Domain-Independent, Composable Web Services Policy Assertions

Anne H. Anderson
Sun Microsystems, Inc.
anne.anderson@sun.com

Abstract

The current model for the predicates, or "Assertions", used in a WS-Policy instance is for each policy domain to design new schema elements for that domain's Assertions. Their semantics are defined in an associated specification and are domain-specific. This model leads to interoperability and maintenance problems and hinders dynamic service composition.

WS-PolicyConstraints is a domain-independent language for writing Assertions that is based on the Web Services Policy Language subset of XACML; it differs in addressing only the Assertion layer. This paper describes problems with domain-specific Assertions, the WS-PolicyConstraints alternative, and problems encountered in the development of this language.

1. Introduction

Various standards groups have created specifications for supporting Web Services requirements in their domains of interest, such as "secure messaging" and "reliable messaging". To address different environments, these specifications allow various options. Specification of option combinations acceptable to a Web Service is the primary function of what has come to be referred to as the service's "policy", since this information is related to deployment choices rather than to the interfaces and business logic of the service, both of which are addressed by other standards. In order to interact successfully, a Web Services consumer and provider need to select a combination of options that they both support.

There are currently no standard languages for expressing Web Services policies. While other proposals have been made (WSDL 2.0 "compositors" (Boolean operators) [1]; the Web Services Policy Language (WSPL) [2]; Rei and KAoS, both based on semantic web concepts [3]), it is the Web Services Policy Framework (WS-Policy) [4] that has generated the most industry interest. In WS-Policy, as in some of the other proposals, a policy is a Boolean combination of predicates, or "Assertions", that evaluate to "true" for a particular option selection or set of related option selections. Several policy Assertion specifications, such as WS-SecurityPolicy [5] and WS-ReliableMessaging Policy [6], assume a "Boolean combinations" policy model.

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 and present a Kerberos token*"

might be expressed in the following general form:

```
Policy {
  AND {
    "roles  include 'authorized-vendor'",
    XOR {
      "authn tokens include 'X509Certificate'",
      AND {
        "authn tokens include 'KerberosToken'",
        "network address is in
          '129.156.220.0/255.255.255.0'" }}}}
```

where the Boolean combinations layer specifies the "Policy", "AND", and "XOR" parts of this policy (shown in bold). WS-Policy leaves the design and specification of the Assertions themselves, such as "authn tokens include 'KerberosToken'", to groups concerned with particular domains of interest. These groups so far have designed their Assertions as new domain-specific XML elements whose semantics must be derived from a reading of the corresponding specification. As an example, WS-SecurityPolicy is a specification for a set of Assertions designed to describe options regarding the authentication tokens, encryption algorithms for confidentiality, signature algorithms for integrity, etc. that are specified in the WS-Security standard [7].

2. Problems with domain-specific Assertions

Without a common language for expressing Assertions, the cost of developing and maintaining support for each new Assertion is high. Each new Assertion's semantics for verification and intersection must be specified, implemented, adapted to each host platform, tested, and maintained. Some Assertions might even be proprietary, and not all policy processors would have licenses to use them. If a Web Services application depends on a particular Assertion, and one of the service components that must process it does not have the correct code module for that Assertion, the application may not be able to run on that server. This is a serious interoperability concern. These problems are particularly significant with dynamic service composition and large-scale service brokering, where a domain's Assertions may be opaque to the compositor or broker, which does not deal with the domain itself.

Another problem with domain-specific Assertions is dealing with combinations of Assertions to specify a single complex requirement. For example, an Assertion about a signature algorithm type may be applied to another Assertion about X509 authentication tokens. The current solution to this problem in WS-SecurityPolicy is to support policies nested inside of Assertions. This makes it especially difficult to match compatible Assertions between two policies, as the functions of the policy framework and Assertion layers are mixed.

Semantic web policies avoid some of these problems by capturing the semantics of the Assertions in a standard language. Other problems persist, however. The policy components must have access to the ontologies and taxonomies for the domain, so domain-specific information is still required and may not be available. While it is possible to query a semantic web policy using a given set of policy variable values, there is no general way to compute the intersection between two semantic web policies. Businesses may require legally enforceable agreements on what Assertions mean, and so far semantic web definitions have not proven they can meet that test.

3. An alternative: domain-independent assertions

An alternative solution is to express Assertions by using constraint functions over policy variables that depend only on the generic data type of the variable's values. The language for making Assertions needs to know only the semantics of the data type and the semantics of a small set of standard functions for intersecting or comparing sets of values of those data types. The semantics of the policy variables themselves must be understood by the service endpoints that must implement and enforce the policy, as with any policy mechanism, but a generic policy processor can evaluate any Assertion given a set of variable values and can compute the intersection of any two Assertions.

