Predicates for Boolean web service policy languages

Anne H. Anderson
Sun Microsystems Laboratories
Burlington, MA
Anne.Anderson@sun.com

ABSTRACT
Four of the web service policy languages that have been proposed as the basis for a new standard are based on Boolean combinations of predicates. This paper discusses why these types of policy languages are of interest to industry, proposes an abstract layering for them, and compares the predicate forms used by two of these languages.

General Terms
Standardization, Languages.

Keywords
web services, policy.

1. INTRODUCTION
At the W3C Workshop on Constraints and Capabilities for Web Services [1], various proposals for a standard language for use in expressing policies for web services were presented. Four of the languages presented were variations on Boolean combinations of predicates: the *Web Services Policy Framework (WS-Policy)* [2], the *Web Services Description Language (WSDL)* [3] with the addition of *compositors* [4], the *XACML profile for web services (WSPL)* [5], and a language outline from IONA Technologies [6]. These languages differ in the predicates that are used. In WS-Policy, the predicates are *Assertions* that return a Boolean result, but are not otherwise defined in the policy framework itself; *Assertion* definitions are to be provided as part of each domain-specific document that defines items to be controlled by a policy. In WSDL *compositors,* the predicates are WSDL Boolean *Features, Properties,* or nested *compositor* (Boolean operator) expressions; *Features* and *Properties* are not further defined, although some semantic guidance is provided. In XACML WSPL, the predicates are XACML [7] functions that return a Boolean result and operate on *Attributes* and literal values, where an *Attribute* may be a name/type/value triple or a node in an XML document identified by an XPath [8] expression. In the IONA outline, the predicates are simple XML elements, with at most a *Yes/No* parameter; the process of defining the elements to be used is not elaborated in the outline proposal.

All these languages must rely on some mechanism for associating policies with services or service elements. WS-Policy relies on *Web Services Policy Attachment (WS-PolicyAttachment)* [9]. WSDL relies on attachment points defined in WSDL itself. WSPL relies on a specified convention for the use of the XACML *Target* element. The IONA outline does not describe its mechanism.

This paper will discuss why these languages are of interest to industry, will propose an abstraction for the layering of functionality involved in such languages, along with the functions of each layer, and will compare the forms of two types of predicates, discussing their advantages and disadvantages.

WWW 2005, May 10--14, 2005, Chiba, Japan.

2. WEB SERVICE POLICIES
This section describes, from this author's industry point of view, how "web service policy" has come to be defined by industry and why these Boolean combination policy languages have been of interest to industry.

The proponents of these Boolean policy languages view "web service policy" as being focused primarily on those aspects of a service required to establish a connection and a session such that message exchanges can be initiated. This focus arises from the fact that these aspects of policy are almost universal among web services – they all need to establish mutually agreeable security and reliable messaging parameters, for example, and standards for such parameters already exist. The proponents recognize that more complex languages may be required for some application-specific policy negotiations, but before such negotiations can occur, communication must usually be established. It may also be necessary to identify candidate service providers from a large pool, and thus highly efficient policy matching is a primary goal. Access control (web and OS) and security parameter (IPSec) policies with these same constraints have been in production use for years, so the design of "web service policy" languages has tended to grow out of those models.

A standard language for addressing basic web service communication is urgently needed, so industry is looking for a solution that can be standardized quickly. The W3C Workshop's call for position papers included a basic test case that position papers were supposed to address. Several of the Boolean combination policy language proposals included concrete solutions for this test case. Rightly or wrongly, the fact that none of the semantic web language proposals addressed the specific use case did not lessen some industry skepticism about whether semantic web languages are ready for production use in this area.

3. POLICY USES AND PROCESSORS
In order to develop appropriate web service policy languages, it is important to understand how web service policies will be used and which components of a web services architecture will use them.

One important use is simply for a service provider to publish its policies. A service consumer can query the policies of a provider instance and dynamically configure itself to those policies. The policy processor in this case is the consumer service application itself. In addition to processing the policy expression, the consumer must implement any functionality necessary for conforming to the policy. The consumer must understand the semantics of the items controlled by the policy in order to implement this functionality.

A second important use is for a service to verify that communications and messages it receives conform to its own policy. A service may have an internal policy that is more complex or more complete than the policy it publishes publicly,

but any communication would usually need to satisfy at least the published policy. Verifying that a communication or message conforms to a given policy does not require that the verifier understand the semantics of the items controlled by the policy, but only that the verifier know how to match communication or message information against the policy.

