Understanding web service interoperability

Issues in integrating multiple vendor web services implementations

Frank Cohen

February 01, 2002

Web services technology has the promise to provide a new level of interoperability between software applications. It should be no wonder then that there is a rush by platform providers, software developers, and utility providers to enable their software with SOAP, WSDL, and UDDI capabilities. In this article, Frank Cohen describes inconsistencies in web service implementations and how to evaluate SOAP technology to deliver better software, faster.

O brave new world

Stepping onto the new web services island one might think "Oh brave new world that has such wonders." Then the reality of web services sets-in. Dozens of platform providers, independent software vendors, and utility software developers have implemented web services protocols (SOAP, WSDL, and UDDI) in their products. While the protocols are now nearly two years old and working drafts of the next specifications (SOAP 1.2 for example) are in the works, developers have had to interpreted the meaning in parts of the specifications. Interpretation allows interoperability problems to seep into SOAP-based web services.

Web service interoperability goals are to provide seamless and automatic connections from one software application to another. SOAP, WSDL, and UDDI protocols define a self-describing way to discover and call a method in a software application -- regardless of location or platform. Data is marshaled into XML request and response documents and moved between software packages using HTTP or message-based protocols. Interoperability problems creep in at the discovery, definition, and request/response mechanisms.

Discovery

In the dreamy world of web service utopia every software application is coded with a self-discovery and self-categorization method. Software that lacks a needed function looks into a UDDI-based registry of services and automatically makes a contract with a found web service to handle the task. WSDL and SOAP enable communication once a web service function is found. The problem then is to categorize what the function does so it may be found. UDDI defines TModels that are taxonomies to describe the location, path and character of the function.
UDDI enables businesses to host on-line registries of available web services. On the public Internet Microsoft, HP and IBM have begun offering UDDI registries to businesses. A business uses the UDDI TModel system to categorize the hosted web service. And therein lies the problem: UDDI allows multiple taxonomies and expects self-policing for mistaken entries in the registry. For example, suppose a web service that prints and sends invoices lists itself in a UDDI registry using an SIC code but does not list geographic information. Using such a web service from the other side of the planet would work. However, it may easier to lick the stamp yourself.

In time UDDI will be well-used and understood by the traditional taxonomy providers, including LCSH (Library of Congress Subject Heading), FAST (Faceted LCSH), DDC (Dewey Decimal Classification), and LCC (Library of Congress Classification.) Until the taxonomy experts add their practical knowledge of developing and maintaining directory structures in UDDI, plan for interoperability problems.

Definition

Web services uses WSDL to define how to make a request to a SOAP-based method. WSDL assumes cooperation from companies that define custom data types. Such cooperation is put to the test by collaborative organizations that are establishing interoperability test suites. For example, SOAPBuilders is an open group of SOAP developers that define interoperability test suites to check SOAP custom data types for compatibility. The test suites emerging today begin with the WSDL definition of a SOAP interface. They test the contents of the request and response documents for valid data.

This has put renewed energy behind WSDL efforts. New technologies such as Cape Clear CapeStudio and BEA Cajun automatically develop WSDL documents for SOAP-based web services. Tools such as these eliminate poorly constructed WSDL that appears when developers hand-code WSDL documents.

WSDL documents have been known to cause interoperability problems. For example, consider the above snipped of WSDL for a software test web service. The WSDL defines how to send a testRequest command. However, the <definitions> element fails to define the name space. The correct <definitions> element should look as shown in Listing 2.
The developer likely thought the web service would default to the standard W3 SOAP name space. While this may work for primitive data types like String, there are known interoperability problems with Date data types that will appear later in this article. Without specifying the namespace, the web service will likely fail to handle the data type correctly.

**Request/response**

SOAP defines a standard way for software applications to call each other’s methods and to pass data. SOAP requests are XML documents containing a description of the namespace, method called and data. XML tries to be fairly flexible for developers to write XML elements and definitions. The flexibility can be a problem for SOAP interoperability.

For example, here's what a typical SOAP response document might look like:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <SOAP-ENV:Body>
    <ns1:echoStringResponse xmlns:ns1="http://soapinterop.org/">
      <result xsi:type="xsd:string">Hello!</result>
    </ns1:echoStringResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

This response document is sending back a String containing the text "Hello!" The `<result>` element also includes the `xsi:type="xsd:string"` parameter which gets deserialized into a Java String object. Many SOAP tools add explicit typing information into the request and response document. On the other hand, some SOAP libraries return response with no type information.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <SOAP-ENV:Body>
    <ns1:echoStringResponse xmlns:ns1="http://soapinterop.org/">
      <result>Hello, I'm a string!</result>
    </ns1:echoStringResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The `<result>` element in the above response document includes no type information. In this case the SOAP library that deserializes the `<result>` value must look into the WSDL description of the service to find the description of the return type. If the WSDL does not define a response type then it is anyone’s guess what type of object will be returned.

