Web Services Policy 1.5 - Primer

W3C Working Draft 05 June 2007

This version:
http://www.w3.org/TR/2007/WD-ws-policy-primer-20070605
Latest version:
http://www.w3.org/TR/ws-policy-primer
Previous version:
http://www.w3.org/TR/2007/WD-ws-policy-primer-20070330
Editors:
Asir S Vedamuthu, Microsoft Corporation
David Orchard, BEA Systems, Inc.
Frederick Hirsch, Nokia
Maryann Hondo, IBM Corporation
Prasad Yendluri, webMethods, Inc.
Toufic Boubez, Layer 7 Technologies
Ümit Yalçinalp, SAP AG.

This document is also available in these non-normative formats: PDF, PostScript, XML, and plain text.

Abstract

Web Services Policy 1.5 - Primer is an introductory description of the Web Services Policy language. This document describes the policy language features using numerous examples. The associated Web Services Policy 1.5 - Framework and Web Services Policy 1.5 - Attachment specifications provide the complete normative description of the Web Services Policy language.

Status of this Document

This section describes the status of this document at the time of its publication. Other documents may supersede this document. A list of current W3C publications and the latest revision of this technical report can be found in the W3C technical reports index at http://www.w3.org/TR/.

This is an updated Working Draft of the Web Services Policy 1.5 - Primer specification. This Working Draft was produced by the members of the Web Services Policy Working Group, which is part of the W3C Web Services Activity. The Working Group has not yet decided if it will advance this Working Draft to Recommendation Status. Several issues have already been filed on this document and are recorded in Bugzilla. The Working Group has not yet considered all of these issues.

A list of changes in this version of the document [p.45] and a diff-marked version against the previous version of this document are available. Changes in this version of the document encompass editorial changes to align with the OASIS WS-SecurityPolicy and the W3C WS-Addressing Metadata specification.

Note that this Working Draft does not necessarily represent a consensus of the Working Group. Discussion of this document takes place on the public public-ws-policy@w3.org mailing list (public archive) and within Bugzilla. Comments on this specification should be made following the Description for Issues of the Working Group.

Table of Contents

Appendices

---

1. Introduction

This document, *Web Services Policy 1.5 - Primer*, provides an introductory description of the Web Services Policy language and should be read alongside the formal descriptions contained in the WS-Policy and WS-PolicyAttachment specifications.

This document is:

- for policy expression authors who need to understand the syntax of the language and understand how to build consistent policy expressions,
- for policy implementers whose software modules read and write policy expressions and
- for policy assertion authors who need to know the features of the language and understand the requirements for describing policy assertions.

This document assumes a basic understanding of XML 1.0, Namespaces in XML, WSDL 1.1 and SOAP.

Each major section of this document introduces the features of the policy language and describes those features in the context of concrete examples.

2. Basic Concepts: Policy Expression [p.4] covers the basic mechanisms of Web Services Policy. It describes how to declare and combine capabilities and requirements of a Web service as policy expressions, attach policy expressions to WSDL constructs such as endpoint and message, and re-use policy expressions.

3. Advanced Concepts: Policy Expression [p.17] this is the advanced section that provides more in-depth materials for policy implementers and assertion authors. It explains the basics of normalizing policy expressions, merging policies, determining the compatibility (intersection) of policies, the policy data model, the policy expression and the extensibility points built into the Web Services Policy language.

4. Versioning Policy Language [p.36] provides examples and best practices on versioning of the policy language itself, mostly intended for policy implementers.

The Web Services Policy 1.5 - Guidelines for Policy Assertion Authors specification provides guidelines for designing policy assertions and enumerates the minimum requirements for describing policy assertions in specifications.

This is a non-normative document and does not provide a definitive specification of the Web Services Policy language. **B. XML Namespaces** [p.41] lists all the namespaces that are used in this document. (XML elements without a namespace prefix are from the Web Services Policy XML Namespace.)

2. Basic Concepts: Policy Expression

2.1 Web Services Policy

Web services are being successfully used for interoperable solutions across various industries. One of the key reasons for interest and investment in Web services is that they are well-suited to enable service-oriented systems. XML-based technologies such as SOAP, XML Schema and WSDL provide a broadly-adopted foundation on which to build interoperable Web services. The WS-Policy and WS-PolicyAttachment specifications extend this foundation and offer mechanisms to represent the capabilities and requirements of Web services as Policies.