A third important use is for determining a mutually agreeable policy between a service consumer and a service provider. This operation might be performed by a service broker that accepts service registrations and client requests for services, and matches consumers with providers where there is a mutually acceptable policy. The entity that determines the mutually compatible policy need not understand the semantics of the policy items, but only that it can determine the intersection between two policies.

Policies can be used in other ways not directly involving interactions between service providers and service consumers. An example would be the use of a policy for describing the values to be used in a particular deployment of a service. Such a policy might specify which of various options supported by the service are to be enabled, and with which values, in this particular deployment. In this case, the service application is the policy processor, and must understand the semantics of the policy items.

4. POLICY LAYERS

Several functional layers can be identified for such policy languages. The languages all require some underlying "vocabulary" that defines the items to be controlled by a policy, some mechanism for expressing predicates related to that vocabulary, a mechanism for expressing Boolean combinations of predicates, and a mechanism for associating the policy with a service or service element. The following diagram illustrates this layering, along with examples of where such layers are specified:

Table 1. Policy layers

Layer	Specification examples		
Vocabulary	WS-Security, WS-Reliability		
Predicate	WS-Security Policy, WS-Reliability Policy	XACML functions	undefined
Boolean combination	WS-Policy	XACML Boolean operators	WSDL compositors
Association	WS-Policy Attachment	XACML target	WSDL

Additional functions can logically be assigned to these layers. 1) An "`or`" of two predicates means that either predicate is acceptable, but at the time communication is established, one of the options must be selected. This suggests there should be a mechanism for specifying preferences among "`or`"d predicates, which would have to be specified at the Boolean combination layer. 2) Likewise, a single predicate may indicate that a range or set of values is acceptable for some item (e.g. "key length must be at least 1024 bits"), yet one value must be selected at the time communication is established. Preferences for these must be specified at the predicate layer. 3) A policy consumer needs to know the universe of items controlled by the policy and the defaults for items not included in the policy: Must there be a predicate for each item? Are unmentioned items prohibited or unrestricted? This functionality belongs at the Boolean combination layer. 4) Depending on how the defaults are specified, the predicate layer may need to provide predicates to indicate that a particular item is prohibited or is unrestricted. 5) In order to match policies, there must be a way to tell which predicates refer to the same underlying vocabulary item. 6) In order to determine if two policies are consistent, there needs to be a way to determine the set of values, if any, that satisfies each of two different predicates over the same vocabulary item.

The major difference between these Boolean combination policy languages is in the way the predicates themselves are defined. The other layers are functionally equivalent, although the syntax differences could affect the ease with which web service specifications can be associated with policies. Since neither the WSDL nor the IONA proposals describe their predicate layers in detail, the remainder of this paper will focus on WS-Policy and WSPL.

5. WS-POLICY

5.1 WS-Policy Overview

WS-Policy is a proprietary specification developed by a group of companies that includes IBM, Microsoft, and BEA. As of the writing of this paper, it has not been submitted to any standards body.

WS-Policy defines two Boolean operators - `<All>` (Boolean "and") and `<ExactlyOne>` (exclusive-or) - that may be applied to sequences of *Assertion* predicates. These operators may be nested. Previous versions of WS-Policy included a mechanism for providing hints about the policy writer's preferences among various alternatives, but this mechanism was omitted from the most recent version.

5.2 WS-Policy Predicate Layer

In WS-Policy, each web service specification must define a set of policy *Assertions* to be used in expressing policy predicates related to the vocabulary defined in the specification. For example, if the underlying vocabulary specification defines an XML schema element `<v:A>` that is to be controlled by web service policies, then there must one or more additional elements defined for use in expressing the policy predicates relating to `<v:A>`.

WS-SecurityPolicy [10], which defines the *Assertion* predicates to be used with the WS-Security [11] vocabulary, is the example used in the WS-Policy specification. Each new domain's vocabulary will require its own set of *Assertion* predicates, although the WS-Policy authors suggest that in the future, such *Assertions* will be defined as part of the underlying vocabulary specification – WS-Security and WS-Reliability are examples of legacy specifications for which external *Assertions* must be defined.

