

Fachgebiet für Offene
Kommunikationssysteme (OKS)

VHE Web Services

Project in WS 2002/03

Björn Schünemann
(schueni@cs.tu-berlin.de)

Table of Contents

1. Introduction	3
2. Context Manager	4
2.1 Creating Client	6
2.2 Test this Service	6
3. Evaluation of supported Types	6
3.1 Toolkits in Axis and WSDP	7
3.2 Creating Test Service	8
3.2.1 Writing Interface and Building WSDL with Axis Tool	8
3.2.2 Building and Deploying a Service with Axis and JBoss.....	9
3.3 Clients with Axis	11
3.3.1 Creating Axis stubs	11
3.3.2 Building and Running Client with Axis	11
3.4 Clients with WSDP	12
3.4.1 Creating WSDP stubs.....	12
3.4.2 Building and Running Client with WSDP	13
3.5 Supported Types.....	14
4. Conclusion.....	16
Appendix A: Used Application.....	17
Appendix B: Setting Up	17
Appendix C: Authors Address	17
Appendix D: Bibliography.....	17

1. Introduction

Web Services, in the general meaning of the term, are services offered via the Web. In a typical Web Services scenario, a business application sends a request to a service at a given URL. The service receives the request, processes it, and returns a response.

Web Services are self-contained pieces of code that have three distinguishing properties:

1. They communicate in an interoperable XML protocol, such as SOAP.
2. They describe themselves in an interoperable XML meta-format, such as WSDL.
3. They are able to federate globally through XML-based registry services, such as UDDI.

Web services depend on the ability of parties to communicate with each other even if they are using different information systems. They also depend on the ability using different computing platforms to communicate with each other. In addition to data portability and code portability, Web services need to be scalable, secure, and efficient, especially as they grow.

There are Java-based Web Services (e.g. constructed with Axis or WSDP) and non-Java based Web Services (e.g. constructed with .NET).

The Simple Object Access Protocol (SOAP) is an RPC-protocol that uses XML as the encoding language and that uses standard internet protocols, such as HTTP and SMTP as the transport medium. It's a lightweight protocol for exchange of information in a decentralized, distributed environment.

For SOAP, there exists also corresponding meta-language that is called Web Service Description Language (WSDL). WSDL has a similar function than the Interface Definition Language has in Corba. It describes service interfaces (port types) in terms of the operations (request-messages and response-messages) that can be invoked. It describes particular service instances by binding a port-type to a particular transport endpoint.

The Universal Description Discovery Integration (UDDI) is a standardized interface to Web Service registries - a kind of global and platform neutral version of JNDI in which WSDL descriptions are hosted and can be searched. UDDI registries are themselves Web Services in that they support the SOAP protocol.

We can distinguish two modes of XML/SOAP messaging. The simplest, but very inconvenient mode is to interact with the XML/DOM-documents representing method requests and responses immediately. It is suitable for untyped applications or special-purpose logic that is based on XML-processing anyway. There is an API under development, JAXM (Java API for XML Messaging), which will support this mode.

Whenever we are not primarily interested in manipulating XML and have a strongly-typed environment, such as in J2EE logic, we need a more elaborated mode that is an extension to Remote Method Invocation (RMI). The corresponding extension API to the Java2 platform is called JAX-RPC. In JAX-RPC, an XML-message is processed through a chain of so-called handlers before it is (de-)serialized into/from Java objects using a set of so-called type-mappings. Type-mappings associate a Java class and a corresponding XML type by dedicated Serializer and Deserializer implementations. This is a very powerful, yet extensible pattern and you see that even SUN can learn from flaws they built-into their native serialization API.

2. Context Manager

The **VHE Platform** provides functional components for the seamless service provision in heterogeneous network environment having regard of network and terminal capabilities. It provides such features like **Location Based Support, Accounting, Billing, Access Control, Subscription, Session Management, Personalization and Adaptation.**

The Context Manager builds a facade of the VHE Portal for the VHE Services. Through the Context Manager the VHE Services can retrieve the user preferences and session data. The VHE Service has a single point in order to get the whole information. The appropriated data can be retrieved through the user context id, which holds such information like time, location, user session and the current VHE service.

