

Web Services Policies

If you were developing an online bookstore and needed to connect with other services such as book distributors and credit-card payment processors, the Web services model would likely offer a good approach. In addition to the

basic Web services architecture, various specifications exist for

ANNE
ANDERSON
Sun
Microsystems

adding security, reliable messaging, and transaction mechanisms to Web services messages. Specifications for privacy, authorization, session management, and so on will likely emerge in the future.

The complication is that these specifications include numerous options to let them meet requirements for various applications. How does a prospective client know which options a particular Web service requires or supports? How does a service installer tell a service which options it should support? Specifying option choices for Web services in basic communication areas such as security, reliable messaging, privacy, and transactions is the immediate goal underlying efforts to define Web services policies.

Functional layers

Services and their administrators might use Web services policies in the following scenarios:

- Administrators could use policies to tell services to use particular subsets of the options they're programmed to support.
- Provider services could publish their policies to let developers program consumer services or client applications that conform to them.

- Consumer services could compare their own policies with those of potential providers to ensure compatibility.
- Services could select compatible policy options to use with each other.
- Services could use policies to verify that transactions conform to agreed-upon policies.
- Third-party service brokers or agents could use policies to match consumers with compatible providers.
- Services could link with each other to provide consumers with more powerful composite services that incorporate the policies of all involved.

A complete solution for these scenarios involves numerous functions for dealing with policies, including transport, authentication, authoring and analysis tools, negotiation mechanisms, compliance auditing, and life-cycle management. For many of these functions, products from different vendors have no need to interoperate—if vendors compete to provide better solutions, users will benefit. For others, however, interoperability between products is required. Because policy expression, policy association with service interfaces, and policy-nego-

tiation protocols all affect interoperability, open standards will be most important in these areas.

To date, industry has devoted little effort to consumer-side service policies or to negotiating policies between consumers and providers. At Web services' current stage of development, a provider service specifies its policies and client developers use implementation-dependent methods to choose one set of policy options for each interface. Consumers have no way to choose the best provider dynamically or automatically, but such mechanisms will be necessary to achieve Web services' full potential.

Although standards groups aren't yet working on policy negotiation, several standards proposals exist for policy expression and for association with provider services. To understand these proposals, it helps to identify which aspects of a policy each one addresses. Logically, we can divide these aspects into functional layers that build on each other:

- a *service interface binding* layer associates a policy with a service interface;
- a *domain binding* layer associates a policy with a policy topic or "domain" (such as authentication or reliable messaging);
- a *policy* layer identifies sets of acceptable policy options;
- an *assertion* or *predicate* layer expresses individual policy options, and
- a *vocabulary* layer identifies the terms and values in a policy option.

Figure 1 illustrates these layers using an example policy for authentication that says "Use either an X.509 certificate with a key length of at least

1024 bits, or else use a Kerberos ticket-granting ticket.” In the example, text color identifies the parts of the policy that belong to the different functional layers.

The service-interface binding layer (magenta) associates a Web service interface description with a policy—using a *policy identifier*, in this example. It can also bind a policy to a group of interfaces, in which case, it’s important to know whether a group policy augments any policies associated with individual interfaces in the group or whether one overrides the other.

The domain-binding layer (red) associates a policy with a particular *functional domain* that the policy covers. The domain might be common to many services (security, privacy, or reliable messaging, for example) or specific to a particular application (such as options for a book-order service). A single service interface can have a separate policy for each domain of interest. In this example, the domain association is part of the policy itself, but the service interface description that points to the policy could specify the domain instead.

The policy layer (black) is the policy itself. In the policy languages currently proposed for Web services, the policy layer provides a *policy envelope* along with Boolean operators like **AND** and **XOR** to let policy writers combine policy options specified at the next layer.

The assertion layer (green) consists of predicates, called *assertions*, that specify particular policy options. The assertion layer defines constraints on variables and values specified in the next layer—the vocabulary.

The vocabulary layer (blue) specifies variable names and potential values that the policy domain uses. An assertion associates a variable defined in the domain’s policy vocabulary with one or more allowed values, also defined in the vocabulary.