As an example, an Assertion saying "authentication tokens must include an X509Certificate" taken from WS-SecurityPolicy looks as follows (showing only the required information):

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

Using a functional constraints model, this Assertion can be written as:

```
<Function Id="string-equal">
  <Variable Id="authn-token-type"/>
  <Value>X509Token</Value>
</Function>
```

Alternatively, the functional constraints model can use XPath [8] expressions as policy variables in a way that allows the presence of an X509 token in a Web Services message to be verified directly by the policy processor, still without having to understand what an "X509Token" is:

```
<Function Id="must-be-present">
   <Value>
//S11:Envelope/S11:Header/wss:Security/ds:KeyInfo/ds
:X509Data
   </Value>
</Function>
```

At first glance, the functional constraints Assertion looks more complex, but its implementation is much simpler. With the domain-specific model, a special code module must be supplied that understands how to match an <sp:X509Token> element against other Assertions, and how to verify variable values or messages against it to see if they contain an X509 token. With the functional constraints model, any code module that understands the generic functions "string-equal" or "must-be-present" will be able to match or verify these Assertions. Any Assertion, from any domain, that uses string equality constraints or XPath expression node counts can be handled using the same code module. Assertions that require nested policies are simpler in the functional constraints model because all Assertions, "inner" or "outer", can be handled by the same generic code module. The greater complexity of the XML itself is hidden from users, who use Graphic User Interfaces or code developer tool annotations rather than editing Assertions directly.

4. The WS-PolicyConstraints language

WS-PolicyConstraints [9] is a language for writing domain-independent functional constraints over domain-specific policy variables. WS-

PolicyConstraints is based on a subset of WSPL, with the parts of WSPL that overlapped and conflicted with WS-Policy removed. WSPL was selected as a base because its functional constraints come from the OASIS Standard eXtensible Access Control Markup Language (XACML) [10], which has a number of available implementations deployed by numerous vendors. WSPL also specifies simple, efficient operations for computing the intersection of any two of its functional constraints, using type-based equality, floor/ceiling, and set operations.

WS-PolicyConstraints extends WSPL to deal with additional requirements. For example, because WS-PolicyConstraints is designed to work with any Boolean policy framework that may become standard, it allows for differences in the way certain functionality, such as specifying the semantics of a variable for which no constraint is specified, is divided between the framework layer and the Assertion layer. It adds an XML attribute to range constraints to say which end of the range is preferred It also defines constraints for additional useful data types.

Separating the functions of Assertion intersection and verification from the necessarily domain-specific functions of policy variable implementation and enforcement is consistent with models used in the access control and network management worlds, where a domain-specific Policy Enforcement Point is architecturally separate from a domain-independent Policy Decision Point [11].

5. Problems in designing a domain-independent language

As a proof-of-concept, the Assertions defined in WS-SecurityPolicy were expressed successfully using WS-PolicyConstraints [12]. The intent was not to define a replacement for WS-SecurityPolicy, but to use a set of actual Assertions to test the expressiveness of the language.

The original thought was that all Assertions could be stated as functions using XPath expressions into the Web Service messages as the policy variables. The advantage of such Assertions is that the policy processor can verify that a message satisfies a given Assertion without any domain-specific code at all. Some practical problems led to modifications to this approach, however.

First, some Assertions constrain the way a message is to be processed, where the processing is not reflected in the syntax of the message itself. An example is a requirement that intermediaries prepend new metadata headers to existing headers. It became apparent that some policy variables need to be defined independently of the syntax of the message itself. While these variables are domain-specific, the Assertions over them can still be domain-independent. The policy processor will be able to match two instances of such an Assertion to see if they are compatible, but will be unable to verify that the message actually conforms to the Assertion without domain-specific assistance. A similar example is a requirement that referenced authentication tokens must be external to the message. In order to verify this Assertion, the policy processor needs help in recognizing references that are not external, as references use domain-specific forms.

The second problem arose with Assertions over non-XML data embedded in a message, such as Assertions over the content of fields in embedded binary objects. New functions were defined for extracting the values of fields from common types of data, such as X509 Certificates. The expectation is that future data will be expressed using XML, so the number of such domain-specific functions needed should be limited.

Third is the need to support complex policy variables that include several inter-dependent components. A new “limit-scope” function was introduced to group multiple related constraints for this.

