

# WS-PolicyConstraints: A Domain-Independent Web Services Policy Assertion Language

Anne Anderson <anne.anderson@sun.com>  
Sun Microsystems Laboratories

3 November 2005

## Executive Summary

The “guts” of a web services policy is the statements, or “Assertions”, it makes about the acceptable and supported values for various policy parameters. For example, an Assertion may describe a list of acceptable and supported encryption algorithms, or acceptable and supported privacy guarantees.

Currently, there is no generic language for expressing Assertions. The impact is that every policy processor must install a new code module to handle the language used for each new or modified Assertion that the processor will support. As more and more Assertions are defined, this model becomes unworkable: there is a very low probability that every policy processor will contain a code module, in the proper revision, for each of the Assertions it must support.

*WS-PolicyConstraints* is a proposed language for expressing Assertions that solves this problem. The policy parameters and their values are defined by the groups that need the policies, but the language for describing the sets of acceptable and supported values for these parameters is generic. With *WS-PolicyConstraints*, a single code module can support every policy written using this generic language, regardless of where or how its policy parameters are defined. Policy parameters can be augmented or revised without requiring changes to the policy processor. This will vastly improve policy interoperability and greatly reduce development and maintenance costs.

*WS-PolicyConstraints* does not compete with *WS-Policy*, *WS-SecurityPolicy*, or *WS-ReliableMessaging Policy*. It is layered on top of *WS-Policy* just like *WS-SecurityPolicy* and *WS-ReliableMessaging Policy*, and actually enhances the value of *WS-Policy*. A code module to support policies written with *WS-PolicyConstraints* can co-exist with modules supporting other types of Assertions such as those in *WS-SecurityPolicy* and *WS-ReliableMessaging Policy*.

## Introduction

In order to interact, web services need more information than their *WSDL (Web Services Definition Language)* [1] description provides. For example, they may need to know what confidentiality mechanisms each side supports and requires, what identity, role, and authentication attributes each side supports and requires, or what privacy guarantees each side supports and requires. They may also need to know application-specific requirements not captured in the *WSDL*, such as shipment time guarantees or price. Requirements about this type of information have come to be referred to as the web service's “policy”, since this information is related to deployment choices rather than to the business logic and interfaces of the service. The pieces of policy information, such as “encryption algorithm” or “shipment time guarantee”, are called the policy “vocabulary”. Policy vocabularies are defined by standards groups, industry groups, vendors, or application developers who are concerned with some

area that requires policy. These policy areas, such as “privacy”, “secure messaging”, “reliable messaging”, or a specific application area, are called “domains”.

A web services policy is the combinations of policy vocabulary values that the service requires or supports. We say “combinations” because the value for one vocabulary item may depend on the values of other vocabulary items. For example, a service may require two-factor authentication for messages coming from outside its firewall, but may accept password authentication for messages coming from inside, so the value for authentication type depends on the value for source IP address.

## Web Services Policy Languages

There is currently no standard language for expressing web services policies. The best-known proposal is the *Web Services Policy Framework (WS-Policy)* [2] proposed by Microsoft, IBM, BEA, and others, but this proposal has not yet been submitted to any standards body. Lesser-known proposals include a proposal from Oracle to add “compositors” (Boolean operators) [3] to the *WSDL 2.0* “Features and Properties” element, and the *Web Services Policy Language (WSPL)* [4][5] developed within the *OASIS eXtensible Access Control Markup Language (XACML)* Technical Committee (but not progressed to standard status). All of these proposals have the same basic structure: a “policy” is a Boolean combination of predicates, or “Assertions”, that specify acceptable values for one or more vocabulary items. Several domain-specific policy specifications, including *WS-SecurityPolicy* [6] and *WS-ReliableMessaging Policy* [7], assume such a policy structure. Other types of policy languages based on semantic web concepts have been proposed, but so far have not been considered mature enough for consideration as an industry standard. So it appears that the Boolean combinations model will be the one used for web services policies.

