

On Expressing Web Services Constraints & Capabilities

Jong Lee, BEA Systems <jong.lee@bea.com>

David Orchard, BEA Systems <david.orchard@bea.com>

One useful view of Web services is as a “protocol construction toolkit”; that is, it is an effort to provide software developers with an easy- and ready-to-use stack of common functions that can be composed at will, depending on the task at hand.

As more of these functions have been defined, it has become necessary to describe when and how they are used in a particular instance of a protocol. For example, it is often desirable to communicate the need to encrypt a message to its sender, along with the details of how it should be encrypted.

Until now, this information has been conveyed in an ad hoc fashion. Although WSDL provides an extensibility model, it places a considerable burden upon the designers and users of such extensions, and it requires that such extensions be supported by the WSDL processor of any interested application. Moreover, expressing relationships between extensions (e.g., “you must do A and B, or C”) is extremely difficult without defining new “umbrella” extensions that cover them all.

A framework that allows for the expression of domain-specific configuration information and allowable combinations across domains would address these shortcomings.

Furthermore, it would afford the opportunity to separate the expression of policy from its attachment to specific Web services instances, thereby allowing enough flexibility to attach policy through UDDI and other mechanisms.

BEA believes this to be the task at hand when we talk about “Web Services Constraints and Capabilities,” otherwise referred to as “Web Services Policy.” We informally refer to it the “Phone Call Avoidance Protocol,” because -- if we are successful -- consumers and providers will be able to skip a large amount of the ad hoc configuration required to use Web services today.

This position paper relates BEA’s thinking with regard to the issues in this area. Specifically, we enumerate common scenarios that such a framework should address, discuss the appropriate scope for a solution, and highlight one candidate, WS-Policy.

Motivating and Illuminating a Framework Solution

To understand what a Policy framework should be capable of, we need to understand what is required of it. This section relates the goals that BEA has in this space. In doing so, we identify a number of common *tasks* that a solution must enable.

Consumer Configuration Task

The most basic task for a Policy framework is the configuration of a Web services consumer, so that it can successfully communicate with a given Web service provider. This includes the various parameters that affect the messages sent and received, as well as those that pertain to the larger interaction between the parties involved (e.g., session establishment and maintenance). For example,

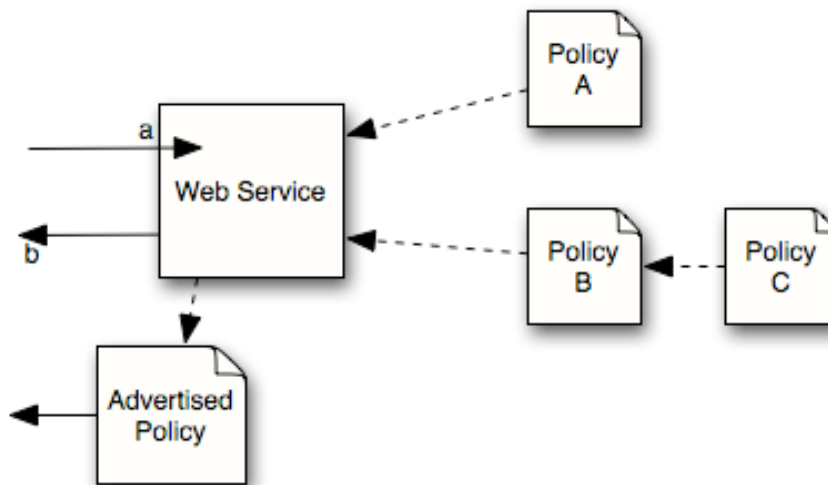


Here, we are concerned with configuring the behaviour of the node marked “Consumer” when it sends message a, receives message b, and when it establishes any parameters that affect any grouping of those messages. This could include the encryption of either or both messages, including one or more SOAP headers in them, affecting their serialization and encoding on the wire, or session-level parameters such as reliability that keep state between the messages.

This task implies that the service provider is able to communicate its various requirements, capabilities and preferences to the consumer, which will be able to evaluate them and choose behaviours that are compatible with its own capabilities, requirements and preferences. Thus, a framework needs to allow both parties to express the *alternatives* acceptable to them, in a clear and unambiguous fashion. In particular, the model of such a framework should allow the alternatives to be normalized, so that each alternative is a whole unit that can be selected completely, rather than containing further qualifications.

Provider Configuration Task

Similarly, the provider of a Web service needs to be configured. However, we believe that this task will be fundamentally different from consumer configuration for the foreseeable future. Rather than allowing the other party (in this case, the consumer) to configure it, provider configuration is characterized by allowing the configuration to be expressed in an abstract, implementation- and service-independent way by the same party.



Here, we see a number of policies (A, B, and C) being used to collectively configure the behaviour of a Web service, which then makes a policy based upon them available to consumers.

This is attractive because it will enable Web services to be managed more easily and flexibly. For example, it would allow a business to set an enterprise-wide minimal policy, to be applied to individual services as they're deployed, no matter what particular platform they're deployed upon. It would also allow a market for Web services management and configuration tools to develop.