Service metadata is an expression of the visible aspects of a Web service, and consists of a mixture of machine- and human-readable languages. Machine-readable languages enable tooling. For example, tools that consume service metadata can automatically generate client code to call the service. Service metadata can describe different parts of a Web service and thus enable different levels of tooling support.

First, service metadata can describe the format of the payloads that a Web service sends and receives. Tools can use this metadata to automatically generate and validate data sent to and from a Web service. The XML Schema language is frequently used to describe the message interchange format within the SOAP message construct, i.e. to represent SOAP Body children and SOAP Header blocks.

Second, service metadata can describe the ‘how’ and ‘where’ a Web service exchanges messages, i.e. how to represent the concrete message format, what headers are used, the transmission protocol, the message exchange pattern and the list of available endpoints. The Web Services Description Language is currently the most common language for describing the ‘how’ and ‘where’ a Web service exchanges messages. WSDL has extensibility points that can be used to expand on the metadata for a Web service.

Third, service metadata can describe the capabilities and requirements of a Web service, i.e. representing whether and how a message must be secured, whether and how a message must be delivered reliably, whether a message must flow a transaction, etc. Exposing this class of metadata about the capabilities and requirements of a Web service enables tools to generate code modules for engaging these behaviors. Tools can use this metadata to check the compatibility of requesters and providers. Web Services Policy can be used to represent the capabilities and requirements of a Web service.

Web Services Policy is a machine-readable language for representing the capabilities and requirements of a Web service. These are called 'policies'. Web Services Policy offers mechanisms to represent consistent combinations of capabilities and requirements, to determine the compatibility of policies, to name and reference policies and to associate policies with Web service metadata constructs such as service, endpoint and operation. Web Services Policy is a simple language that has four elements - `Policy, All`, `ExactlyOne` and `PolicyReference` - and one attribute - `wsp:Optional`.

2.2 Simple Message

Let us start by considering a SOAP Message in the example below.

Example 2-1. SOAP Message

```
<soap:Envelope>
  <soap:Header>
            <wsa:To>http://x.example.com/realquote</wsa:To>
            <wsa:Action>http://x.example.com/GetRealQuote</wsa:Action>
  </soap:Header>
  <soap:Body>...</soap:Body>
</soap:Envelope>
```

This message uses message addressing headers. The `wsa:To` and `wsa:Action` header blocks identify the destination and the semantics implied by this message respectively. (The prefix `wsa` is used here to denote the Web Services Addressing XML Namespace. **B. XML Namespaces** [p.41] lists all the namespaces and prefixes that are used in this document.)

Let us look at a fictitious scenario used in this document to illustrate the features of the policy language. A Web service developer is building a client application that retrieves real time stock quote information from Company-X, Ltd. Company-X supplies real time data using Web services. The developer has Company-X's advertised WSDL description of these Web services. Company-X requires the use of addressing headers for messaging. Just the WSDL description is not sufficient for the developer to enable the interaction between her client and these Web services. WSDL constructs do not indicate requirements such as the use of addressing.

(*The example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious. No association with any real company, organization, product, domain name, email address, logo, person, places, or events is intended or should be inferred.*)

Providers have the option to convey requirements, such as the use of addressing, through word-of-mouth and documentation – as they always have. To interact successfully with this service, the developer may have to read any related documentation, call someone at Company-X to understand the service metadata, or look at sample SOAP messages and infer such requirements or behaviors.

Web Services Policy is a machine-readable language for representing these Web service capabilities and requirements as policies. Policy makes it possible for providers to represent such capabilities and requirements in a machine-readable form. For example, Company-X may augment the service WSDL description with a policy that requires the use of addressing. The client application developer can use a policy-aware client that understands this policy and engages addressing automatically.

How does Company-X use policy to represent the use of addressing? The example below illustrates a policy expression that requires the use of addressing.

Example 2-2. Policy Expression

```
<Policy>
  <wsam:Addressing>…</wsam:Addressing>
</Policy>
```

The policy expression in the above example consists of a Policy main element and a child element wsam:Addressing. Child elements of the Policy element are policy assertions. Company-X attaches the above policy expression to a WSDL binding description.