Using the Boolean combinations model, a web services policy indicating that:

“the requester must be an authorized-vendor and must either present an X509Certificate or must be inside our firewall with a Kerberos token”

might be expressed in the following general form:

```
Policy {  
  AND {  
    "roles must include 'authorized-vendor'",  
    OR {  
      AND {  
        "authentication-tokens must include 'KerberosToken'",  
        "network-domain is '129.156.220.0/255.255.255.0'"  
      },  
      "authentication-tokens must include 'X509Certificate'",  
    },  
  },  
}
```

*WS-Policy* and the *WSDL* “compositors” proposal specify the “Policy”, “AND”, and “OR” parts of this policy (parts in bold), but do not specify how the individual Assertions, such as “authentication-tokens must include 'KerberosToken'”, are expressed. *WS-Policy* requires each policy domain to design and specify its own Assertions. As an example, *WS-SecurityPolicy* is a set of Assertions designed to describe requirements about authentication tokens, encryption algorithms for

confidentiality, signature algorithms for integrity, etc.. Similarly *WS-ReliableMessaging Policy* is a set of Assertions designed to describe requirements related to reliable messaging. Assertions will be needed for all types of policy vocabulary items, including required roles for authorization, network addresses of service consumers and providers, privacy requirements, and vocabulary items related to particular service applications. The list of domains that need policy Assertions is quite large.

### **Problems with Domain-Specific Policy Assertions**

This domain-specific approach to Assertion specification has serious implications. Since there is no generic, domain-independent language for expressing the Assertions, each policy processing engine must be configured with a code module that understands how to process each particular Assertion that the engine is to support. So, for example, to support *WS-SecurityPolicy*, a policy engine must be configured with code modules that understand each of the Assertions defined in the *WS-SecurityPolicy* specification: “UsernameToken”, “EncryptedParts”, “SignedParts”, “AlgorithmSuite”, etc. These modules handle the work of matching one service's *WS-SecurityPolicy* Assertions against another service's *WS-SecurityPolicy* Assertions, or for verifying that a message satisfies the *WS-SecurityPolicy* Assertions. If an industry or standards group specifies a format for a “NetworkAddress” Assertion, a new code module supporting that Assertion must somehow be added to every policy engine that is to deal with network addresses. If new types of authentication tokens are introduced, new Assertions must be designed, and new code modules to process those Assertions must be developed and installed in each policy processor. If a new secure hash algorithm is introduced, the code modules that handle the “AlgorithmSuite” Assertion must be updated on each policy processor. As more and more Assertions are developed, the probability that every policy processor will have the correct versions of all code modules required to handle all policies becomes low. It is also a lot of work to develop and test the Assertion-handling modules, adapt them for different platforms, and deploy them to all the policy processors in the field. Some Assertions might even be proprietary, and not all policy processors would have licenses for the required code modules. If your application depends on a particular Assertion, and the policy processor on one of the servers that must process it does not have the correct code module, your application will not be able to run on that server. This is a serious interoperability concern.

### **The Alternative: Domain-Independent Policy Assertions**

There is a better model for the specification of Assertions. This model has been successfully used for several years in the OASIS Standard *eXtensible Access Control Markup Language (XACML)* [8]. It lets each domain define the policy vocabulary items it needs, either in the form of a schema or in the form of *SAML* [9] or *XACML* “Attributes”. The meaning of these policy vocabulary items must be understood by the service endpoints that must implement the policy. The Assertions, however, use a single generic, domain-independent language to describe the acceptable values for the vocabulary items, and a single standard *XACML* policy processor can evaluate any policy written in *XACML*.

To compare the two models, a domain-specific Assertion saying “the authentication tokens must include an X509Certificate” taken from *WS-SecurityPolicy* looks as follows (showing only the required information):

```
<sp:X509Token/>
```

Using the *XACML* model, this Assertion could be written as:

```

<xacml:Apply FunctionId="anyURI-equal">
  <xacml:AttributeDesignator AttributeId="token-type" DataType="anyURI"/>
  <xacml:AttributeValue DataType="anyURI">sp:x509Token</AttributeValue>
</xacml:Apply>

```

Alternatively, the *XACML* model could describe this Assertion in a way that allows the presence of an X509 token in the message to be verified directly by the policy processor, without having to understand what an “X509Token” is:

```

<xacml:Apply FunctionId="must-be-present">
  <xacml:AttributeValue DataType="xpath-expression">
    //S11:Envelope/S11:Header/wss:Security/ds:KeyInfo/ds:X509Data
  </xacml:AttributeValue>
</xacml:Apply>

