An Introduction to the Web Services Policy Language (WSPL)

Anne H. Anderson
Sun Microsystems Laboratories
Anne.Anderson@sun.com

Abstract

The Web Services Policy Language (WSPL) is suitable for specifying a wide range of policies, including authorization, quality-of-service, quality-of-protection, reliable messaging, privacy, and application-specific service options. WSPL is of particular interest in several respects. It supports merging two policies, resulting in a single policy that satisfies the requirements of both, assuming such a policy exists. Policies can be based on comparisons other than equality, allowing policies to depend on fine-grained attributes such as time of day, cost, or network subnet address. By using standard data types and functions for expressing policy parameters, a standard policy engine can support any policy. The syntax is a strict subset of the OASIS eXtensible Access Control Markup Language (XACML) Standard. WSPL has been implemented, and is under consideration as a standard policy language for use with web services.

1. Introduction

A web service has various aspects and features that can be controlled or described using policy rules. Examples of such aspects or features include authentication, authorization, quality-of-service, quality-of-protection, reliable messaging, privacy, and application-specific service options. Interoperability, usability, and reliability of web services will benefit from use of a common policy language for expressing these types of policies. The Web Services Policy Language (WSPL) [1] is an excellent candidate for accomplishing this goal.

WSPL is of particular interest in several respects. First, it allows policy negotiation by supporting the merging of policies from two sources. The result is a single policy that represents the intersection of the two source policies, assuming such a policy exists. Second, policies can be based on policy parameter comparisons other than simple equality matching. This allows policies to depend on fine-grained parameters such as time of day, cost, or network subnet address. Third, by using a set of standard data types and functions for expressing policy parameters, a standard implementation of the language can support any policy.

The syntax of WSPL is a strict subset of the OASIS eXtensible Access Control Markup Language (XACML) Standard [2]. WSPL has been implemented, and is under consideration as a standard policy language for web services.

WSPL was developed based on use cases and requirements [3] that were collected and reviewed in a public forum.

The examples in this paper are shown in a general form rather than in the XML [4] syntax actually used by WSPL. Logically, the examples are equivalent to the corresponding policy expressed in the exact XML syntax, but the general form should be much easier for the typical reader to understand.

2. WSPL overview

A WSPL policy is a sequence of one or more rules, where each rule represents an acceptable choice for satisfying the policy. Rules are listed in order of preference, with the most preferred choice listed first.

A WSPL rule is a sequence of predicates. Each 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. A predicate may specify a constraint based on a literal value, as in "A > 3", or it may place a constraint based on the value of another attribute, as in "A > B". All of the predicates in a rule must be satisfied in order for the rule to be satisfied.

Each policy also states the target aspect of the web service that is covered by that policy.

An example WSPL policy, expressed in a general form, is shown in Figure 1.

Proceedings of the Fifth IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY'04)
0-7695-2141-X/04 $ 20.00 © 2004 IEEE

```
Policy (Aspect = “Quality of Protection”) {
  Rule {
    Signature-Algorithm = “RSA-SHA1”,
    Key-Length >= 2048 }
  Rule {
    Signature-Algorithm = “RSA-SHA1”,
    Key-Length >= 1024,
    Source-Domain = “EXAMPLE.COM” }
  Rule {
    Source-Domain = “MY.EXAMPLE.COM” }
}
```

Figure 1. QoP policy example

The target aspect of this policy is the web service's quality of protection (QoP) requirements and offerings. There are three rules, indicating that there are three choices for satisfying the QoP requirements of this web service. The first rule, the most preferred choice, says that the Signature-Algorithm attribute must have the value “RSA-SHA1” and the Key-Length attribute value must be at least 2048 (bits). The second rule states that, if the Source-Domain attribute is “EXAMPLE.COM” (presumably a somewhat trusted domain), then the Signature-Algorithm must still be “RSA-SHA1”, but the Key-Length may be shorter, but at least 1024 (bits). The third rule states that, where the Source-Domain attribute is “MY.EXAMPLE.COM” (presumably the service's local domain), digital signatures are not required at all.

A policy may state offerings or options rather than requirements. An example is shown in Figure 2. This policy might be published by a service that is selling access to a database.

```
Policy (Aspect = “Service Levels”) {
  Rule {
    Member-level = “Gold”,
    Transaction-Fee = 5 }
  Rule {
    Member-level = “Gold”,
    Time >= 9pm,
    Time <= 6am }
  Rule {
    Member-level = “Tin”,
    Transaction-Fee = 25 }
}
```

Figure 2. Service options policy example

This policy specifies that a “Gold” level member pays a fee of 5 (cents) per transaction normally, but between the hours of 9pm and 6am, transactions are free. A “Tin” level member pays a fee of 25 (cents) per transaction.