Two approaches exist for specifying assertions. For example, to de-

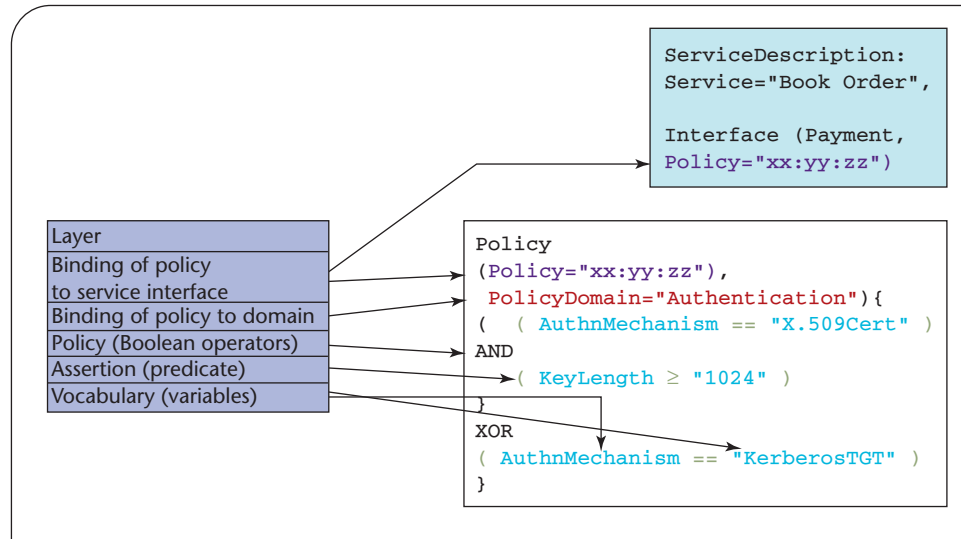


Figure 1. A policy example and its functional elements. The purple box lists functional layers involved in policy expressions. Arrows connect each layer to a corresponding element in the service description (blue box) and in the policy itself (white box). In the white box, all elements in the same text color as an element marked with an arrow belong to the same functional layer.

fine a new assertion saying “for authentication, use X.509 certificates with a minimum key length of 1024 bits,” a policy designer could create a new XML structure, such as

```
<AuthenticationAssertion>
  <Authentication
    Mechanism>X.509Cert
  </Authentication
    Mechanism>
  <MinKeyLength>1024
  </MinKeyLength>
</AuthenticationAssertion>
```

Alternatively, the designer could use a language of generic *constraint functions*, such as those defined in the Extensible Access Control Markup Language (XACML)¹ or in the XQuery and XPath functions and operators.² Using this approach, we could express the same requirements using two constraint functions, such as

```
<StringEquals>
<Variable>
  Authentication
  Mechanism</Variable>
<Value>X.509Cert
</Value>
```

```
</StringEquals>
<IntegerGreaterThanOr
  Equal>
  <Variable>KeyLength
  </Variable>
  <Value>1024</Value>
</IntegerGreaterThanOr
  Equal>
```

Technical committees in standards groups such as the Organization for the Advancement of Structured Information Standards (Oasis) and the World Wide Web Consortium (W3C) have already defined policy vocabularies for specific general domains, and industry groups are likely to define more for their own industry domains. If the policies don’t employ constraint functions, the vocabulary definers must also define new XML structures to associate vocabulary items with values (as in the first example) as well as define how policy software should interpret those structures. A human policy reader can guess that `<MinKeyLength>1024</MinKeyLength>` probably means that any value greater than or equal to 1024 is okay, but a developer must

Table 1. Policy proposals categorized by functional layer.

| LAYER | PROPOSALS ADDRESSING THE LAYER |
|---|--|
| 5. Vocabulary (variables) | WS-ReliableMessaging Policy, WS-SecurityPolicy, WS-AtomicTransaction, Extensible Access Control Markup Language (XACML) |
| 4. Assertion (predicates) | WS-ReliableMessaging Policy, Web Services Policy Language (WSPL), WS-PolicyConstraints, WS-SecurityPolicy, WS-AtomicTransaction, XACML |
| 3. Policy (Boolean operators) | WS-Policy, WSPL, XACML |
| 2. Binding of policy to domain | WS-Policy, WSPL |
| 1. Binding of policy to service interface | WS-PolicyAttachment, WSPL |

program policy-evaluation software to know this. No special software is required to interpret new assertions written using constraint functions; generic constraint-function software can interpret, verify, and, for some languages, negotiate all assertions.

Standards status

Several committees and working groups within Oasis and the W3C are looking at policies. Table 1 shows some of the proposals for addressing the various functional layers.

The policy layer is obviously the key for Web services policies because it determines the general policy model. A group of major vendors developed the WS-Policy³ language for this layer in 2002, and it gained wide support—although its proprietary nature was a significant barrier to the specification of other layers. In April 2006, the vendors finally submitted the WS-Policy specification to the W3C for consideration as a possible standard. Another policy-layer proposal—to extend the Web Services Description Language (WSDL) with Boolean operators—failed to gain sufficient support within the W3C (<http://lists.w3.org/Archives/Public/www-ws-desc/2004Jan/0153.html>); one objection was that service interface descriptions aren't the only places that need to refer to policies, and so a more general solution would be preferable. A third policy-layer proposal is to use some form of XACML, which is already an approved Oasis standard for access control and authorization policies written in XML. Yet, XACML isn't

designed to be a general-purpose Web services policy language. The XACML technical committee designed an XACML profile suitable for general Web services policy use,⁴ as well as for authorization policies for Web services, but standardization of that profile is blocked due to disagreements over whether “Web services policy” falls within the committee charter's scope.

Building on the policy layer is the domain-binding layer. The WS-Policy language includes a place to specify domain identifiers (although the specification doesn't always use these identifiers consistently, the standardization process should clarify this mechanism). The XACML profile for Web services also defines a domain-binding mechanism, but WS-Policy's submission to a standards group leaves little motivation to pursue an alternative at this layer.