Example 2-3. Policy Expression Attached to Binding

```
<wsdl:binding name="AddressingBinding" type="tns:RealTimeDataInterface" >
  <Policy>
    <wsam:Addressing>…</wsam:Addressing>
  </Policy>
  …
</wsdl:binding>
```

Policies can also be attached to WSDL using references (See **2.10 Referencing Policy Expressions** [p.13] .)

The `wsam:Addressing` element is a policy assertion. (The prefix `wsam` is used here to denote the Web Services Addressing – Metadata XML Namespace.) This assertion identifies the use of Web Services Addressing information headers. A policy-aware client can recognize this policy assertion, engage addressing automatically, and use headers such as `wsa:To` and `wsa:Action` in SOAP Envelopes.

It is important to understand the association between the SOAP message and policy expression in the above example. As you can see by careful examination of the message, there is no reference to any policy expression. Just as WSDL does not require a message to reference WSDL constructs (such as port, binding and portType), Web Services Policy does not require a message to reference a policy expression though the policy expression describes the message.

2.3 Secure Message

In addition to requiring the use of addressing, Company-X requires the use of transport-level security for protecting messages.

Example 2-4. Secure Message

```
<soap:Envelope>
  <soap:Header>
    <wss:Security soap:mustUnderstand="1" >
      <wsu:Timestamp wsu:Id="_0">
       <wsu:Created>2006-01-19T02:49:53.914Z</u:Created>
       <wsu:Expires>2006-01-19T02:54:53.914Z</u:Expires>
      </wsu:Timestamp>
    </wss:Security>
   <wsa:To>http://x.example.com/quote</wsa:To>
   <wsa:Action>http://x.example.com/GetRealQuote</wsa:Action>
  </soap:Header>
  <soap:Body>...</soap:Body>
</soap:Envelope>
```

The SOAP message in the example above includes security timestamps that express creation and expiration times of this message. Company-X requires the use of security timestamps and transport-level security - such as `HTTPS` – for protecting messages. (The prefixes `wss` and `wsu` are used here to denote the Web Services Security and Utility namespaces.)

Similar to the use of addressing, Company-X indicates the use of transport-level security using a policy expression. The example below illustrates a policy expression that requires the use of addressing and transport-level security for securing messages.

Example 2-5. Addressing and Security Policy Expression

```
<Policy>
  <wsam:Addressing>...</wsam:Addressing>
  <sp:TransportBinding>...</sp:TransportBinding>
</Policy>
```

The `sp:TransportBinding` element is a policy assertion. (The prefix `sp` is used here to denote the Web Services Security Policy XML Namespace.) This assertion identifies the use of transport-level security – such as `HTTPS` - for protecting messages. Policy-aware clients can recognize this policy assertion, engage transport-level security for protecting messages and include security timestamps in SOAP Envelopes.

The client application developer can use a policy-aware client that recognizes this policy expression and engages both addressing and transport-level security automatically.

For the moment, let us set aside the contents of the `sp:TransportBinding` policy assertion and consider its details in a later section.

2.4 Other Assertions

Thus far, we explored how Company-X uses policy expressions and assertions for representing behaviors that must be engaged for a Web service interaction. What is a policy assertion? What role does it play? In brief, a policy assertion is a piece of service metadata, and it identifies a domain (such as messaging, security, reliability and transaction) specific behavior that is a requirement. Company-X uses a policy assertion to convey a condition under which they offer a Web service. A policy-aware client can recognize policy assertions and engage these behaviors automatically.

Providers, like Company-X, have the option to combine behaviors for an interaction from domains such as messaging, security, reliability and transactions. Using policy assertions, providers can represent these behaviors in a machine-readable form. Web service developers can use policy-aware clients that recognize these assertions and engage these behaviors automatically.

Who defines policy assertions? Where are they? Policy assertions are defined by Web services developers, product designers, protocol authors and users. Like XML Schema libraries, policy assertions are a growing collection. Several WS-* protocol specifications and applications define policy assertions:

- Web Services Security Policy [*WS-SecurityPolicy [p.44]*]
- Web Services Reliable Messaging Policy [*Web Services Reliable Messaging Policy [p.44]*]
- Web Services Atomic Transaction [*Web Services Atomic Transaction [p.43]*]
- Web Services Business Activity Framework [*Web Services Business Activity Framework [p.43]*]
- Devices Profile for Web Services [*Devices Profile for Web Services [p.43]*]
- …

2.5 Combining Policy Assertions

Policy assertions can be combined in different ways to express consistent combinations of behaviors (capabilities and requirements). There are three policy operators for combining policy assertions: `Policy`, `All` and `ExactlyOne` (the `Policy` operator is a synonym for `All`).