The policies for all aspects of a web service are collected into a “PolicySet”. A PolicySet, like a Policy, has a target, but in this case the target specifies the service identifier and the service port type. In the case where a service needs different policies for different operations or messages supported by the service, there may be a second level of PolicySets nested inside the top, service level, PolicySet. The target for each second level PolicySet specifies the operation, and optionally the message, to which the policies within the PolicySet apply.

The policy model for a web service policy is shown in Figure 3.

```
PolicySet (target=<port type>) {
  PolicySet (target=<operation/message>) {
    Policy (target=<aspect>) {
      Rule {
        <predicate>, ...
      } ...
    } ...
  } ...
  Policy (target=<aspect>) {
    Rule {
      <predicate>, ...
    } ...
  } ...
}
```

Figure 3: WSPL policy model

There may be any number of operation/message PolicySets, Policies, Rules, and predicates. The use of the operation/message PolicySet level is optional. Where they are used, the Policies within an operation/message PolicySet all apply to the operation and message specified in the target of that PolicySet.

3. Policy negotiation

One of WSPL's strengths as a web service policy language is its ability to support the negotiation of mutually acceptable policies between two services or between a service consumer and the service itself. For example, a user may wish to use a service. Both the user and the service may have policies for the Quality of Protection parameters they are able to support. In order for the user to interact successfully with the service, the user's application must determine whether it can satisfy the provider's requirements, and, if there are multiple acceptable choices, which one is preferred by the user. This negotiation is accomplished by having the user's application merge the user's policy with the policy of the service.

In order to merge two WSPL policies, several steps are performed. First, the targets of the two policies must match. If they do not match, then the two policies are not compatible. Second, rules from the two policies are paired in all possible combinations, sorted first by the preference order of the party doing the merging, and second by the preference order of the other party (other algorithms could be used). Third, each rule pairing is combined to produce a single new rule. If the two rules in the pairing can not be combined, then the pairing is eliminated. The resulting set of rules represents the combined policy. If this set is empty, then the two policies are not compatible.

The step of combining a rule pairing is done by combining the predicates in the two rules. Predicates that constrain the same attribute must be combined in such a way that the resulting predicate represents the intersection of the original predicates. As an example, a rule stating "A >= 200" would be combined with a rule stating "A >= 300" to produce a result of "A >= 300". If predicates that constrain the same attribute from the two rules in the pair can not be combined, then the rule pairing is incompatible and is eliminated. WSPL specifies in detail how to combine all standard predicate types.

Once two policies have been merged, service-defined descriptions of attributes are used to select one rule from the merged policy and to apply its attributes appropriately. For example, one rule might specify "time of day" attribute values that are not compatible with the time at which this particular service request is being made, so that rule would be eliminated from consideration. Another rule might specify a "role" attribute value that this particular requester is unable to supply, so that rule would also be eliminated. Once inapplicable rules have been eliminated, the first remaining rule is typically selected. This rule might specify a "hash algorithm" attribute value to be used as input to the digital signature operation applied to the service request message. A "member status" attribute value might be obtained from a trusted authority as an attribute certificate and supplied as part of the service request. The specification and application of such attribute descriptions is outside the scope of WSPL.

In cases where all policies in a service's PolicySet are to be negotiated, all policies must be merged. In this case, the merging algorithm will first attempt to pair up operator/message PolicySets and Policies by their targets. If both PolicySets do not contain operator/message PolicySets and Policies that have matching Target values, then the two PolicySets are not compatible. In other cases, such as where the user's application is already coordinated with some aspects of a service's policy, it will be necessary to merge only specific policies. The choice of which policies must be merged depends on the service definition.

Policy negotiation may be either static or dynamic. That is, it may be done once for two parties that have static policies, with the result re-used for each communication between them. Alternatively, it may be done at runtime, based on policies that may represent dynamic constraints relevant to a particular service request. Specification of when policy negotiation must occur is outside the scope of WSP, and would be defined by a particular web service.