The next topics describe how to write a client using the web service interface of the ContextManager. This section describes the methods which are called by the web service client.

public Object getSessionData(String userContextID, String key) throws RemoteException;

The VHE service developer has an opportunity to save the session data in the VHE Portal service session object.

public void putSessionData(String userContextID, String key, Object value) throws RemoteException;

This method provides the modification and storage of serializable objects.

public Integer registerUser(String userID, String firstName, String lastName) throws RemoteException;

User has to be registered for the personalized service usage. The registerUser-method creates a new UserRecord and returns 0 as a result, if the userID is unique, if such userID already exist a -1 is returned.

public Integer setUserID(String userContextID, String userID, Boolean addCurrentTerminalID) throws RemoteException;

For the personalization of the VHE Services the user has to be identified at first. So the VHE Service Provider has an opportunity to make a relation between the user and the current user context. If the user is a registered user in the VHEPortal, the method setUserID returns 0, if user is not registered -1. Optionally the current terminalID can be added to the user personal info. If the addCurrentTerminalID is true so the terminalID will be added to the user profile info, if false than not.

public String getUserID(String userContextID) throws RemoteException;

getUserID returns the current user identifier, for anonymous VHE Portal usage user id is set to 'Gast'. User 'Gast' is not a registered user! That means there is not particular user info to this user.

public HashSet getProfileNames(String userContextID) throws RemoteException;

A user has always an opportunity the select one of his profiles.

public Map getUserInformation(String userContextID) throws RemoteException;

According to the user context id, the VHE service provider can retrieve the user information e.g. name, last name, contact info ...

public Map getGlobalPreferencesSet(String userContextID, String profileCN) throws RemoteException;

The VHE service developer can read, but not write the global preferences set. The profile name is not necessary, than it can be chosen automatically.

public Map getServiceDefaults(String userContextID, String profileCN) throws RemoteException;

The VHE service developer can retrieve the defaults situated in the defaults-file, which is described at the beginning of this document.

public Map getServicePreferencesSet(String userContextID, String profileCN) throws RemoteException;

This method provides the access to the service preferences set.

public Serializable getAttribute(String userContextID, String profileCN, String attributeCN) throws RemoteException;

The VHE service developer can retrieve not only the whole set of the attributes situated in the service specific profile, but also only one “stand alone” attribute. The attribute must be declared in default-ServicePreferencesSet!

public String getAttributeValue(String userContextID, String profileCN, String attributeCN) throws RemoteException;

The VHE service developer can retrieve with the help of this function a string value of an attribute.

public HashSet getAttributeValueSet(String userContextID, String profileCN, String attributeCN) throws RemoteException;

The VHE Developer can retrieve not only a string value of an attribute, but also a value set, if the attribute contains a set of values.

public void setAttribute(String userContextID, String profileCN, String attributeCN, String attributeValue) throws RemoteException;

This method modifies the available attributes declared in the default-ServicePreferencesSet.

public void setAttribute(String userContextID, String profileCN, String attributeCN, Set attributeSet) throws RemoteException;

In this case complex attribute consisting of HashSet can be stored in the service preferences.

public void deleteServicePreferencesSet(String userContextID, String profileCN) throws RemoteException;

Removes the service preferences set from the user profile.

Note: Information about creating the client is not so detailed in the next sections. The chapter about the test service goes into more detail and shows the creation step by step.

contextManagerServiceAxisClient.zip contains the source code for the VHE service client. See readme.txt for more information.

2.1 Creating Client

Because of problems with automatic generation of the WSDL file (if you tack on “?wsdl” to the end of the URL, the service will automatically generate the service description normally), it’s necessary to write a Java interface that describes the methods of the web service. You find the code of the interface in

```
contextManagerServiceAxisClient\ContextManagerServiceWSDL\  
ContextManagerServiceInterface.java.
```

contextManagerServiceAxisClient\generateWSDL.bat compiles the code and generates the WSDL file in

```
contextManagerServiceAxisClient\ContextManagerServiceWSDL\  
ContextManagerService.wsdl.
```

Now it is possible to create the stubs with the WSDL2Java Axis tool.

contextManagerServiceAxisClient\generateStubs.bat does that for you and writes the stubs in folder

```
contextManagerServiceAxisClient\ContextManagerService\.
```

After generating stubs it’s easy to create the client with the remote method calls of the web service. The client is implemented in contextManagerServiceAxisClient\
ContextManagerService\Test.java.

contextManagerServiceAxisClient\compileClient.bat compiles the client.

contextManagerServiceAxisClient\generateJavaDoc.bat creates the javadoc documentation in contextManagerServiceAxisClient\doc\.

contextManagerServiceAxisClient\startTestClient.bat starts the client.