Let us consider the `All` operator first. The policy expression in the example below requires the use of addressing and transport-level security. There are two policy assertions. These assertions are combined using the `All` operator. Combining policy assertions using the `Policy` or `All` operator means that all the behaviors represented by these assertions are required.

Example 2-6. Addressing and Security Policy Expression

```
<All>
  <wsam:Addressing>…</wsam:Addressing>
  <sp:TransportBinding>…</sp:TransportBinding>
</All>
```

In addition to requiring the use of addressing, Company-X allows either the use of transport- or message-level security for protecting messages. Web Services Policy language can indicate this choice of behaviors in a machine-readable form. To indicate the use of message-level security for protecting messages, Company-X uses the `sp:AsymmetricBinding` policy assertion (see the example below).

Example 2-7. Asymmetric Binding Security Policy Assertion

```
<sp:AsymmetricBinding>…</sp:AsymmetricBinding>
```

The `sp:AsymmetricBinding` element is a policy assertion. (The prefix `sp` is used here to denote the Web Services Security Policy namespace.) This assertion identifies the use of message-level security – such as *WS-Security 1.0* - for protecting messages. Policy-aware clients can recognize this policy assertion, engage message-level security for protecting messages and use headers such as `wss:Security` in SOAP Envelopes.

To allow the use of either transport- or message-level security, Company-X uses the `ExactlyOne` policy operator. Policy assertions combined using the `ExactlyOne` operator requires exactly one of the behaviors represented by the assertions. The policy expression in the example below requires the use of either transport- or message-level security for protecting messages.

Example 2-8. Transport- or Message-Level Security Policy Expression

```
<ExactlyOne>
  <sp:TransportBinding>…</sp:TransportBinding>
  <sp:AsymmetricBinding>…</sp:AsymmetricBinding>
</ExactlyOne>
```

Company-X requires the use of addressing and requires the use of either transport- or message-level security for protecting messages. They represent this combination using the `All` and `ExactlyOne` operators. Policy operators can be mixed to represent different combinations of behaviors (capabilities and requirements). The policy expression in the example below requires the use of addressing and one of transport- or message-level security for protecting messages.

Example 2-9. Addressing and Transport- OR Message-Level Security Policy Expression

```
<All>
  <wsam:Addressing>…</wsam:Addressing>
  <ExactlyOne>
    <sp:TransportBinding>…</sp:TransportBinding>
    <sp:AsymmetricBinding>…</sp:AsymmetricBinding>
  </ExactlyOne>
</All>
```

Using this policy expression, Company-X gives the choice of mechanisms for protecting messages to clients (or requesters).

2.6 Optional Policy Assertion

Through a customer survey program, Company-X learns that a significant number of their customers prefer to use the Optimized MIME Serialization (as defined in the MTOM specification) for sending and receiving messages. Company-X adds optional support for the Optimized MIME Serialization and expresses this optional behavior in a machine-readable form.

To indicate the use of optimization using the Optimized MIME Serialization, Company-X uses the `mtom:OptimizedMimeSerialization` policy assertion (see the example below).

Example 2-10. Optimized MIME Serialization Policy Assertion

```
<mtom:OptimizedMimeSerialization />
```

The `mtom:OptimizedMimeSerialization` element is a policy assertion. (The prefix `mtom` is used here to denote the Optimized MIME Serialization Policy namespace.) This assertion identifies the use of MIME Multipart/Related serialization as required for request and response messages. Policy-aware clients can recognize this policy assertion and engage Optimized MIME Serialization for messages. The semantics of this assertion are reflected in messages: they use an optimized wire format (MIME Multipart/Related serialization).

Like Company-X's optional support for Optimized MIME Serialization, there are behaviors that may be engaged (in contrast to must be engaged) for a Web service interaction. A service provider will not fault if these behaviors are not engaged. Policy assertions can be marked optional to represent behaviors that may be engaged for an interaction. A policy assertion is marked as optional using the `wsp:Optional` attribute. Optional assertions represent the capabilities of the service provider as opposed to the requirements of the service provider.