This task implies that policies can be modularized and later composed to form sensible service-specific configurations. It also requires issues like versioning, extensibility and inheritance to be available in the framework. Just as there are "from WSDL" and "from code" scenarios for creating Web services, we see corresponding "from Policy" and "from code" scenarios for configuring them.

Message Validation Task

An important facet of the both configuration tasks is the validation of incoming and outgoing messages. This implies a certain tension with them; where they focused on defining acceptable behaviours that different nodes can evidence, this task requires that acceptable message forms, as measured at a very specific place, be defined. As a result, the framework, as well as individual assertions, will need to be able to describe not only the desired behaviours available, but also how to validate their execution.

Service Query and Selection Tasks

Finally, there may be scenarios in the future where policy information about a set of services can be used to select the most appropriate partner for a particular interaction.

Such a decision may be based upon straightforward selection of the most compatibly configured service (e.g., "I support UTF-8 and so does service Y") or it may be more qualitative (e.g., "Service X has a more desirable privacy policy"). Although we expect these tasks, when performed, to be based upon a policy framework, we do not anticipate it being practical to automate them in the near future.

Too Big, Too Small and Just Right: Scoping the Solution

Examining the Call for Participation to this workshop makes it quickly clear that this area of activity is neither new nor small. Indeed, if one considers it as being primarily concerned with 'metadata,' it becomes quite broad.

Although an attempt at defining a Web-wide (or even broader) framework for configuration and metadata is a tempting technical challenge, we think it likely that such an effort will be prone to failure due to that breadth. If it does succeed, it will likely be so generic as to be inadequate to the requirements that specific use cases place upon it. Furthermore, we don't believe it's possible to deliver such a broad solution in a timely manner. In our judgment, a one-size-fits-all solution will neither be adequate to the requirements of Web services, nor do justice to those of the Web at large. Therefore, BEA believes that the focus of this work should be providing a Web services-specific solution.

Web services is the realization of a set of architectural constraints, some of which are described in the Web Services Architecture document. Furthermore, it is embodied by a set of existing standards: SOAP for messaging, WSDL for description, XML Schema for syntactic description, and UDDI for service registries. Clearly, Web services technologies are evolving and improving over time, but the problem domain is fairly constrained in terms of existing technologies and currently deployed software. Therefore, any solution must embrace today's Web service technologies.

We also believe that the use cases for Policy within Web services need to be constrained; although there are many interesting potential uses for such a framework, BEA believes that we should focus on problems that are encountered and understood today. This means that, for the foreseeable future, standardization work in this area should concentrate on configuration tasks, avoiding attempts to address more complex and open-ended scenarios such as business-level policies (e.g., "open between 9am and 6pm,") and interdependent, multi-stage policies. Note that these scenarios could still build upon such a framework.

Furthermore, we understand that there are existing frameworks that could be bent to these tasks. In selecting a candidate for development, we ask that the amount of change and development required to fit it to the specific requirements at hand be considered. Although there are many different paths that could be taken, we believe there are relatively few that will lead to a successful, timely and well-scoped solution.

That said, it is important to consider and incorporate existing policy vocabularies, so that there is not meaningless duplication and unnecessary misalignment. In particular, existing HTTP and SOAP headers, as well as other forms of metadata, should be examined to assure that their semantics are aligned with those of the appropriate Policy assertions.

For example, HTTP content negotiation headers, such as Accept, Accept-Language and so forth, can be easily thought of as statements of policy. Furthermore, many emerging Web services standards related to reliability, addressing and other functions contain SOAP headers that can also be seen as policy statements.

In choosing a technology, particular care should be taken to make sure that it is easy and intuitive for both authors of domain-specific vocabularies (e.g., security, reliability) and for end users. We believe that one of the biggest challenges that this effort will face is the balance of usability and simplicity against functionality. Although the primary mechanism for achieving this should be use case-driven design, we ask that when there is doubt, a preference be given to simplicity.

One of our greatest concerns is that diverging approaches to this problem are advocated. While we are hopeful that they can be rationalized, BEA has reservations about recommending a partial solution, such as "Properties and Features" in WSDL 2.0, if it were to start an effort to standardize a separate, more complete solution.

Finally, we ask that decisions about what particular functionality such a framework should provide (as opposed to that which is left to individual domains of interest) be weighed carefully. Although such a framework can provide considerable utility, the simplicity and usability of the solution, as well as the applicability of the features to all use cases, rather than specific domains of interest, needs to be considered.

One Candidate: WS-Policy

With this guidance in mind, BEA, along with our partners, has been working on a candidate solution, called WS-Policy

<<http://dev2dev.bea.com/technologies/webservices/standards.jsp>>. We refer readers to that set of specifications for more information.