In comparing policies, conceptually each policy is first converted to Disjunctive Normal Form, such that the policies become sequences of acceptable alternative sets of *Assertions*. The intersection of two policies includes the "compatible policy alternatives (if any) included in both requester and provider policies. Intersection is a commutative, associative function that takes two policies and returns a policy." If the intersection is empty, the two policies are incompatible. A set of *Assertions* in one policy is compatible with a set of *Assertions* in another policy if each instance of an *Assertion* type in one policy is compatible with each instance of that *Assertion* type in the other policy. If an instance of a given *Assertion* occurs in only one set, then "the behavior associated with that *Assertion* type is

prohibited in the intersection of those policies", although this interpretation does not seem semantically consistent: if one policy requires encryption, and the other says nothing about encryption, then prohibiting encryption is not compatible with the first policy.

5.3 WS-Policy Predicate Processing

The specification that defines *Assertion* `<vp:A ...>` must Whether two instances of a given *Assertion* type are compatible is determined by the semantics defined in the domain-specific *Assertion* specification. The WS-Policy authors intend to provide guidance to *Assertion* developers on how to write *Assertions* that can be compared easily [12].

An *Assertion* may be a complex XML type. For example:

```
<vp:A attrB="..." attrC="...">
     <vp:D>example1</vp:D>
     <vp:E>25</vp:E>
     <vp:F attrG="..." />
</vp:/A>
```

The specification that defines *Assertion* `<vp:A ...>` must define all possible variations of this element that a service consumer might request, what the intersection of any two instances of this *Assertion* is, which combinations are not allowed, and how the various forms of the *Assertion* relate to acceptable instances of the underlying domain-specific vocabulary that is the subject of the policy. Any policy processor that must verify a message against or compare instances of `<vp:A ...>` must incorporate a code module that implements the semantics specified for `<vp:A ...>`.

6. WSPL

6.1 WSPL Overview

The syntax of WSPL is a strict subset of the OASIS eXtensible Access Control Markup Language (XACML) Standard. Additional semantics have been specified in the WSPL specification. A WSPL prototype has been implemented.

A WSPL policy is a sequence of one or more rules, where each rule represents an acceptable alternative. A rule is a sequence of predicates, all of which must be satisfied in order for the rule to be satisfied. Rules are listed in order of preference, with the most preferred choice listed first. A WSPL policy is in Disjunctive Normal Form, where the rules are logically connected with "OR" and the predicates within each rule are connected with "AND".

A more complete description of WSPL is contained in [13].

6.2 WSPL Predicate Layer

WSPL defines a standard language for use in specifying predicates that constrain domain-specified vocabulary items. WSPL predicates are XACML functions that return Boolean values. The parameters to the functions are XACML *Attributes* and literal values. An *Attribute* corresponds to a domain-defined vocabulary item. *Attributes* are referenced in two ways, depending on how the domain defines them. An *AttributeDesignator* references a vocabulary item using a domain-defined URI and a standard data type. An *AttributeSelector* specifies a vocabulary item using an XPath expression that selects the vocabulary item from a domain-defined XML document. This document is usually an instance of the schema that defines the domain vocabulary.

Each WSPL predicate places a constraint on the value of an *Attribute*. The constraint operators are: equals, greater than, greater than or equal to, less than, less than or equal to, set-equals, and subset. All the comparison operators are strongly typed and must agree with the data types specified for the function parameters. WSPL supports the rich set of data types used in XACML: string, integer, floating point number (double), date, time, Boolean, URI, hexBinary, base64Binary, dayTimeDuration, yearMonthDuration, x500Name, and rfc822Name. These data types are all taken from the XML Schema [14], with the exception of the two duration types taken from XQuery Operators [15], and the two name types taken from XACML.

6.3 WSPL Predicate Processing

In order to find the intersection of two WSPL policies, several steps are performed. First, the *targets* of the two policies must match (*Targets* are described more completely in [13]). If the *targets* do not match, then the two policies are not compatible. Second, a new policy is created in which there is one rule for each pair of rules from the original policies, where the new rule contains all the predicates from the two original rules. For any given set of vocabulary item values, this new policy will return "true" if and only if both original policies would return true, since the new policy retains all the constraints from the two original policies. WSPL rules are listed in order of preference in a policy: if one rule precedes another, then the policy owner prefers the combination of vocabulary item values specified by the first rule to the combination specified by the second rule. By default the entity that performs a policy intersection preserves the preferences of one policy completely, and the preferences of the second policy to the extent that those are consistent with the preferences of the first. More complex preference combining algorithms could be used, but there is always the possibility of preference conflicts, and the combining algorithm must have some mechanism for resolving these.