The service-interface binding layer builds on the policy and domain-binding layers. To satisfy the requirements at this layer, the WS-Policy developers created a separate proposal called WS-PolicyAttachment,⁵ which extends WSDL to include policy bindings. The developers submitted the WS-PolicyAttachment specification to the W3C in April along with WS-Policy. The XACML profile for Web services also defines a service-interface binding mechanism, but again, there's now little motivation to pursue it.

The policy layer builds on the assertion layer. Several standards groups assumed that WS-Policy or something similar would eventually

become a standard and worked with that model. Given that WS-Policy doesn't define a language for the assertion layer, each of these groups wrote its own by defining new XML structures for assertions using that group's vocabulary. The Oasis Reliable Exchange (WS-RX) technical committee, for example, uses the WS-ReliableMessaging Policy proposal for reliable messaging policy assertions.⁶ For security-related assertions, the new Oasis Secure Exchange (WS-SX) technical committee is using the WS-SecurityPolicy proposal,⁷ which addresses a domain that spans standards under both the WS-SX technical committee and the Oasis Web Services Security (WSS) technical committee. XACML, which uses constraint functions, is the obvious choice for authorization-policy assertions because it's already an Oasis standard. A policy written in WS-Policy covering multiple domains might include an entire XACML policy as a single, complex assertion for the authorization domain. Developers could also use XACML's constraint functions as a generic language for writing assertions in any domain. The Oasis domain-independent policy-assertion language mailing list (<http://lists.oasis-open.org/archives/dipal-discuss/>) recently discussed a language called WS-PolicyConstraints, which uses XACML constraint functions to express policy assertions for any domain's vocabulary and can also handle automated policy negotiation. It's not clear whether this language will move toward standardization, however. Yet another asser-

tion layer option is the XQuery language, although it doesn't support automated policy negotiation.

Most standards groups have defined their vocabularies as part of their assertions. XACML defines a few standard variables, such as "current time," which are likely to be useful in many domains, but authorization vocabularies are nearly always application- or enterprise-specific, and so are unlikely to require global standardization.

The current trend toward using new XML structures for each new assertion type will inhibit policy interoperability, maintainability, and automated negotiation. A generic constraint-function language such as WS-PolicyConstraints would help, but even a standardized language for constructing assertions leaves the problem of too many options: if every service can choose among increasing numbers of policy options, the probability of any two services having compatible policies diminishes. As one possible solution, organizations such as the Liberty Alliance (www.projectliberty.org) and the Web Services Interoperability Organization (www.ws-i.org) might develop profiles specifying a small number of options for individual policy domains once the underlying standards are available.

More generally, it's unclear whether the simple Boolean predicate approach to Web services policies will be adequate to support more dynamic, composable Web services and a wider variety of policy domains. The current Web services policy model doesn't support delegation, trust negotiation, or obligations associated with policy choices, for example. Increasingly complex Web services in the future could require more complex policy languages. □

References

1. T. Moses, ed., *Extensible Access Control Markup Language (XACML)*,

version 2.0, Organization for the Advancement of Structured Information Standards standard, 1 Feb. 2005; www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml.

2. *XQuery 1.0 and XPath 2.0 Functions and Operators*, World-Wide Web Consortium (W3C) candidate recommendation, 3 Nov. 2005; www.w3.org/TR/xquery-operators/.
3. J. Schlimmer et al., *Web Services Policy Framework (WS-Policy)*, joint specification by IBM, BEA Systems, Microsoft, SAP AG, Sonic Software, and VeriSign, 9 Mar. 2006; www.ibm.com/developerworks/library/ws-polfram/.
4. A. Anderson, "An Introduction to the Web Services Policy Language," *Proc. Fifth IEEE Int'l Workshop on Policies for Distributed Systems and Networks (Policy 04)*, IEEE CS Press, 2004; available at <http://research.sun.com/projects/xacml/Policy2004.pdf>.
5. C. Sharp et al., *Web Services Policy Attachment (WS-PolicyAttachment)*, joint specification by BEA Systems, IBM, Microsoft, SAP AG, and Sonic Software, 9 Mar. 2006; www.ibm.com/developerworks/library/specification/ws-polatt/.
6. D. Davis et al., *Web Services Reliable Messaging Policy Assertion (WS-RM Policy)*, Organization for the Advancement of Structured Information Standards, committee draft 03, 14 Mar. 2006; www.oasis-open.org/committees/tc_home.php?wg_abbrev=ws-rx.
7. C. Kaler and A. Nadalin, *Web Services Security Policy Language (WS-Security Policy)*, version 1.1, joint specification by IBM, Microsoft, RSA Security, and VeriSign, July 2005; www.ibm.com/developerworks/library/specification/ws-secpol/.

Anne Anderson is a staff engineer at Sun Microsystems. Her current research interests include policy languages and distributed systems security. Anderson has an MS in computer science from San Diego State University. She is a member of the IEEE and the ACM, and is one of the developers of the OASIS Extensible Access Con-

trol Markup Language (XACML). Contact her at anne.anderson@sun.com.