Requirements

- Optional fields in messages
- Possible extensions to messages
- Version compatibility

Example description

As an example we take the srmPrepareToPut method.

Two different WSDL versions are presented.

The first one - srm-3.0.example.wsdl - is an exact representation of the current specification.

The second - srm-3.1.example.wsdl - presents some extensions, added either as optional values or explicit extension elements.

The relevant method is composed by:

Input Message

```
String              userID,
String              authorizationID,
String              transferProtocols[],
String              userRequestDescription,
Boolean             streaming,
EnumOverwriteMode   overwriteOption,
PutFileRequest {
       String               toSURL,
       int                  desiredLifetime,
       EnumFileStorageType  toFileStorageType,
       String               toStorageSystemInfo,
       Int                  knownSizeOfFile,
       String               spaceToken
}[]
Int                  desiredLifetime,
EnumFileStorageType  toFileStorageType,
String               toStorageSystemInfo,
String               spaceToken
```

Output Message

```
String requestToken,
ReturnStatusForRequest  {
       EnumStatusCode  statusCode,
       String          explanation
}
```

Document/Literal encoding and programming styles (RPC/Wrapped)

Using Document/Literal SOAP encoding a call to a service is simply a message exchange, that can be one or two-way. The message is defined in a schema and the toolkits will generate an equivalent object in their native language.

So for the srmPrepareToPut call taken as an example, Axis will generate a SrmPrepareToPutRequest and an SrmPrepareToPutRequest, that leave the code for the call as in:

```
public gov.lbl.sdm.srm3_0.impl.SrmPrepareToPutResponse srmPrepareToPut(gov.lbl.sdm.srm3_0.
   throws java.rmi.RemoteException {

   ReturnStatusForRequest requestStatus = new ReturnStatusForRequest();
   requestStatus.setStatusCode(gov.lbl.sdm.srm3_0.EnumStatusCode.fromString("SRM_SUCCESS")
   requestStatus.setExplanation("sample-explanation");

   gov.lbl.sdm.srm3_0.impl.SrmPrepareToPutResponse resp = new gov.lbl.sdm.srm3_0.impl.SrmP
   resp.setRequestToken("sample-token");
   resp.setRequestStatus(requestStatus);

   return resp;
}
```

An alternative to the object BLOB approach is 'wrapping' and 'unwrapping' the message into multiple fields, making it look like a traditional RPC style call. It leaves the code as in:

```
public void srmPrepareToPut(String userID, String authorizationID, String[] transferProtoc
   boolean streaming, EnumOverwriteMode overwriteOption, PutFileRequest[] request, int des
   EnumFileStorageType toFileStorageType, String toStorageSystemInfo, String spaceToken,
   javax.xml.rpc.holders.StringHolder requestToken,
        gov.lbl.sdm.srm3_0.holders.ReturnStatusForRequestHolder requestStatus) throws java

   requestToken = new javax.xml.rpc.holders.StringHolder();
   requestToken.value = "sample-token";

   ReturnStatusForRequest requestStatusObj = new ReturnStatusForRequest();
   requestStatusObj.setStatusCode(gov.lbl.sdm.srm.EnumStatusCode.fromString("SRM_SUCCESS")
   requestStatusObj.setExplanation("sample-explanation");
   requestStatus = new gov.lbl.sdm.srm3_0.holders.ReturnStatusForRequestHolder();
   requestStatus.value = requestStatusObj;
}
```

In this case it also happens that the return message contains more than one single object, so the approach defined in JAX/RPC is to have 'Holder' objects that can be filled with proper values to be returned to the client.

It is important to mention that 'on-the-wire' the message is identical in both cases. It is simply a different style of processing the request and building the response inside the code.

Optional fields in messages

Defining a field inside a message as optional allows backward compatibility. If an old client comes up to a new server with what will be an incomplete message, the server will still be able to process it, and take action as needed - either returning an error if the request cannot be completed, or most likely filling up the missing fields with default values and processing the request.

Using Document/Literal this can be achieved as it is supported by XML Schema.

Example taken from srm-3.1.example.wsdl:

```
<xsd:complexType name="PutFileRequest">
  <xsd:sequence>
    <xsd:element name="toSURL" type="xsd:string"/>
    <xsd:element name="desiredLifetime" type="xsd:int"/>
    <xsd:element name="toFileStorageType" type="srm:EnumFileStorageType"/>
    <xsd:element name="toStorageSystemInfo" type="xsd:string"/>
```

```
      <xsd:element name="knownSizeOfFile" type="xsd:int"/>
      <xsd:element name="spaceToken" type="xsd:string"/>
      <xsd:element name="newFieldInObject" type="xsd:string" minOccurs="0"/>
    </xsd:sequence>
  </xsd:complexType>
```

So we added a new field to the PutFileRequest object, but made sure it was optional, so that the server can still process messages coming from clients which do not fill it up.