In the example below, the Optimized MIME Serialization policy assertion is marked optional. This policy expression allows the use of optimization and requires the use of addressing and one of transport- or message-level security. If a client sends an optimized (MTOM) message, this will be indicated by characteristics associated by using such an optimized message, including a wire format that is a Multipart/Related message and a content-type header of "application/xop+xml" for the outer package. In this case, the response message will also be optimized, also having a Multipart/Related message and content-type header of "application/xop+xml". Note that when optimized messages are used, the Multipart/Related message can have a single part containing the primary SOAP envelope.

Example 2-11. Optional MIME Serialization, Addressing and Transport- OR Message-Level Security Policy Expression

```
<All>
  <mtom:OptimizedMimeSerialization wsp:Optional="true"/>
  <wsam:Addressing>…</wsam:Addressing>
  <ExactlyOne>
    <sp:TransportBinding>…</sp:TransportBinding>
    <sp:AsymmetricBinding>…</sp:AsymmetricBinding>
  </ExactlyOne>
</All>
```

Company-X is able to meet their customer needs by adding optional support for the Optimized MIME Serialization. Optional support is outlined in section 3.4 Web Services Policy 1.5 - Framework and detailed in section 4.5.2, Web Services Policy 1.5 - Guidelines for Policy Assertion Authors, specifically for Optimized MIME Serialization. An optional policy assertion represents a behavior that may be engaged. When a policy assertion is absent from a policy vocabulary (See section 3.2, Web Services Policy 1.5 - Framework), a policy-aware client should not conclude anything (other than 'no claims') about the absence of that policy assertion. See section **2.11 Attaching Policy Expressions to WSDL** [p.15] on the absence of policy expressions.

2.7 Ignorable Policy Expressions

Suppose Company-X decides that it will log SOAP messages sent and received in an exchange. This behavior has no direct impact on the messages sent on the wire, and does not affect interoperability. Some parties might have a concern about such logging and might decide not to interact with Company-X knowing that such logging is performed. To address this concern, Company-X includes a Logging assertion in its policy to enable such parties to be aware of logging. By marking the Logging assertion with the `wsp:Ignorable` attribute with a value of "true" Company-X indicates that a requester may choose to either ignore such assertions or to consider them as part of policy intersection. An assertion that may be ignored for policy intersection is called an ignorable assertion.

The `wsp:Ignorable` attribute allows providers to clearly indicate which policy assertions indicate behaviors that don't manifest on the wire and may not be of concern to a requester when determining policy compatibility. Using the `wsp:Optional` attribute would be incorrect in this scenario, since it would indicate that the behavior would not occur if the alternative without the assertion were selected.

Example 2-12. Ignorable Logging Policy Assertion

```
<log:Logging wsp:Ignorable="true" />
```

(The log: prefix is used here to denote a hypothetical example namespace for this example logging policy assertion.)

The attribute `wsp:Ignorable` is of type *xs:boolean*. Omitting this attribute is semantically equivalent to including it with a value of "false".

The use of the `wsp:Ignorable` attribute has no impact on normalization. Assertions marked with the `wsp:Ignorable` attribute remain marked with the `wsp:Ignorable` attribute after normalization. Please note that the impact of the ignorable attribute is at the discretion of policy consumers through selection of "lax" or "strict" mode (See **3.4.1 Strict and Lax Policy Intersection** [p.25]). Therefore ignorable assertions may have an effect on determining compatibility of provider and consumer policies.

2.8 Marking Assertions both Optional and Ignorable

As described in the sections above and in Section **3.4.1 Strict and Lax Policy Intersection** [p.25] , the WS-Policy 1.5 specification defines two attributes that can be used to mark an assertion: wsp:Optional and wsp:Ignorable.

The WS-Policy Framework allows a policy assertion to be marked with both "optional" and "Ignorable" attributes simultaneously. The presence of "@wsp:optional=true" on an assertion is a syntactic compact form for two alternatives in normal form, one with the assertion and the other without the assertion. Hence syntactically marking an assertion "A" with both the @wsp:Optional and @wsp:Ignorable with the value of "true" for both, is equivalent to two alternatives; one where the assertion A exists with @wsp:Ignorable=true and the second where the assertion A does not exist.

2.9 Nested Policy Expressions

In the previous sections, we considered two security policy assertions. In this section, let us look at one of the security policy assertions in little more detail.