A fourth problem arose from the need to match XPath expressions that were being used as identifiers for policy variables. Two different XPath expressions may select the same nodes in an XML document, but it is impossible to determine this in many cases based only on the document's schema. For example, one XPath expression may specify the second instance of a given element, whereas another XPath expression may specify an element having a particular XML attribute value. The nodes selected may be identical for some instances of the schema, but different for others. To address this, XPath expressions were limited to forms for which intersections are well-defined by prohibiting relative expressions, XPath Query functions (such as selecting an element with a particular XML attribute value), and ordered node selectors (such as //msg/element[3]). The new “limit-scope” function for grouping constraints was sufficient to satisfy use cases for the prohibited XPath Query functions. This limited form of XPath is expected to be sufficient for almost all policies, but more application experience is required. Note that while [13] defines the intersection of two general XPath expressions, but there is no guarantee that any XML document instance can actually satisfy the intersection so specified.

6. Co-existence

Certain domain-specific Assertion definitions, such as WS-SecurityPolicy, are already in use. It is neither reasonable nor necessary to expect that such Assertions will be rewritten using WS-PolicyConstraints. Since WS-Policy processors must be designed to support the addition of new Assertion modules, a module to support WS-PolicyConstraints can be plugged in alongside modules for domain-specific Assertions. Policy Assertions written using domain-specific languages will be handled by their respective modules, whereas Assertions written using WS-PolicyConstraints will be handled by its module. This also allows a new domain-specific Assertion to be created if a case arises where WS-PolicyConstraints is not powerful enough to express the necessary semantics.

7. Conclusion

WS-PolicyConstraints promises to significantly reduce the cost of supporting new Assertions. It can improve interoperability and maintainability of Web Services policy processors as well as allowing free development of new application-specific Assertions without requiring changes to deployed policy processors. Since the semantics it must support are limited to a fixed set of functions, it is easy to test an implementation of WS-PolicyConstraints for correctness.

WS-PolicyConstraints has not yet been implemented, but most of its functions are taken from XACML, which has been widely deployed and has an unencumbered open source implementation available; its function matching and intersection operations are taken from WSPL, which has been implemented and used successfully. WS-PolicyConstraints has been successfully used to express a significant collection of domain-specific Assertions, demonstrating its flexibility and expressive power.

8. References

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

[2] 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.

[3] G. Tonti, J. Bradshaw, R. Jeffers, R. Montanari, N. Suri, A. Uszok, "Semantic Web Languages for Policy Representation and Reasoning: A Comparison of KAoS, Rei, and Ponder", ISWC 2003: Proceedings of the Second International Semantic Web Conference, 2003.

[4] J. Schlimmer, ed., "Web Services Policy Framework (WS-Policy)", September 2004 http://schemas.xmlsoap.org/ws/2004/09/policy/.

[5] T. Nadalin, ed., "Web Services Security Policy Language (WS-SecurityPolicy)", Version 1.1, July 2005, http://msdn.microsoft.com/library/en-us/dnglobspec/html/ws-securitypolicy.pdf.

[6] S. Batres and C. Ferris, ed., "Web Services Reliable Messaging Policy Assertion (WS-RM Policy)", February 2005, OASIS WS-RX TC Committee Draft, October 2005, http://www.oasis-open.org/committees/download.php/15189/wsrmp-1.1-spec-cd-01.pdf.

[7] A. Nadalin, et al., ed., "Web Services Security: SOAP Message Security 1.1 (WS-Security 2004)", OASIS WSS TC Working Draft 7, November 2005.

[8] W3C, "XML Path Language (XPath) 2.0", W3C Candidate Recommendation, November 2005.

[9] A. Anderson, ed., "XACML-based Web Services Policy Constraint Language (WS-PolicyConstraints)", Working Draft 6, 24 October 2005, http://research.sun.com/projects/xacml/ws-policy-constraints-current.pdf.

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

[11] A. Westerinen, et al., "Terminology for Policy-Based Management", IETF RFC 3198, November 2001.

[12] A. Anderson, "WS-Security policy profile of WS-PolicyConstraints," Working Draft 3, 28 June 2005, http://research.sun.com/projects/xacml/ws-security-profile-of-ws-policy-constraints-wd-03.pdf.

[13] B. C. Hammerschmidt, M. Kempa, V. Linnemann, "On the Intersection of XPath Expressions", Proceedings of the 9th International Database Engineering & Application Symposium (IDEAS 2005), 25.-27. July 2005, Montreal, Canada