WS-Types

Authors:
Ruwan Wijesinghe, Virtusa Corp.
Tyrell Perera, Virtusa Corp.

Introduction:

Here we suggest a new revolutionary web service specification that will totally change how people use web services or how people develop web service client applications. Currently web services do not support OOP style programming. Today, the objects returned from the web services only have properties. They do not have any methods.

This new technology will enable web services to return objects with both properties and methods. When these methods are called they will call some other web services. Since the service URL of the web services can be assigned dynamically, we can use them to develop service brokers that work seamlessly. Once this specification is implemented, a code that would do the web service access would look like this.

```
//This is the serviceType of
// http://companyServer/CompanyInfo web service located at US
CompanyInfo compInfo = new CompanyInfo();

//Branch1 information is handled by branch1Server server
//located at another server in same intranet
BranchInfo branch1Info = compInfo.getBranchInfo("Branch1");

//Branch2 information is handled by branch2Server server
//located at UK
BranchInfo branch2Info = compInfo.getBranchInfo("Branch2");

Employee[] employees = branch1Info.getEmployeesRequstingTransfers();

for(int i=0; i< employees.length; i++){
      ServiceRecords serviceRec = employees[i].getServiceRecords();
      if (isEligibleToTransfer(serviceRec)){
            branch1Info.removeEmployee(employees[i]);
            branch2Info.addEmplyee(employees[i]);
      }
}
```

Sample 1: Demonstrate the use in enterprise applications

```
//This is the serviceType of ManagementService web service
ManagementService manService = new ManagementService();

//This object referes to a web service in the mail server
MailServer mailServer = manService.getMainMailServer();
MailFolder mailFolder =
            mailServer.getFolder(userName, password, folderName);

MailMessage[] messages = mailFolder.getMessagesBySender(sender);

for(int i=0; i<messages.length; i++){
      DisplayMessage( messages[i] );
}
```

Sample 2: Demonstrate the use in management applications

Objectives:

- Enable client stub generation tools to create a class hierarchy to access web services. These classes must have both properties and methods and must be able to pass between different web services
- Enable the development of enterprise wide or world wide shared type systems. The objects of these types can be transferred between web services that are based on these shared type systems
- Enable the seamless transition between different web services. This is optimal for broker patterns
- Enable client side tools to easily generate a complete help documentation of the client side class hierarchy

Sub Objectives:

- Enable the transition to WS-Types simple. To achieve this,
    - Make it possible to transfer an existing web service into WS-Types service with basic features, without changing a single line of code (Just publishing a WSTD file in a web share is sufficient for this)
    - Make it easy to convert existing web services to WS-Types service with full features by sampling adding a filter that will add some SOAP headers
    - Making it possible for the existing web service proxies and proxies generated using none OOP languages to function correctly even after the conversion to full featured WS-Types service

Basic Components:

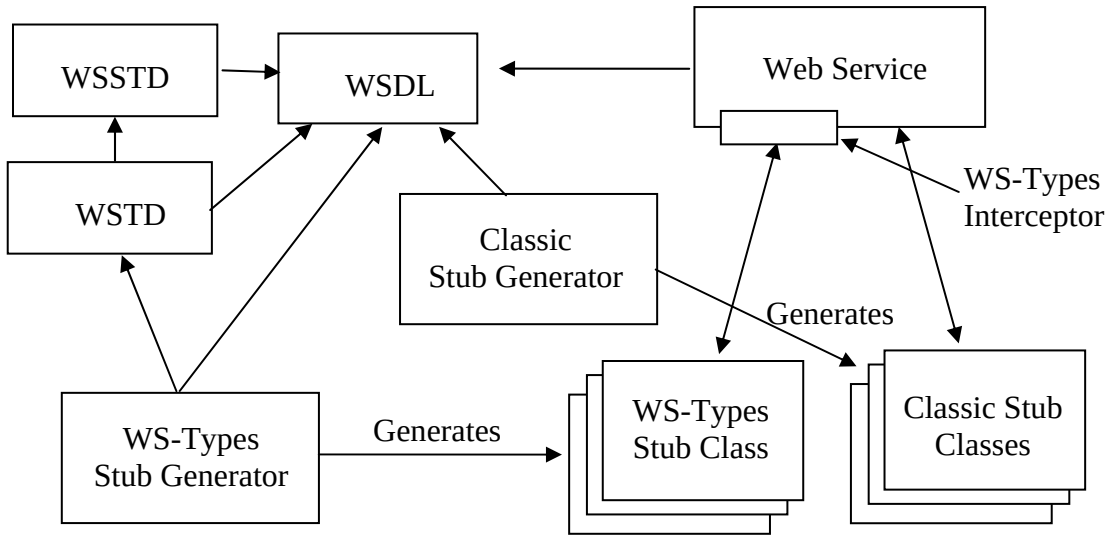

Fig. 1 WS-Types Components

Classic web service stub generators are pointed at WSDL. However WS-Types stub generators are point to a WSTD document available in at a web share. WSTD documents may import WSSTD documents which contains shared type definitions. WS-Types stub generators uses the meta-data available in these document and generate a separate stub classes for each type described in these documents. The property mapping and method call information in WSTD and WSSTD documents are hard coded in methods of these stub classes.

WS-Types stub classes can access web services that support WS-Types as well as Web Services that do not. The web services that support WS-Types add special SOAP headers to the reply messages and understand the WS-Types headers in the request messages. This can be implemented using WS-Types interceptors (filters) added to the web service framework. Even though web services that do not support WS-Types can be used by WS-Types stub classes, in order to use the features like dynamically assigned service URLs, object version numbers, object expiration, etc. we need we need the web service to support WS-Types specification.

WSTD – Web Service Type Description

A simple sample WSTD document is given below (This document must be available in a web share accessible for the stub class generator),

```
<?xml version="1.0" encoding="utf-8"?>
<typeDescription xmlns="http://sample.org/ws-types">
        <types>
                <import url="http://www.sampleserver.com/testservice/shared.wstd" />
                <entryType binding="tns:ServiceSoap"
                                xmlns:tns="http://www. myserver.com/tstservice/service1"
                                wsdl="http://www. myserver.com/tstservice/service1?wsdl"
                                serviceUrl ="http://www. myserver.com/tstservice/service1>
                        <documentation>
                            <document type="xml"
                                url="http://www.myserver.com/tstservice/Service1Desc.xml" />
                          <document type="html"
                                url="http://www.myserver.com/tstservice/Service1Desc.html" />
                        </documentation>
                        <method operation="getEmployee">
                              <inputMessage>
                                    <parameter name=”name”
                                          valueSourceType="input" valueSourceId = "name" />
                              </inputMessage>
                        </method>
                </entryType>
        </types>
</typeDescription>
```

Types Element

Three types of elements can exist in the “types” element of a WSTD document. They are “entryType”, “import” and “type”. “entryType” elements describe the entry types. “type” elements describe the normal types and “import” element imports the types from a WSSTD document.

Entry Types

Entry types are special types which act as entry points to the web service type system. These types have fixed service URLs. The stub classes of these types can be initiated in the client side and used as the starting point of the web service access. These types should not have any properties and resembles a normal stub class that is used today.

In order to make it possible to create an enterprise wide (or sometimes world wide) type system, all the other types are defined in a separate shared document called WSSTD document. These shared type documents are imported in WSTD documents. The only difference between WSTD and WSSTD documents is that WSTD documents must contain at least one entry type, where as WSSTD documents do not have any such restrictions.

Documentation section in this document gives information required to generate help documents in the client side (this is an important step to give easy access usage information to every developer)