The client is a simple non-graphical application. The user chooses the method is calling on the server.

2.2 Test this Service

The server providing the Context Manager Service is not reachable outside FOKUS (service creator). There is no way to test the client within the period of coding in detail. That’s why bugs can exist in the program code. The server will be reachable in future.

3. Evaluation of supported Types

To call a remote procedure, the client program invokes a method on a stub, a local object that represents the remote service. The stub invokes routines in the JAX-RPC runtime system.

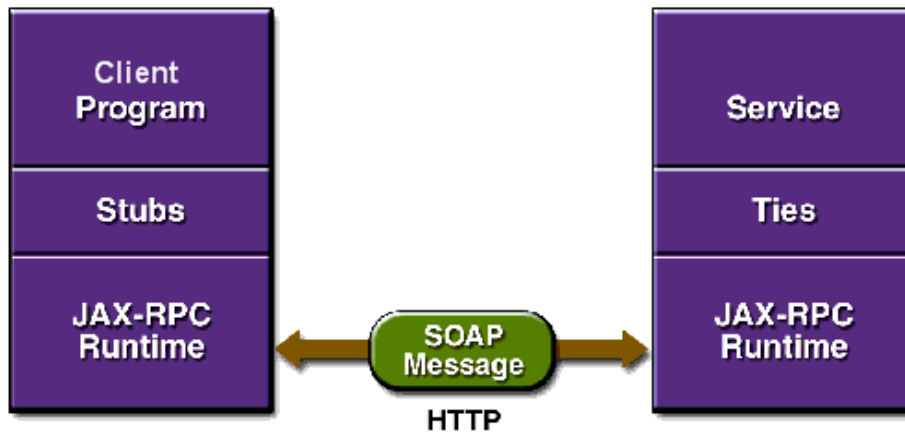
The runtime system converts the remote method call into a SOAP message and then transmits the message as an HTTP request.

When the server receives the HTTP request, the JAX-RPC runtime system extracts the SOAP message from the request and translates it into a method call. The JAX-RPC runtime system

invokes the method on the tie object. The tie object invokes the method on the implementation of the service.

The runtime system on the server converts the method's response into a SOAP message and then transmits the message back to the client as an HTTP response.

On the client, the JAX-RPC runtime system extracts the SOAP message from the HTTP response and then translates it into a method response for the program.



JAX-RPC converts Java classes to XML data types. For example, JAX-RPC maps the `java.lang.String` class to the `xsd:string` XML data type. Application developers should be aware that not every class in the Java 2 Standard Edition (J2SE) can be used as a method parameter or return type in JAX-RPC.

The next sections describe how to test the mapping of different Java classes to SOAP/XML types. `testDataTypes.zip` contains the source code. See `readme.txt` for more information.

Note: The service is implemented using Axis. WSDP has problems with WSDL files generated by Axis in some cases. That's why a few classes can't transmit between WSDP client and Axis service. WSDP clients and services should not have this problem.

3.1 Toolkits in Axis and WSDP

Axis is an Open Source implementation of the JAX-RPC API from Apache. It realizes the basic concepts of handlers and type-mappings driven by a lightweight SAX-based processing engine. Axis comes with a dedicated deployment format called Web Service Deployment Descriptor (WSDD) which configures the runtime engine with new request/response flows, service providers, and type-mappings. Axis comes with a ready-made http transport listener in the form of a Servlet that can be installed in any compliant web-container.