The next step is to merge the predicates in each of these new rules such that, for each vocabulary item referenced in the new rule, there is a single predicate (or two predicates in the case of a range of vocabulary item values bounded at each end) that will be true if and only if all predicates in the rule that reference that vocabulary item are true. WSPL specifies the computation of such predicates, based on the laws of arithmetic and logic, for every function operator and data type. For example, the two predicates "`Attribute A > Value B`" and "`Attribute A = Value C`" are both true if and only if "`Value B > Value C`" and "`Attribute A = Value C`". If "`Value B`" is not greater than "`Value C`", then the two predicates are incompatible, and thus the new rule can never be true and is eliminated from the new policy. After this step, each remaining rule is internally consistent: there are no conflicting predicates over the same vocabulary item. The two original policies are incompatible if and only if this resulting set of rules is empty.

The intersection of any two policies specified using the WSPL predicate language can be computed. Computing this intersection requires no knowledge of the semantics of the referenced domain-specific vocabulary items, but depends only on the semantics of the set of standard functions and data types. The resulting policy is in a form such that a policy user can select any rule, select values for each vocabulary item consistent with the predicates in that rule, and that resulting set of values will be acceptable to both original policies.

7. COMPARISON OF PREDICATE FORMS

Both these styles of predicate specification have their advantages and disadvantages.

A single WS-Policy predicate can control multiple related items in the underlying vocabulary; each WSPL predicate applies to only one item. We have designed an extension to WSPL, however, that allows predicates pertaining to related items to be grouped.

A WS-Policy predicate can be abstract. For example, one *Assertion* can state that a digital signature is required, without specifying any details about the syntax of that signature. This same *Assertion* could be used with multiple digital signature syntaxes. A WSPL predicate on the other hand, if it uses XPath expressions to reference actual nodes in an instance of the the underlying vocabulary schema, must depend on an actual node value that will be present in particular schema instances. This can make policies complex if there are multiple ways a particular requirement could be met in a schema instance (for example, there are multiple ways to reference an object to be signed in a message when using the XML Digital Signature standard). XACML name/type/value *Attributes* can be defined, however, to accomplish the same abstraction functions as WS-Policy *Assertions*.

The WS-Policy *Assertions* that need to be compared between two policies can be easily determined, because the *Assertions* will usually have the same name; there might be cases where two different *Assertions* might need to be compared, however, as when a consumer asserts a "MaximumBuyingPrice" *Assertion*, while a provider asserts a "MinimumSellingPrice" *Assertion*. Comparable WSPL *AttributeDesignators* can always be matched, because they must have the same name; similar "maximum" and "minimum" semantics are captured in the function operator rather than in the *Attribute* itself. If *AttributeSelectors* using XPath expressions are used, however, there may be multiple expressions that point to the same node in a schema instance. We are trying to define a subset of XPath that uniquely identifies each node to deal with this problem.

A WS-Policy *Assertion* can specify requirements on document creation, such as the requirement that information describing each document processing step be prepended to previous step information, thus allowing the steps to be "undone" in order by the message receiver. An XACML *Attribute* could be defined to express such semantics, but it can not be done with XPath expressions, since there is nothing in the document that indicates the order in which nodes were added. Note that this type of predicate can not be verified against a given message; it must simply be asserted as a requirement on a document processor.

In order to use a WS-Policy *Assertion* for message verification, the verification engine must include special code that knows how to relate that *Assertion* to a particular type of message. A WSPL predicate that uses XPath expressions can be used directly to verify that the predicate is satisfied in a message.

WS-Policy *Assertions* may be defined in proprietary specifications. Even if the specification is eventually standardized, there can be a long period during which the specification is under development and is not available to all implementers of policy processors. Particularly for policies related to application-specific vocabularies, there may be limited incentive to rush the policy specification to standardization. WSPL predicates, however, can refer directly to the underlying vocabulary specification, and the semantics of those predicates are standard and do not depend on the underlying specification. Alternatively, an XSLT can be used to translate information from an instance of a proprietary schema into a non-proprietary format such as XACML *Attributes* for use in specifying policies.

The Boolean operators defined in WS-Policy can be nested, resulting in a compact policy format; in order to process a policy, it must be at least nominally converted to Disjunctive Normal Form. In WSPL, the policies are always in Disjunctive Normal Form. This, along with the fact that functions are used to specify semantics, rather than having the semantics be implicit in the predicate itself, means that a given policy expressed in WSPL will almost always require more bytes for its expression than a corresponding WS-Policy policy.