```

With the domain-specific model, a code module must be supplied that understands what a `<sp:X509Token>` element means, how it is to be matched against other Assertions, and how it is to be verified against messages to see if they contain an X509 token. With the *XACML* model, any code module that understands the generic functions “anyURI-equal” and/or “must-be-present” and the generic data types “anyURI” and/or “xpath-expression” will be able to match or verify this Assertion. Any Assertion, from any domain, that uses a URI-valued vocabulary item can be handled using the same code module. The endpoints will still need to verify any actual signatures that use this token; signature verification is usually considered to be outside the scope of policy.

This is a very simple example, and there are of course subtleties that are being glossed over here.

### The *WS-PolicyConstraints* Language

To support the domain-independent Assertions model, we have developed an *XACML*-based language called *WS-PolicyConstraints* [10]. The Assertion language itself is domain-independent, but it can be used with any domain-specific policy vocabulary, or even directly with requirements on the contents of messages by using *XPath* [11] expressions. *WS-PolicyConstraints* is a subset of *WSPL*, with the parts of *WSPL* that overlapped and conflicted with *WS-Policy* and *WS-PolicyAttachment* [12] removed.

A number of questions are often asked about *WS-PolicyConstraints* and the domain-independent Assertions model.

#### Question 1: How many generic functions and data types are required?

The list is quite reasonable. *XACML* uses 20 different data types, including “string”, “integer”, “dateTime”, “anyURI”, “Boolean”, etc. *XACML* also defines several hundred standard functions, but only a subset of these are needed for use in web services policies: “equal”, “greater-than-or-equal”, “greater-than”, “less-than”, and “less-than-or-equal” (for each data type where such comparisons make sense) and a few special match functions for data types commonly used in policies, such as X509 Distinguished Names, e-mail addresses, and IP addresses. These have proven sufficient for expressing complex policies in a variety of domains.

#### Question 2: Isn't the Assertion matching process complex and time-consuming?

The answer is an emphatic “No”. The functions selected for use with web services policies are trivial to match. For example, if one policy states that “price must be greater-than-or-equal to '5'” and another policy states that “price must be equal to '20'”, simple integer floor/ceiling checks can be used to determine that “price must be equal to '20'” is the only value satisfying both policies. The remaining function matching operations are just as simple and are described in the *WS-PolicyConstraints* specification.

### **Question 3: What about complex, multi-valued vocabulary items?**

People often ask whether an *XACML*-style language can deal with complex, structured vocabulary items. An example might be a requirement that the AES encryption algorithm be used with a key length of 128, or the 3DES encryption algorithm must be used with a key length of 168; a 3DES key with length 128 would not be acceptable. *WS-PolicyConstraints* can indeed handle such requirements – there is a mechanism for placing multiple requirements on the same element in an XML schema (a schema element for describing all requirements for a single encryption mechanism, for example).

### **Question 4: Are domain-specific Assertions ever required?**

Domain-specific Assertions may be required for some types of policies; there has not been enough policy experience in industry to make a definitive answer. We have found no use cases so far that can not be handled at least as well with *WS-PolicyConstraints* as with domain-specific Assertions.

If domain-specific Assertions are required, however, code modules to support them can co-exist with the code module that supports *WS-PolicyConstraints*. There is no conflict in using both approaches so long as each vocabulary item is used with only one type of Assertion language.

### **Question 5: Does *WS-PolicyConstraints* conflict with *WS-Policy* or with *WS-SecurityPolicy*?**

The answer again is an emphatic “No”. *WS-PolicyConstraints* is strictly a language for expressing Assertions. It is designed for use within a policy framework such as *WS-Policy*. It is a perfect complement to *WS-Policy*: *WS-Policy* handles the “policy” and Boolean combinations layer in a domain-independent way, while *WS-PolicyConstraints* handles the Assertions layer in a domain-independent way. A code module for handling Assertions written using *WS-PolicyConstraints* can co-exist with code modules for handling domain-specific Assertions written using *WS-SecurityPolicy*, *WS-ReliableMessaging Policy*, or other domain-specific languages.