As you would expect, securing messages is a complex usage scenario. Company-X uses the `sp:TransportBinding` policy assertion to indicate the use of transport-level security for protecting messages. Just indicating the use of transport-level security for protecting messages is not sufficient. To successfully interact with Company-X's Web services, the developer must know what transport token to use, what secure transport to use, what algorithm suite to use for performing cryptographic operations, etc. The `sp:TransportBinding` policy assertion can represent these dependent behaviors. In this section, let us look at how to capture these dependent behaviors in a machine-readable form.

A policy assertion – like the `sp:TransportBinding` - identifies a visible domain specific behavior that is a requirement. Given an assertion, there may be other dependent behaviors that need to be enumerated for a Web Service interaction. In the case of the `sp:TransportBinding` policy assertion, Company-X needs to identify the use of a transport token, a secure transport, an algorithm suite for performing cryptographic operations, etc. A nested policy expression can be used to enumerate such dependent behaviors.

What is a nested policy expression? A nested policy expression is a policy expression that is a child element of a policy assertion element. A nested policy expression further qualifies the behavior of its parent policy assertion.

In the example below, the child `Policy` element is a nested policy expression and further qualifies the behavior of the `sp:TransportBinding` policy assertion. The `sp:TransportToken` is a nested policy assertion of the `sp:TransportBinding` policy assertion. The `sp:TransportToken` assertion requires the use of a specific transport token and further qualifies the behavior of the `sp:TransportBinding` policy assertion (which already requires the use of transport-level security for protecting messages).

Example 2-13. Transport Security Policy Assertion

```
<sp:TransportBinding>
  <Policy>
    <sp:TransportToken>
      <Policy>
        <sp:HttpsToken>
          <wsp:Policy/>
        </sp:HttpsToken>
      </Policy>
    </sp:TransportToken>
    <sp:AlgorithmSuite>
      <Policy>
        <sp:Basic256Rsa15/>
      </Policy>
    </sp:AlgorithmSuite>
    …
  </Policy>
</sp:TransportBinding>
```

The `sp:AlgorithmSuite` is a nested policy assertion of the `sp:TransportBinding` policy assertion. The `sp:AlgorithmSuite` assertion requires the use of the algorithm suite identified by its nested policy assertion (`sp:Basic256Rsa15` *in the example above*) and further qualifies the behavior of the `sp:TransportBinding` policy assertion.

Setting aside the details of using transport-level security, Web service developers can use a policy-aware client that recognizes this policy assertion and engages transport-level security and its dependent behaviors automatically. That is, the complexity of security usage is absorbed by a policy-aware client and hidden from these Web service developers.

In another example, WS-Security Policy defines a sp:HttpToken assertion to contain three possible nested elements, sp:HttpBasicAuthentication, sp:HttpDigestAuthentication and sp:RequireClientCertificate. When the HttpToken is used with an empty nested policy in a policy expression by a provider, it will indicate that none of the dependent behaviors namely authentication or client certificate is required. A non-anonymous client who requires authentication or client certificate will not be able to use this provider solely on the basis of intersection algorithm alone.

Example 2-14. Empty Nested Assertion

```
<sp:TransportToken>
    <wsp:Policy>
      <sp:HttpsToken>
       <wsp:Policy/>
      </sp:HttpsToken>
    </wsp:Policy>
</sp:TransportToken>
```

2.10 Referencing Policy Expressions

Company-X has numerous Web service offerings that provide different kinds of real-time quotes and book information on securities such as `GetRealQuote`, `GetRealQuotes` and `GetExtendedRealQuote`. To accommodate the diversity of Company-X's customers, Company-X supports multiple WSDL bindings for these Web services. Company-X provides consistent ways to interact with their services and wants to represent these capabilities and requirements consistently across all of their offerings without duplicating policy expressions multiple times. How? It is simple - a policy expression can be named and referenced for re-use.

Section **2.2 Simple Message** [p.5] , showed how a policy expression can be attached directly to a binding inline. A single policy expression may be used in several parts of a WSDL document. In this case it is desirable to use references to the policy expression rather than to directly inline the policy expression.

A policy expression may be identified by an IRI and referenced for re-use as a standalone policy or within another policy expression. There are three mechanisms to identify a policy expression: the `wsu:Id` `xml:id` and `Name` attributes. A `PolicyReference` element can be used to reference a policy expression identified using either of these mechanisms.