From this author's industry perspective, the most significant difference between WS-Policy *Assertions* and WSPL predicates is that each *Assertion* has unique domain-defined semantics that must be captured in a code module incorporated into any entity that must process the *Assertion,* either to compare it or to verify it. Each new domain-defined set of *Assertions* requires that policy processors be updated to support those; any change to existing *Assertions* likewise requires processor updates. Any processor that has not been updated will not be able to process new or modified *Assertions,* making it less likely that policies will be interoperable between different platforms. As more and more *Assertions* are defined, the footprint and maintenance complexity of each policy processor increases. WSPL predicates, on the other hand, use a finite, standard set of functions that do not depend on domain-defined semantics. Any WSPL processor can process any WSPL policy, new or old, and regardless of whether the underlying vocabulary is defined in a proprietary specification or not.

As a proof-of-concept, this author has translated all the *Assertions* defined in WS-SecurityPolicy into WSPL. This exercise was successful in demonstrating that WSPL can handle the policy semantics of a real-life domain.

8. SUMMARY

The web service policy languages that use Boolean combinations of predicates differ primarily in the forms those predicates take. In WS-Policy, predicates are XML elements whose syntax and semantics are domain-specific, with each policy item or group of items having its own set of predicates. In WSPL, predicates are standard XACML functions over a reference to a policy vocabulary item and a literal value. Both forms have advantages and disadvantages. The primary advantage of the WS-Policy form is that predicates tend to be compact and easy to read. The primary disadvantage is that policy processors must be configured to support the syntax and semantics of each predicate type that will be used by any policy. The primary advantage of the WSPL form is that a standard policy processor is able both to compute the intersection of any two policies and to verify any message against a policy. The primary disadvantage is that predicates that directly reference nodes in a domain schema instance may be overly specific, although WSPL also supports the creation of new vocabulary items to express more abstract requirements. WS-Policy currently has no preference mechanism, and the semantics of missing predicates appears to be incorrect; WSPL allows policy alternatives to be ordered by preference. WSPL needs an XPath subset that can be used to uniquely identify a policy item.

9. REFERENCES

[1] W3C, *W3C Workshop on Constraints and Capabilities for Web Services*, http://www.w3.org/2004/09/ws-cc-program., 12-13 October 2004.

[2] J. Schlimmer, ed., *Web Services Policy Framework (WS-Policy)*, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglobspec/html/ws-policy.asp, September 2004.

[3] W3C, *Web Services Description Language (WSDL) 1.1*, W3C Note, http://www.w3.org/TR/wsdl, 15 March 2001.

[4] U. Yalcinalp, *Proposal for adding Compositors to WSDL 2.0,* http://lists.w3.org/Archives/Public/www-ws-desc/2004Jan/0153.html, 26 January 2004.

[5] T. Moses, ed., *XACML profile for Web-services*, http://www.oasis-open.org/committees/download.php/3661/draft-xacml-wspl-04.pdf, Working draft 04, 29 Sept 2003 (also known as "*Web Services Policy Language (WSPL)*").

[7] T. Moses, eds., *OASIS eXtensible Access Control Markup Language (XACML)*, OASIS Standard 2.0, http://www.oasis-open.org/committees/xacml, 1 February 2005.

[8] W3C, *XML Path Language (XPath), Version 1.0*, W3C Recommendation, http://www.w3.org/TR/xpath, 16 November 1999.

[9] C. Sharp, ed., *Web Services Policy Attachment (WS-PolicyAttachment)*, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglobspec/html/ws-policy.asp, September 2004.

[10] A. Nadalin, ed., *Web Services Security Policy Language (WS-SecurityPolicy)*, Version 1.0, http://msdn.microsoft.com/webservices/default.aspx?pull=/library/en-us/dnglobspec/html/ws-securitypolicy.asp, 18 December 2002.

[11] A. Nadalin, et al, eds., *WS-Security*, OASIS Standard 1.0, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss, 6 April 2004.

[12] J. Schlimmer, personal communication, 12 October 2004.

[13] A. Anderson, *An Introduction to the Web Services Policy Language (WSPL),* Proceedings of the Fifth IEEE International Workshop on Policies for Distributed Systems and Networks, Yorktown Heights, New York, 7-9 June 2004, pp. 189-192.

[14] W3C, *XML Schema Part 2: Datatypes*, W3C Recommendation, http://www.w3.org/TR/xmlschema-2/, 2 May 2001.

[15] W3C, *XQuery 1.0 and XPath 2.0 Functions and Operators*, W3C Working Draft 2002, http://www.w3.org/TR/2002/WD-xquery-operators-20020816, 16 August 2002.