The Java2WSDL and WSDL2Java tools make it easy to develop a new web service:

- Use the Java2WSDL tool to create a WSDL file from an interface
- Use a WSDL file to build with the WSDL2Java tool the appropriate client/server bindings for the web service

The Java Web Services Developer Pack (Java WSDP) is an all-in-one download from SUN containing key technologies to simplify building of Web services using the Java 2 Platform. The technologies comprising the Java WSDP are among others:

- Java API for XML-based RPC (JAX-RPC)

WSDP contains tools to create client/server bindings:

- `wsdeploy`: generates the ties (server-side bindings) and the WSDL file
- `wscompile`: generates the stubs (client-side bindings)

3.2 Creating Test Service

The test service is a simple service without useful scenario. It contains many methods with different parameters and return types. The only sense is testing support of these classes in Axis and WSDP.

Note: This is not part of the client developers.

3.2.1 Writing Interface and Building WSDL with Axis Tool

The Java2WSDL emitter makes it easy to develop a WSDL for a web service. Write and compile a Java interface (or class) that describes the web service interface.

`testDataTypes\wsdl\TestWSDLInterface.java` describes a web services with many different data types for method parameters and return values.

```
package wsdl;
/**
 * Interface for creating test wsdl for testing supported types
 *
 * @author Bjoern Schuenemann (schueni@cs.tu-berlin.de)
 */
public interface TestWSDLInterface {

    public boolean primitives1(byte arg1) throws RemoteException;
    public byte primitives2(boolean arg1) throws RemoteException;
    public double primitives3(float arg1) throws RemoteException;
    public float primitives4(double arg1) throws RemoteException;
    public int primitives5(long arg1) throws RemoteException;
    ...
}
```

Note: If you compile your class with debug information, Java2WSDL will use the debug information to obtain the method parameter names.

Now you can use the Java2WSDL tool to create a WSDL file from the interface above.

`testDataTypes\generateWsd1.bat` will do that for you:

```

echo - compiling TestWSDLInterface.java
javac -g -classpath ... wsdl\TestWSDLInterface.java

echo - generating WSDL
java -classpath ... org.apache.axis.wsdl.Java2WSDL
    -o wsdl\Test.wsdl
    -l"http://localhost:8080/Test"
    -n "urn:Test"
    -p"Test" "urn:Test"
    wsdl.TestWSDLInterface

```

- classpath includes all necessary axis jar files
- o indicates the name of the output WSDL file
- l indicates the location of the service
- n is the target namespace of the WSDL file
- p indicates a mapping from the package to a namespace
- the class specified contains the interface of the web service

The output WSDL document will contain the appropriate WSDL types, messages, portType, bindings and service descriptions to support a SOAP rpc, encoding web service. If a specified interface methods reference other classes, the Java2WSDL tool should generate the appropriate xml types to represent the classes and any nested/inherited types. The tool should support JAX-RPC complex types (bean classes), extension classes, enumeration classes, arrays and Holder classes. More information about the real support of these features follows in the next sections.

3.2.2 Building and Deploying a Service with Axis and JBoss

Write and compile a web service implementing the service interface above. You will find this code in `testDataTypes\Service\TestService.java`.

```

// TestService.java

package Service;

import ...

/**
 * Class for creating webservice to test supported types
 *
 * @author Bjoern Schuenemann (schueni@cs.tu-berlin.de)
 */
public class TestService {

    // primitives
    public boolean primitives1(byte arg1) throws RemoteException{
        if (arg1 <= 10) return true;
        else return false;
    }
    public byte primitives2(boolean arg1) throws RemoteException{
        if (arg1 == true) return 10;
        else return 20;
    }
    public double primitives3(float arg1) throws RemoteException{
        return (double)arg1;
    }
    ...
}

```

Copy `TestService.class` to `testDataTypes\axis.war\WEB-INF\classes\Service`. The next step is to copy the `axis.war` folder to `%jboss%\server\default\deploy\`, where `%jboss%` is the home directory of JBoss. `axis.war` contains the axis toolkit and also the web service (see last step). Now start JBoss, by going to `%jboss%\bin` and executing `run.bat`. After the server has started successfully, the web service would have been deployed.