Any of the following SOAP requests will be understood by a server implementing the second WSDL.

```
<?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>
         <srmPrepareToPutElem xmlns="http://sdm.lbl.gov/srm-wg/impl/srm-3.1">
            <userID xmlns="">user-id</userID>
            <authorizationID xmlns="">authz-id</authorizationID>
            <transferProtocols xmlns="">protocol-1</transferProtocols>
            <transferProtocols xmlns="">protocol-2</transferProtocols>
            <userRequestDescription xmlns="">user-request-description</userRequestDescript
            <streaming xmlns="">false</streaming>
            <overwriteOption xmlns="">NEVER</overwriteOption>
            <request xmlns="">
               <toSURL>to-surl-1</toSURL>
               <desiredLifetime>100000</desiredLifetime>
               <toFileStorageType>PERMANENT</toFileStorageType>
               <toStorageSystemInfo>to-storage-system-info-1</toStorageSystemInfo>
               <knownSizeOfFile>1000</knownSizeOfFile>
               <spaceToken>space-token-1</spaceToken>
            </request>
            <request xmlns="">
               <toSURL>to-surl-2</toSURL>
               <desiredLifetime>200000</desiredLifetime>
               <toFileStorageType>PERMANENT</toFileStorageType>
               <toStorageSystemInfo>to-storage-system-info-2</toStorageSystemInfo>
               <knownSizeOfFile>2000</knownSizeOfFile>
               <spaceToken>space-token-2</spaceToken>
            </request>
            <desiredLifetime xmlns="">1000</desiredLifetime>
            <toFileStorageType xmlns="">PERMANENT</toFileStorageType>
            <toStorageSystemInfo xmlns="">to-storage-system-info</toStorageSystemInfo>
            <spaceToken xmlns="">space-token</spaceToken>
         </srmPrepareToPutElem>
      </soapenv:Body>
   </soapenv:Envelope>
```

And the new one with an extra field for PutFileRequest:

```
   <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>
         <srmPrepareToPutElem xmlns="http://sdm.lbl.gov/srm-wg/impl/srm-3.1">
            <userID xmlns="">user-id</userID>
            <authorizationID xmlns="">authz-id</authorizationID>
            <transferProtocols xmlns="">protocol-1</transferProtocols>
            <transferProtocols xmlns="">protocol-2</transferProtocols>
            <userRequestDescription xmlns="">user-request-description</userRequestDescript
            <streaming xmlns="">false</streaming>
            <overwriteOption xmlns="">NEVER</overwriteOption>
            <request xmlns="">
               <toSURL>to-surl-1</toSURL>
               <desiredLifetime>100000</desiredLifetime>
```

```
               <toFileStorageType>PERMANENT</toFileStorageType>
               <toStorageSystemInfo>to-storage-system-info-1</toStorageSystemInfo>
               <knownSizeOfFile>1000</knownSizeOfFile>
               <spaceToken>space-token-1</spaceToken>
               <newFieldInObject>new-field-in-object</newFieldInObject>
            </request>
            <request xmlns="">
               <toSURL>to-surl-2</toSURL>
               <desiredLifetime>200000</desiredLifetime>
               <toFileStorageType>PERMANENT</toFileStorageType>
               <toStorageSystemInfo>to-storage-system-info-2</toStorageSystemInfo>
               <knownSizeOfFile>2000</knownSizeOfFile>
               <spaceToken>space-token-2</spaceToken>
               <newFieldInObject>new-field-in-object</newFieldInObject>
            </request>
            <desiredLifetime xmlns="">1000</desiredLifetime>
            <toFileStorageType xmlns="">PERMANENT</toFileStorageType>
            <toStorageSystemInfo xmlns="">to-storage-system-info</toStorageSystemInfo>
            <spaceToken xmlns="">space-token</spaceToken>
         </srmPrepareToPutElem>
      </soapenv:Body>
   </soapenv:Envelope>
```

Possible extensions to messages

It might happen that a message is defined for a fixed set of fields, but leaves space for extra fields that are up to the client and server to defined and support.

Using XML Schema to define the message exchanges between client and server this can be achieved by having a special ExtensionType defined, which can take any number of any kind of object.

As an example, we extend the smPrepareToPutRequest object (the request message definition) to allow additional fields in the end of the message. The schema definition looks like this:

```
<xsd:complexType name="ExtensionType">
  <xsd:sequence>
    <xsd:any processContents="lax" minOccurs="1" maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="srmPrepareToPutRequest">
  <xsd:sequence>
    <xsd:element name="userID" type="xsd:string"/>
    <xsd:element name="authorizationID" type="xsd:string"/>
    <xsd:element name="transferProtocols" type="xsd:string" maxOccurs="unbounded"/>
    <xsd:element name="userRequestDescription" type="xsd:string"/>
    <xsd:element name="streaming" type="xsd:boolean"/>
    <xsd:element name="overwriteOption" type="srm:EnumOverwriteMode"/>
    <xsd:element name="request" type="impl:PutFileRequest" maxOccurs="unbounded"/>
    <xsd:element name="desiredLifetime" type="xsd:int"/>
    <xsd:element name="toFileStorageType" type="srm:EnumFileStorageType"/>
    <xsd:element name="toStorageSystemInfo" type="xsd:string"/>
    <xsd:element name="spaceToken" type="xsd:string"/>
    <xsd:element name="newFieldInRequest" type="xsd:string" minOccurs="0"/>
    <xsd:element name="extension" type="srm:ExtensionType" minOccurs="0" maxOccurs="1"/>
  </xsd:sequence>
</xsd:complexType>
```