The sample Workshop scenario could be addressed with the following policy annotation on the service's WSDL description. Note that while we believe the Policy framework and attachment mechanism to be capable of satisfying this scenario, the individual assertions are not as mature.

```
01 <wsdl:definitions name="StockQuote"
    xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/08/policy"
    xmlns:p="http://schemas.xmlsoap.org/ws/2002/12/policy"
    xmlns:wsm="http://schemas.xmlsoap.org/ws/2004/03/rm"
    xmlns:wss="http://schemas.xmlsoap.org/ws/2002/12/secext"
    xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis
200401-wss-wssecurity-utility-1.0.xsd" ... >
02 <wsp:UsingPolicy wsdl:Required="true"/>
03 <wsp:Policy wsu:Id="SRT" >
04 <p:SpecVersion
URI="http://schemas.xmlsoap.org/ws/2002/07/secext"/>
05 <wsp:ExactlyOne>
06 <wsse:SecurityToken>
07 <wsse:TokenType>wsse:X509v3</wsse:TokenType>
</wsse:SecurityToken>
08 <wsse:SecurityToken>
09 <wsse:TokenType>wsse:UsernameToken</wsse:TokenType>
</wsse:SecurityToken>
</wsp:ExactlyOne>
10 <p:SpecVersion URI="http://schemas.xmlsoap.org/ws/2004/03/rm" />
11 <wsm:SequenceCreation/>
12 <wsm:BaseRetransmissionInterval Milliseconds="3000"/>
</wsp:Policy>
13 <wsp:Policy wsu:Id="EncryptSubscribeHeader">
14 <wsse:Confidentiality>
15 <wsse:Algorithm Type="wsse:AlgEncryption"
URI="http://www.w3.org/2001/04/xmlenc#3des-cbc"/>
16 <wsse:KeyInfo><xenc:EncryptedKey/></wsse:KeyInfo>
17 <wsse:MessageParts
Dialect="http://schemas.xmlsoap.org/2002/12/wsse#part" >
p:Header(tns:SubscribeHeader)
</wsse:MessageParts>
</wsse:Confidentiality>
18 <wsse:Integrity>
19 <wsse:Algorithm Type="wsse:AlgCanonicalization"
URI="http://www.w3.org/Signature/Drafts/xml-exc-c14n" />
20 <wsse:Algorithm Type="wsse:AlgSignature"
URI="http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
21 <wsse:MessageParts
Dialect="http://schemas.xmlsoap.org/2002/12/wsse#part" >
22 p:Body() p:Header(tns:SubscribeHeader)
</wsse:MessageParts>
</wsse:Integrity>
</wsp:Policy>
```

```

23 <wsdl:binding name="StockQuoteSoapBinding"
    type="tns:StockQuotePortType" >
24   <wsoap:binding style="document"
    transport="http://schemas.xmlsoap.org/soap/http"/>
25   <wsp:PolicyReference URI="#SRT" />
26   <operation name="GetLastTradePrice" >
27     <wsoap:operation soapAction="http://example.com/TradePrice" />
28     <input>
29       <wsoap:body use="literal" />
30       <wsoap:header message="tns:SubscribeToQuotes"
    part="subscribeheader" use="literal"/>
31     <wsp:PolicyReference URI="#EncryptSubscribeHeader" />
    </input> ... </operation></binding> ... </definitions>

```

Line 2 indicates that the WSDL is extended using WS-Policy; in this particular example, each of the provided bindings uses WS-Policy, and the WSDL should not be processed without understanding WS-Policy, so `@wsdl:Required="true"`.

Line 3 is a policy expression that expresses requirements for security (lines 4-9) and reliability (lines 10-12); this policy is named using `@wsu:Id` for convenient reference from a binding.

Security starts with line 4, which is a generic assertion that indicates support for a specification, identified by URI, is required; in this case, WS-Security is required. Line 5 is a choice between assertions defined in WS-SecurityPolicy that indicate an X.509 security token (lines 6-7) or a username and password (lines 8-9).

Reliability starts with line 10, indicating support for WS-ReliableMessaging is required; the assertion in line 11 indicates that the RM destination is responsible for creating RM sequences, and the assertion in line 12 is a parameterized assertion indicating that the RM source will wait 3 seconds for an acknowledgement before retransmitting messages.

Line 13 lists policy that applies to a specific input message; it too is named using `@wsu:Id`. The assertion in line 14 is defined in WS-SecurityPolicy and indicates encryption using DES (line 15), with a symmetric key protected by a token (lines 5-9), of the Subscribe SOAP header (Line 17). (We assume the Subscribe SOAP header allows for encrypted content.) The assertion in line 18 is also defined in WS-SecurityPolicy and indicates canonicalization (line 19) and signature algorithms (line 20) for signing the SOAP Body and Subscribe header (lines 21-22).

Line 23 is a typical binding to SOAP (line 24). Line 25 attaches the policy in line 3 to this binding, requiring secure and reliable messaging.

Line 26 lists the GetLastTradePrice Message Exchange Pattern. Line 29 defines the SOAP Body for the input message using the existing SOAP binding; line 30 defines a SOAP header block; line 31 attaches the policy in line 13 to this message, requiring the header to be signed and encrypted.