### **Question 6: Has *WS-PolicyConstraints* been implemented?**

The answer is “No”; *WS-PolicyConstraints* itself has not yet been implemented. But, *WS-PolicyConstraints* is a subset of both *XACML* and *WSPL*, both of which have been implemented and used successfully. There is no reason to expect implementation issues that would not have been encountered in *WSPL*.

### **Question 7: What are the plans for standardizing *WS-PolicyConstraints*?**

As a start, Sun and others are requesting a discussion list at OASIS on the topic of a domain-independent web services policy assertion language. This list can be used to assess interest and

develop an appropriate charter for a possible new Technical Committee to standardize such a language. If such a Technical Committee is formed, Sun will contribute *WS-PolicyConstraints* to it on royalty-free terms with the expectation that it will continue to evolve and mature during the standardization process.

### More information on *WS-PolicyConstraints*

The specification for *WS-PolicyConstraints* is available at <http://research.sun.com/projects/xacml/ws-policy-constraints-current.pdf>. Other material related to *WS-PolicyConstraints* may be found at <http://research.sun.com/projects/xacml/>.

### References

- [1] W3C, *Web Services Description Language (WSDL) 1.1*, W3C Note. 15 March 2001. Available <http://www.w3.org/TR/wsdl>
- [2] J. Schlimmer, ed., *Web Services Policy Framework (WS-Policy)*. September 2004. Available <http://www-128.ibm.com/developerworks/webservices/library/specification/ws-polfram/>
- [3] U. Yalcinalp, *Proposal for adding Compositors to WSDL 2.0*. 26 January 2004. Available <http://lists.w3.org/Archives/Public/www-ws-desc/2004Jan/053.html>
- [4] T. Moses, ed., *XACML profile for Web-services*, also known as the *Web Services Policy Language (WSPL)*, Working Draft 04, 29 September 2003. <http://www.oasis-open.org/committees/download.php/3661/draft-xacml-wspl-04.pdf>
- [5] A. Anderson, *An Introduction to the Web Services Policy Language*, Fifth IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY'04). 8 June 2004. Available <http://research.sun.com/projects/xacml/Policy2004.pdf>
- [6] T. Nadalin, ed., *Web Services Security Policy Language (WS-SecurityPolicy)*, Version 1.1. July 2005. Available <http://www-128.ibm.com/developerworks/library/specification/ws-secpol/>
- [7] S. Batres and C. Ferris, ed., *Web Services Reliable Messaging Policy Assertion (WS-RM Policy)*. February 2005. See <http://xml.coverpages.org/ni2005-04-19-a.html>
- [8] T. Moses, ed., *eXtensible Access Control Markup Language (XACML)*, Version 2.0. OASIS Standard. 1 February 2005. Available [http://www.oasis-open.org/committees/tc\\_home.php?wg\\_abbrev=xacml](http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml)
- [9] S. Cantor, et al., ed., *Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0*, OASIS Standard, 15 March 2005. Available [http://www.oasis-open.org/committees/tc\\_home.php?wg\\_abbrev=security#samlv20](http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security#samlv20)
- [10] A. Anderson, , ed., *XACML-based Web Services Policy Constraint Language (WS-PolicyConstraints)*, Working Draft 6. 24 October 2005. Available <http://research.sun.com/projects/xacml/ws-policy-constraints-current.pdf>
- [11] W3C, *XML Path Language (XPath) 2.0*, W3C Candidate Recommendation 3 November 2005. Available <http://www.w3.org/TR/xpath20/>
- [12] C. Sharp, ed., *Web Services Policy Attachment (WS-PolicyAttachment)*. September 2004. Available <http://ftpna2.bea.com/pub/downloads/WS-PolicyAttachment.pdf>

## COPYRIGHT

Copyright © Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, California 95054, U.S.A. All rights reserved.

Unlimited copying without fee is permitted provided that the copies are not made nor distributed for direct commercial advantage, and credit to the source is given. Otherwise, no part of this work covered by copyright hereon may be reproduced in any form or by any means graphic, electronic, or mechanical, including photocopying, recording, taping, or storage in an information retrieval system, without the prior written permission of the copyright owner.

## TRADEMARKS

Sun and Sun Microsystems are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

THIS DOCUMENT IS PROVIDED “AS IS” AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.