And an example request that uses this new extension field can be:

```
<?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>
   <srmPrepareToPutElem xmlns="http://sdm.lbl.gov/srm-wg/impl/srm-3.1">
      <userID xmlns="">user-id</userID>
      <authorizationID xmlns="">authz-id</authorizationID>
      <transferProtocols xmlns="">protocol-1</transferProtocols>
      <transferProtocols xmlns="">protocol-2</transferProtocols>
      <userRequestDescription xmlns="">user-request-description</userRequestDescript
      <streaming xmlns="">false</streaming>
      <overwriteOption xmlns="">NEVER</overwriteOption>
      <request xmlns="">
         <toSURL>to-surl-1</toSURL>
         <desiredLifetime>100000</desiredLifetime>
         <toFileStorageType>PERMANENT</toFileStorageType>
         <toStorageSystemInfo>to-storage-system-info-1</toStorageSystemInfo>
         <knownSizeOfFile>1000</knownSizeOfFile>
         <spaceToken>space-token-1</spaceToken>
      </request>
      <request xmlns="">
         <toSURL>to-surl-2</toSURL>
         <desiredLifetime>200000</desiredLifetime>
         <toFileStorageType>PERMANENT</toFileStorageType>
         <toStorageSystemInfo>to-storage-system-info-2</toStorageSystemInfo>
         <knownSizeOfFile>2000</knownSizeOfFile>
         <spaceToken>space-token-2</spaceToken>
      </request>
      <desiredLifetime xmlns="">1000</desiredLifetime>
      <toFileStorageType xmlns="">PERMANENT</toFileStorageType>
      <toStorageSystemInfo xmlns="">to-storage-system-info</toStorageSystemInfo>
      <spaceToken xmlns="">space-token</spaceToken>
      <extension>
         <myExtensionField1>extension-field-1-value</myExtensionField1>
         <myExtensionField2>extension-field-2-value</myExtensionField2>
      </extension>
   </srmPrepareToPutElem>
</soapenv:Body>
</soapenv:Envelope>
```

The biggest issue with this approach is that by not explicitly defining these fields in the message schema, the toolkits will not be able to properly generate holders for them inside the stubs. They will be kept in generic holders - of 'Object' classes in java - and things like casting to proper objects and even definition of these objects has to be done manually.

Another options is to treat this values directly in XML, by using a SAX or DOM parser, avoiding the need of manually language binding for these extra objects.

Version compatibility

By using document/literal style encoding of SOAP messages a client passes a XML blob, which needs to be defined in a schema and properly namespaced. The toolkits then take this description and build language specific objects that turn the task of processing the messages into something easy.

The major problem is that the obvious solution of putting the version inside the namespace - major and minor - to allow proper extension of its definition will break compatibility between versions.

Taking the given WSDLs as example, in the first one the messages passed related to services calls are defined in the

```
http://sdm.lbl.gov/srm-wg/impl/srm-3.0
```

namespace. On the other hand, in the second WSDL the namespace was changed to

```
http://sdm.lbl.gov/srm-wg/impl/srm-3.1
```

.

Messages coming from a client generated from the first WSDL will give the server:

```
...
<srmPrepareToPutElem xmlns="http://sdm.lbl.gov/srm-wg/impl/srm-3.0">
...
```

causing it to fail, as it was expecting and understands only:

```
...
<srmPrepareToPutElem xmlns="http://sdm.lbl.gov/srm-wg/impl/srm-3.1">
...
```

A solution to the problem can be to keep the namespace generic and version agnostic, something like

```
http://sdm.lbl.gov/srm-wg/impl/srm
```

, and leave the versioning to the files themselves. Another option is to keep only the major version as part of the namespace, assuming no compatibility between clients with different major versions.

How to run the example application

First step is download this file. After extracting its contents, go inside the

```
srm-examples
```

directory and edit the build.xml file, replacing the properties:

```
<property name="tomcat.instdir" value="/home/rbritoda/work/glite-repository/tomcat/5.0.
<property name="tomcat.docbase.srm3.0" value="/home/rbrito/work/tests/srm-examples/auto
<property name="tomcat.docbase.srm3.1" value="/home/rbritoda/work/tests/srm-examples/au
```

with proper values.

- tomcat.insdir is the path to the ROOT dir of your tomcat server.
- tomcat.docbase.srm3.0 here you should replace the path up to srm-examples (excluding) with the location where you extract the contents of the file.
- tomcat.docbase.srm3.1 same as above

Then from inside the directory, run:

ant clean; ant webapp; ant tomcat.configure; ant compiletest

This will compile all classes and create a local tomcat instance. Next step is to start this tomcat instance:

./autogen/tomcat/bin/catalina.sh start. You might need to stop it first.

And as a final step you may want to try the unit tests provided, by doing **ant functest**.

-- RicardoRocha - 12 Oct 2005

This topic: SRMDev > XmlSolution
Topic revision: r2 - 2005-10-12 - RicardoRocha