4. WSPL policy attributes

Very simple policy languages may support only named attributes that are "true" if present, and "false" if missing. WSPL uses XACML attributes, which are always name-value pairs. This allows the use of fine-grained attributes such as cost or time-of-day, where it would not be feasible to specify a policy for each possible value of the attribute, but where it is quite feasible to specify that an attribute must be greater than or equal to a certain value. An attribute that merely needs to be present, but has no other value, is defined as a named attribute with a Boolean value "true".

The definition of the attributes used by a particular service is outside the scope of WSPL. The semantic description of the attribute, while important in creating services and applications, is irrelevant to the WSPL policy and its evaluation. WSPL also does not specify how a policy user obtains values for attributes.

Each attribute must have a name and a data type. The name must be chosen by the service designer in such a way that it will not conflict with names of other attributes that may have different semantics (URLs can be used as attribute names in order to achieve such uniqueness).

Attributes may also be specified by using XPath [5] expressions. In this case, the policy user would be expected to present an XML instance containing values for the attributes when interacting with the service.

WSPL supports the rich set of data types used in XACML: string, integer, floating point number (double), date, time, Boolean, URI, hexBinary, base64-Binary, dayTimeDuration, yearMonthDuration, x500-Name, and rfc822Name. These data types are all taken from the XML Schema, with the exception of the two duration types taken from XQuery Operators [6], and the two name types taken from XACML. Each data type has a corresponding set of supported functions: equal, greater-than, greater-than-or-equal, less-than, less-than-or-equal, set-equals, and subset. This set of data types and functions is powerful enough to allow the expression of rich policies, yet is constrained enough to allow a precise definition of the rules for merging predicates in policies.

5. Design decisions and conclusions

Any language design must make a number of decisions about the underlying data and use models and also about the level of complexity to be supported. This section discusses some of the particular choices that were made in designing WSPL.

First of all, the WSPL language was developed to satisfy a set of use cases and requirements that were collected, reviewed, and published in an open, public forum. The functionality of the language was tailored to satisfy actual use case requirements.

Rather than starting from scratch, the WSPL language was defined as a strict subset of the OASIS eXtensible Access Control Markup Language (XACML) Standard. XACML has been used in a number of projects to date, is available in an open source implementation, has undergone a formal semantic analysis, and is an open standard. By using XACML as the base, WSPL inherits the considerable design work, public scrutiny, and implementation experience that XACML has undergone. Most of the design decisions involved in WSPL were actually made as part of the development of XACML.

One major design decision inherited from XACML is the use of name-value pairs and standard primitive data types for attributes, rather than depending on XML schema extensions. Using attributes defined via XML schema extensions would make merging policies difficult or impossible except where the values of the attributes are exactly equal, since there can be no standard merge algorithms. Use of non-standard algorithms would mean that the policies could not be supported using a base standard policy engine. Note that XACML supports extensibility of data types and functions, although the rich built-in set minimizes the need for such extensions.

Logically, a WSPL Policy is an "or" of "and" predicates, so is in Disjunctive Normal Form. This form makes it feasible to support the policy negotiation feature of the WSPL language. XACML supports full trees of logical expressions, which allows more compact policies. The value of being able to merge policies outweighed the compactness factor, especially considering that most web services will probably use fairly simple policies in their service definitions. Services may use more complex policies internally, and here the full XACML syntax may be used.

The WSPL specification includes bindings to WSDL 1.1 [7], WSDL 1.2, and to SOAP 1.1 [8]. These bindings may need to change as other standards evolve. The core WSPL language would be valuable as part of a number of protocols and standards, and should be considered whenever a "policy" component is being defined.

6. Acknowledgments

Tim Moses (Entrust) conceived of WSPL as a subset of the XACML syntax, and has continued to be the driving force in producing the definition of WSPL. Danfeng Yao (Brown University) implemented WSPL and provided valuable feedback on the language. Seth Proctor (Sun Microsystems Laboratories) and other members of the OASIS XACML TC contributed to the definition of the syntax and semantics of the language.

7. References

[1] 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)").

[2] S. Godik, T. Moses, eds., "OASIS eXtensible Access Control Markup Language (XACML) Version 1.1", OASIS Committee Specification, http://www.oasis-open.org/committees/download.php/4103/cs-xacml-specification-1.1.doc, 24 July 2003.

[3]T. Moses, ed., "Web-services policy language use-cases and requirements", http://www.oasis-open.org/committees/download.php/1608/wd-xacml-wspl-use-cases-04.pdf

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

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

[6] "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.

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

[8] "Simple Object Access Protocol (SOAP 1.1", W3C Note, http://www.w3.org/TR/SOAP, 8 May 2000.