It is also necessary to deploy our web service using the AdminClient of Axis. But first you have to write a Web Service Deployment Descriptor (WSDD). A deployment descriptor contains a bunch of things you want to "deploy" into Axis - i.e. make available to the Axis engine. You will find the deployment descriptor in `testDataTypes\Service\deploy.wsdd`:

```
<deployment
  xmlns="http://xml.apache.org/axis/wsdd/"
  xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">

  <service name="TestService" provider="java:RPC">
    <parameter name="className" value="Service.TestService"/>
    <parameter name="allowedMethods" value="*" />
  </service>

</deployment>
```

The outermost element tells the engine that this is a WSDD deployment, and defines the "java" namespace. Then the service element actually defines the service for us. Our provider is "java:RPC", which is built into Axis, and indicates a Java RPC service. The actual class which handles this is `org.apache.axis.providers.java.RPCProvider`. We need to tell the `RPCProvider` that it should instantiate and call the correct class (e.g. `Service.TestService`), and we do so by including `<parameter>` tags, giving the service one parameter to configure the class name, and another to tell the engine that any public method on that class may be called via SOAP (that's what the "*" means; we could also have restricted the SOAP-accessible methods by using a space or comma separated list of available method names).

Once we have this file, we need to send it to an Axis server in order to actually deploy the described service. We do this with the AdminClient, or the "org.apache.axis.client.AdminClient" class. If you have deployed Axis on a server other than Tomcat, you may need to use the `-p <port>` argument. The default port is 8080.

A typical invocation of the AdminClient looks like the command in `testDataTypes\deployService.bat`:

```
echo - deploying service
java -cp ... org.apache.axis.client.AdminClient Service\deploy.wsdd
```

If successfully deployed, you can invoke <http://localhost:8080/axis/services/TestService> in your web browser. If you access this service URL in a browser, you'll see a message indicating that the endpoint is an Axis service, and that you should usually access it using SOAP. However, if you tack on "?wsdl" to the end of the URL, Axis will automatically generate a service description (wsdl) for the deployed service, and return it as XML in your browser. The resulting description will be used for creating WSDP stubs, described later.

3.3 Clients with Axis

The client is a simple non-graphical application. The user chooses the method is calling on the server. The result of the method call will be displayed.

After creating the stubs it isn't complicated to write the method call.

3.3.1 Creating Axis stubs

The Java tool in "org.apache.axis.wsdl.WSDL2Java" generates the client-side bindings. These stubs are necessary for the communication between service and client via SOAP.

The basic invocation form looks like in `testDataTypes\createAxisStubs.bat`:

```
echo - generating client-side bindings for Axis
java -cp ... org.apache.axis.wsdl.WSDL2Java -o Axis\ wsdl\Test.wsdl
```

This will generate only those bindings necessary for the client. AXIS follows the JAX-RPC specification when generating Java client bindings from WSDL. The output directory for the stubs is `Axis\`.

3.3.2 Building and Running Client with Axis

Now it's time to code the client. The completely code you can find in `testDataTypes\TestClients\TestClientAxis.java`. `testDataTypes\startAxisClient.bat` will start the client.

Here are some important code fragments of the client:

```
package TestClients;

...
import org.apache.axis.client.Call;
import org.apache.axis.client.Service;
import javax.xml.namespace.QName;

public class TestClientAxis {

    private static Service service = new Service();

    /** address of the test Web-Service */
    private static final String endpoint =
        "http://localhost:8080/axis/services/TestService";

    public static void main(String[] args) throws IOException {
        ...
        Boolean ret =
            (Boolean) invokeOperation("primitives1", Boolean.class,
                                     new Object[] { new Byte(arg1) } );
        ...
    }
}
```

```

/**
 * Invokes operation passed by <code>operationName</code>
 * on web service <code>ContextManagerService</code>.
 *
 * @param operationName the operation name.
 * @param returnClass the operation's return class.
 * @param params the invocation parameters.
 * @return result of invocation.
 */
private static Object invokeOperation(String operationName,
                                     Class returnClass, Object[] params) {
    try {
        // The call object is used to actually invoke the web service.
        Call call = (Call)service.createCall();
        // Sets the call object's endpoint address
        call.setTargetEndpointAddress( new java.net.URL(endpoint) );
        // Sets the operation name associated with this Call object.
        // The QName is the qualified name of the method.
        call.setOperationName(new QName(operationName));
        if (returnClass != null) {
            call.setReturnClass(returnClass);
        }
        return call.invoke(params);
    }
    ...
..} // End: invokeOperation
}

```

First we create a new `Service` object and then a `Call` object in `invokeOperation`. These are the standard JAX-RPC objects that are used to store metadata about the service to invoke. We set up our endpoint URL - this is the destination for our SOAP message. We define the operation (method) name of the Web Service in `invokeOperation`. And we actually invoke the desired service, passing in an array of parameters - in this case just one `Byte`.