**Data types**

Data types are where the rubber meets the road in web services. SOAP uses serializer and deserializer objects to translate from the native language of a software application to the SOAP protocols that move the request over the wire. It is here where native languages introduce dependencies on the data. For example, the way Java defines date objects is different than
Microsoft .NET C++ date objects. This has the unfortunate effect of allowing SOAP data types with the same name to have different implementations. Interoperability problems lie ahead.

The most common data types to fail interoperability tests are floating point numbers and dates.

**Floating-point and decimal data types**

Floating-point numbers in SOAP are represented as strings of decimal digits. The SOAP definition for floating point numbers also allows the possibility to handle exponential numbers using notation in use by engineers for decades. In general this works as would be expected. However, when pushed floating-point numbers have problems.

For example, the original IBM SOAP4J implementation (now the Apache SOAP and Apache AXIS libraries) used the Java toString method and constructor to convert floating-point values found in SOAP documents into Java objects. Numbers appear as a series of decimal numbers. When it came to serializing the floating point number "infinity" Java outputs the string as "Infinity". On the other hand, XML Schema serializes infinity as "INF." This caused SOAP4J to have interoperability problems with other SOAP toolkits.

Just as the Internet was born from the cooperation of network administrators, today we see SOAP implementers cooperating to solve interoperability problems. Apache SOAP, the successor to SOAP4J, was changed to accept "INF" as a valid way to serialize infinity.

Decimal data types suffer from language dependencies when pushed too. Decimal data types may represent large numbers up to 40 digits of precision. Relying on all 40 digits in a SOAP request or response is problematic unless both server and client are implemented on the same language. This is also true for fractional seconds with dates and trailing zeros on decimals.

BigDecimal is a good example of interoperability pitfalls introduced by language dependencies. Decimal numbers are a necessary part of financial calculations in banking applications where huge numbers are required. The XML Schema specification for decimal data types allows an arbitrary precision. Decimal data types could represent 1000 digits of precision -- that is a decimal number represented as a string of 1000 one-digit numbers. Apache SOAP is based on the Java implementation of the BigDecimal data type. Java's BigDecimal has an upper limit to precision of a number depending on the underlying operating system (Solaris, Windows, etc.)

XML Schema responded to these kinds of interoperability problems by defining a minimally conforming implementation specification. In the case of decimal data types, XML Schema requires at least 18 digits of precision. Apache and Java meet the requirement. But that does not mean a SOAP-based web service will receive the minimum precision.

Microsoft .NET implementations handle BigDecimal data types up to 29 digits of precision. So what happens to the extra digits of precision when an Apache SOAP request with a BigDecimal data type receives a .NET response? Unfortunately it rests with the local SOAP serializer and deserializer implementation to know. And there is the rub: The SOAP transaction is valid but the data is wrong. Developers need to be smart to code data tests and protections into their software applications to consider invalid data from a SOAP exchange.
Date data types

Date data types suffer from interoperability problems to a greater extent that floating point decimal data types. The XML Schema defines the dateTime data type to contain centuries, years, months, days, hours, minutes, and seconds. But what about milliseconds, microseconds and even smaller measurements of time? We live in a world where 1.1 GHz Intel CPUs sell for less than $150 and users require web services to perform in 2-3 seconds at most. In just about every measurement of web service performance milliseconds are going to count.

XML Schema specifies that any number of digits after seconds may be coded into a dateTime data type but there is no minimum number of digits that an application must support. Apache SOAP uses the Java Date class (java.util.Date) to serialize and deserialize dateTime data types. Java Date supports precision up to the nearest millisecond. .NET's Date data type uses sub-second values up to 4 digits of precision -- so a nanosecond may be represented in a .NET Date data type.

On the horizon

Web services today are provided by the core UDDI, WSDL, and SOAP protocols. On the immediate horizon are a second layer of protocols that define workflow automation, web service management services, and vertical market protocols. Web services greatly help developers build highly integrated solutions. So it should be no surprise to see interoperability problems arrive when workflow automation web services are mixed with vertical market web services.

If web service toolkits are continually improved to solve interoperability problems then customers, users and businesses will more efficiently solve system integration problems than using the existing standards (CORBA, DCOM, and RMI). The more serious web service toolkits vendors have been diligent at solving interoperability problems. If interoperability problems linger or get worse then we are in for slower adoption and much bigger professional services costs to implement integrated systems.

In the meantime, web services is off to a great start with almost universal adoption. The real interoperability challenge may be in areas where SOAP and non-SOAP methods need to communicate, such as CORBA over IIOP.
Related topics

- **Load**, a free open-source utility for testing SOAP-based web services for functionality, scalability, and performance.
- **Web Services Definition Language** (WSDL) specification.
- **SOAP protocol specification**.

© Copyright IBM Corporation 2002

Trademarks
(www.ibm.com/developerworks.ibm/trademarks/)