WSSTD – Web Service Shared Type Description

The shared WSSTD document imported in this documentation is given here.

```
<?xml version="1.0" encoding="utf-8"?>
<typeDescription xmlns="http://sample.org/ws-types">
   <documentation>
        <document type="xml"
                url="http://www.sampleserver.com/testService/SharedTypesDesc.xml" />
        <document type="html"
                url="http://www.sampleserver.com/testService/SharedTypesDesc.html" />
   </documentation>
   <types>
        <type name="Employee" schemaTypeRef="tns:Employee" binding="tns:ServiceSoap"
                        xmlns:tns="http:// www. myserver.com/tstservice /service1"
                        wsdl="http:// www. myserver.com/tstservice /service1?wsdl"
                        defaultServiceUrl="http:// www. myserver.com/tstservice /service1>
                <documentation>
                        <document type="xml"
                                url="http:// www. myserver.com/tstservice /EmployeeDesc.xml" />
                        <document type="html"
                                url="http:// www. myserver.com/tstservice /EmployeeDesc.html" />
                </documentation>
                <properties>
                        <property name=”id” schemaName=”tns:id” access=”readonly” />
                        <property name=”prop1” schemaName=”tns:prop1” access =”both” />
                        <property name=”prop2” schemaName=”tns:prop2” access =”both” />
                </properties>
                <method name=”method1” operation="method1">
                        <inputMessage>
                                <parameter name=”id”
                                        valueSourceType="property" valueSourceId="id" />
                                <parameter  name=”para1”
                                        valueSourceType="input" valueSourceId = "para1" />
                                <parameter name=”para2”
                                        valueSourceType="Input" valueSourceId = "para2" />
                        </inputMessage>
                </method>
                <method name=”method2” operation="method2">
                        <inputMessage>
                                <parameter name=”employee”
                                        valueSourceType="special" valueSourceId="this" />
                                <parameter name=”versionNo”
                                        valueSourceType="special" valueSourceId="version" />
                        </inputMessage>
                </method>
        </type>
   </types>
</typeDescription>
```

Shared Types

The shared types to be used by all the web services in the enterprise are defined here. These types are not tied to a particular web service. The actual web service to handle them is dynamically decided by the SOAP headers of the web service reply that returns the instances of these types (We need to add a WS-Types interceptor into the web service to do this). The default service URL is given to be used in case if this header is not present (if the web service does not support WS-Types)

Here you can see that each type has an attribute called “wsdl”. This attribute defines a WSDL document used by the methods of this type. Then “schemaTypeRef” attribute maps an XML type defined in this WSDL document into the type defined here. The binding attribute refers to a binding defined in the same WSDL and this binding must be used to generate the stub class.

Properties

Then we define the properties. Each property has to be mapped into a property in the XML element defined by “schemaTypeRef” attribute of the type. “access” attribute of each property defines if this property should be read only or not.

Methods

Then we define the method in the client side stub class. Each method is mapped into an operation defined in the WSDL. The “inputMessage” element defines how to fill the elements in the input message for that operation. The name attribute of the parameter tag gives the name of the element of the input message as defined in WSDL. Then the “valueSource” attribute defines the source of the value to be filled. This can have one of the three values, “property” “input” and “special”.

- “property” means that this value should be obtained from one of the properties of this stub class. Then the “valueSourceId” attribute represent the property name.
- “input” means that the value should be obtained as an input parameter of the method. Here the “valueSourceId” is the name of the method parameter.
- “special” means predefined set of special values. In this case “valueSourceId” defines the value id. This id has to be “this” or “version”. The value “this” will submit the object itself where as the value “version” will submit the version number received using SOAP header.

SOAP Headers

The actual URL to be used in the methods of the given type and other relevant information to be used in the client objects are returned as a SOAP header. Since this information is placed in SOAP a header, the web service stubs that are already developed will continue working properly. A sample SOAP header is given here.

```
<td:typeDescription xmlns:td="http://sample.org/ws-types">
        <td:object name="employee">
                <td:serviceURL>
                        http://www.sampleserver.com/testservice/service1
                </td:serviceURL>
                <td:version>342</td:version>
                <td:lifetime createdAt="01-01-06 22:00" expiresAt="01-01-06 23:00" />
                <td:singnagure>...</td:signature>
        </td:type>
<td:typeDescription>
```

The header gives the details of the return object. The actual web service URL to be used in the methods, the version number of the returned object (this is useful in order to provide a static snapshot of dynamically changing data), the created time and the expiry time of the object and the signature (used to validate the object creator, specially useful when the client who receives the object passes that to a different web service). All these elements are optional. If the “serviceURL” is omitted, the default service URL mentioned in the WSSTD document will be used to execute the method calls.

When these objects are passed as the method parameters, the same header with the entries for each object has to be included in the request SOAP message as well.