3.4 Clients with WSDP

The generation of the stubs and the call of the remote method are different to Axis.

3.4.1 Creating WSDP stubs

The WSDP stubs will be generated with the `wscmpile` tool. `testDataTypes\createWSDPStubs.bat` run this tool as follows:

```

echo - generating stubs for WSDP
wscmpile -gen:client -d WSDP wsdl\config.xml

```

The `-gen:client` option instructs `wscmpile` to generate client-side classes such as stubs. The `-d` option specifies the destination directory of the generated files.

The `wscmpile` tool generates files based on the information it reads from `config.xml` file. The location of the `wSDL` file is specified by the `<wSDL>` element of the `config.xml` file, which is in `\testDataTypes\wSDL\`:

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration xmlns="http://java.sun.com/xml/ns/jax-rpc/ri/config">
  <wSDL location=
    "http://localhost:8080/axis/services/TestService?wSDL"
    packageName="Test"/>
</configuration>
```

3.4.2 Building and Running Client with WSDP

`testDataTypes\startWSDPClient.bat` will start the client `testDataTypes\TestClients\TestClientWSDP`. `TestClientWSDP` calls the methods of the service. It makes the calls through a stub, a local object which acts as a proxy for the remote service. Because the stubs is created before runtime (by `wscmpile`), it is usually called a static stub.

The source code for `TestClientWSDP` contains:

```
package TestClients;

...
import javax.xml.rpc.Stub;
import Test.*;

public class TestClientWSDP {

    public static void main(String[] args) throws IOException {
        ...
        try {
            // create stub
            Stub stub =
                (Stub)(new TestServiceService_Impl().getTestService());
            // cast stub to service definition interface
            TestService testService = (TestService)stub;
            ...

