>
   </soapenv:Envelope>
```

SOAP Response

```
HTTP/1.1 200 OK
Server: Microsoft-IIS/5.1
Date: Wed, 28 May 2003 22:11:30 GMT
Content-Type: text/xml; charset="UTF-8"
Content-Length: 539
Expires: -1;

<?xml version="1.0" encoding="UTF-8" standalone="no"?>
   <SOAP-ENV:Envelope xmlns:SOAPSDK1="http://www.w3.org/2001/XMLSchema"
xmlns:SOAPSDK2="http://www.w3.org/2001/XMLSchema-instance"
```

```
xmlns:SOAPSDK3="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
      <SOAP-ENV:Body SOAP-
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
         <SOAPSDK4:addResponse
xmlns:SOAPSDK4="http://amalthea.org/MathsHelper/message/">
            <Result>5</Result>
         </SOAPSDK4:addResponse>
      </SOAP-ENV:Body>
   </SOAP-ENV:Envelope>
```

Section Four: VB6 Client ←→ Java Service

Client (MathsHelperClient.vbs)

```
Same as MathsHelperClient.vbs from Section One
```

Service (MathsHelper.java)

```
Same as MathsHelper.java from Section Two
```

WSDL (MathsHelper.wsdl)

Same as MathsHelper.wsdl from Section One except the following

```
     <wsdl:service name="MathsHelperService">
```

should be replaced by

```
     <wsdl:service name="MathsHelper">
```

SOAP Request

```
POST /axis/services/MathsHelper HTTP/1.1
SOAPAction: ""
Content-Type: text/xml; charset="UTF-8"
User-Agent: SOAP Toolkit 3.0
Host: 127.0.0.1
Content-Length: 500
Connection: Keep-Alive
Cache-Control: no-cache
Pragma: no-cache

<?xml version="1.0" encoding="UTF-8" standalone="no"?>
   <SOAP-ENV:Envelope xmlns:SOAPSDK1="http://www.w3.org/2001/XMLSchema"
xmlns:SOAPSDK2="http://www.w3.org/2001/XMLSchema-instance"
xmlns:SOAPSDK3="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
      <SOAP-ENV:Body SOAP-
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
         <SOAPSDK4:add xmlns:SOAPSDK4="MathsHelper">
            <in0>2</in0>
            <in1>3</in1>
         </SOAPSDK4:add>
      </SOAP-ENV:Body>
   </SOAP-ENV:Envelope>
```

SOAP Response

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Transfer-Encoding: chunked
Date: Wed, 28 May 2003 22:33:48 GMT
Server: Apache Coyote/1.0

<?xml version="1.0" encoding="UTF-8"?>
   <soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
```

```
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <soapenv:Body>
         <ns1:addResponse
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:ns1="MathsHelper">
            <addReturn xsi:type="xsd:int">5</addReturn>
         </ns1:addResponse>
      </soapenv:Body>
</soapenv:Envelope>
```