            boolean ret = testService.primitives1(arg1);
            ...
        }
        ...
    }
}
```

To create the stub, the client invokes the method `getTestService()`. The code of this method is implementation-specific and might not be portable because it relies on the `TestServiceService_Impl` object. The `TestServiceService_Impl` class was generated by `wscmpile` in a preceding section.) After it creates the stub, the client program casts the stub to the type `TestService`, the service definition interface.

3.5 Supported Types

Here is the result of evaluation of supported types:

Types supported by Axis and WSDP

Primitives:

- boolean, byte, double, float, int, long, short
- arrays with member of this types

It is possible to use the relevant wrapper classes instead primitives in Axis clients even if the remote method of the service use primitives.

J2SE SDK Classes:

- java.lang.Boolean (cast to boolean in WSDP)
- java.lang.Byte (cast to boolean in WSDP)
- java.lang.Double (cast to double in WSDP)
- java.lang.Float (cast to float in WSDP)
- java.lang.Integer (cast to int in WSDP)
- java.lang.Long (cast to long in WSDP)
- java.lang.Short (cast to short in WSDP)

It is possible to use primitives instead the relevant wrapper classes in Axis clients even if the remote method of the service use the relevant wrapper class.

It is necessary to cast to the relevant primitives in the method call of the WSDP client. Using wrapper classes produces an error message even if the service method used this class.

- java.lang.String
- java.math.BigDecimal
- java.math.BigInteger
- java.util.Calendar
- java.util.Date (use Calendar in WSDP)

WSDP clients don't support Date, use Calendar instead.

Arrays of J2SE SDK Classes:

- java.lang.Boolean[] (cast to boolean[])
- java.lang.Byte[] (cast to byte[])
- java.lang.Double[] (cast to double[])
- java.lang.Float[] (cast to float[])
- java.lang.Integer[] (cast to int[])
- java.lang.Long[] (cast to long[])
- java.lang.Short[] (cast to short[])

Cast the members of the array to relevant primitives on client side in Axis and WSDP. Wrapper classes are not supported.

- java.lang.String[]
- java.math.BigDecimal[]

- java.math.BigInteger[]
- java.util.Calendar[]

Namespaces:

- QName
- QName[]

Arbitrary Objects:

- registered serializer for these objects required (see Axis and WSDP documentation)

Classes with xml schema anyType in wsdl file

Axis generates for this classes xml schema anyType in wsdl file. This can cause to errors (e.g. client and service are created with different application).

- List
- ArrayList
- LinkedList
- Stack
- AbstractList
- Properties
- TreeMap
- AbstractMap
- Attributes
- IdentityHashMap
- RenderingHints
- WeakHashMap
- Set
- HashSet
- TreeSet
- AbstractSet
- LinkedHashSet
- Serializable

Not supported by Axis

- char
- Character
- Properties
- TreeMap
- Attributes
- IdentityHashMap
- RenderingHints
- WeakHashMap
- AbstractSet
- arrays of collection subinterfaces
- arrays of Serializable
- arrays of java.util.Date

Note: These classes can't be tested with WSDP because the test service is generated with Axis.

Supported by Axis and not supported by WSDP

Collection Subinterfaces (.NET cannot handle them):

WSDP can't interpret (from Axis generated) WSDL that contains mappings for these classes:

- Vector
- Map, HashMap, Hashtable

The remote method call results with an error in WSDP clients for following data types:

- List, ArrayList, LinkedList, Stack, AbstractList
- AbstractMap
- Set, HashSet, TreeSet, LinkedHashSet

Note: The service is implemented using Axis. WSDP has problems with WSDL files generated by Axis in some cases. That's why a few classes can't transmit between WSDP client and Axis service. WSDP clients and services should not have this problem.

4. Conclusion

A key feature of Web Services is the ability to communicate with each other even if they are using different information systems and computing platforms. To call a remote procedure, the client program invokes a method on a stub. The stub invokes routines in the JAX-RPC runtime system. The runtime system converts the remote method call into a SOAP message and then transmits the message as an HTTP request. This mapping between Java data types and SOAP/XML types works well while client and services are created with the same Web Service application. Using different applications can cause problems.

Appendix A: Used Application

- Java(TM) 2 SDK, Standard Edition Version 1.4.1 (<http://java.sun.com/>)
- JBoss 3.0.4 (<http://www.jboss.org/>)
- Apache Axis 1.0 (<http://xml.apache.org/axis/>)
- Java Web Services Developer Pack (WSDP) 1.0_01 (<http://java.sun.com/webservices/webservicespack.html>)

Appendix B: Setting Up

- Installing Java(TM) 2 SDK
- Installing JBoss
- Installing Java WSDP
- Add the bin directories of the Java WSDP and Java SDK installations to the PATH environment variable
- Change the value of the WSDP variable in `testDataTypes\startWSDPClient.bat` to your WSDP home directory

The installation of Axis is not necessary because all used components are integrated in the zip-file.

Appendix C: Authors Address

Björn Schünemann

TU-Berlin

e-mail: schueni@cs.tu-berlin.de

Appendix D: Bibliography

- Axis User's Guide (<http://cvs.apache.org/viewcvs.cgi/~checkout~/xml-axis/java/docs/user-guide.html>)
- The Java Web Services Tutorial (<http://java.sun.com/webservices/docs/1.1/tutorial/doc/index.html>)
- JBOSS 3.0 GETTING STARTED (<http://belnet.dl.sourceforge.net/sourceforge/jboss/QuickStart-30x.pdf>)
- Apache Axis 1.0 Tutorial, Part III: EJB and Web Services Integration (<http://www.ammai.com/modules.php?op=modload&name=Sections&file=index&req=listarticles&secid=2>)
- Simple Object Access Protocol (SOAP) 1.1 (<http://www.w3.org/TR/SOAP/>)
- Web Services Description Language (WSDL) 1.1 (<http://www.w3.org/TR/wsdl>)
- FhG FOKUS: "VHE Portal and Platform Architecture"