Example 2-15. Common Policy Expression

```
<Policy wsu:Id="common">
  <mtom:OptimizedMimeSerialization wsp:Optional="true"/>
  <wsam:Addressing>…</wsam:Addressing>
</Policy>
```

In the example above, the `wsu:Id` attribute is used to identify a policy expression. The value of the `wsu:Id` attribute is an XML ID. The relative IRI for referencing this policy expression (within the same document) is `#common`. If the policy document IRI is `http://x.example.com/policy.xml` then the absolute IRI for referencing this policy expression is `http://x.example.com/policy.xml#common.` (The absolute IRI is formed by combining the document IRI, # and the value of the `wsu:Id` attribute.)

In addition to the Example 2-12, Company-X could have used either the xml:id or wsu:Id. An example of the use of xml:id similar to that of wsu:Id is shown in Example 2-13.

Example 2-16. Common Policy Expression [xml:id]

```
<Policy xml:id="common">
  <mtom:OptimizedMimeSerialization wsp:Optional="true"/>
  <wsam:Addressing>…</wsam:Addressing>
</Policy>
```

Conditions and constraints on the use of the |xml:id| attribute in conjunction with Canonical XML 1.0 are specified in Appendix C of *XML ID [p.45]* and are further detailed in *C14N 1.0 Note [p.42]* . Significant care is suggested in the use of xml:id.

Note:

Note: Canonical XML 1.1 [*XMLID11 [p.45]*] is intended to address the issues that occur with Canonical XML 1.0 with regards to `xml:id`. The W3C XML Security Specifications Maintenance WG has been chartered to address how to integrate Canonical XML 1.1 with XML Security, including XML Signature [*SecSpecMaintWG [p.43]*] (See http://www.w3.org/2007/xmlsec/.)

For re-use, a `PolicyReference` element can be used to reference a policy expression as a standalone policy or within another policy expression. The example below is a policy expression that re-uses the common policy expression above.

Example 2-17. PolicyReference to Common Policy Expression

```
<PolicyReference URI="#common"/>
```

For referencing a policy expression within the same XML document, Company-X uses the `wsu:Id` attribute for identifying a policy expression and an IRI to this ID value for referencing this policy expression using a `PolicyReference` element.

The example below is a policy expression that re-uses the common policy expression within another policy expression. This policy expression requires the use of addressing, one of transport- or message-level security for protecting messages and allows the use of optimization.

Example 2-18. Secure Policy Expression

```
<Policy wsu:Id="secure">
  <All>
    <PolicyReference URI="#common"/>
    <ExactlyOne>
      <sp:TransportBinding>...</sp:TransportBinding>
      <sp:AsymmetricBinding>...</sp:AsymmetricBinding >
    </ExactlyOne>
  </All>
</Policy>
```

The `Name` attribute is an alternate mechanism to identify a policy expression. The value of the `Name` attribute is an absolute IRI and is independent of the location of the XML document where the identified policy expression resides in. As such, referencing a policy expression using the `Name` attribute relies on additional out of band information. In the example below, the `Name` attribute identifies the policy expression. The IRI of this policy expression is `http://x.example.com/policy/common`.

Example 2-19. Common Policy Expression

```
<Policy Name="http://x.example.com/policy/common">
  <mtom:OptimizedMimeSerialization wsp:Optional="true"/>
  <wsam:Addressing>...</wsam:Addressing>
</Policy>
```

The example below is a policy expression that re-uses the common policy expression above.

Example 2-20. PolicyReference to Common Policy Expression

```
<PolicyReference URI="http://x.example.com/policy/common"/>
```

As policy expressions are composed from other policy expressions and assertions from different domains are used in a policy expression, complex expressions will emerge. Naming parts of complex expressions for reuse and building more complex policies through referencing enables building more complicated policy scenerios easily. This approach enables the association of additional policy subjects to identified policy expressions. It also promotes manageability of the expressions as they are uniquely identified and allows profiles for common scenerios to be developed. Note that when a named expression has assertions that contains parametrized expressions, care must be given to ensure that the parameterized content is statically available to enable reuse.

2.11 Attaching Policy Expressions to WSDL

A majority of Company-X's customers use WSDL for building their client applications. Company-X leverages this usage by attaching policy expressions to the WSDL binding descriptions.

In the example below, the `SecureBinding` WSDL binding description defines a binding for an interface that provides real-time quotes and book information on securities. (The prefixes `wsdl` and `tns` are used here to denote the Web Services Description language XML namespace and target namespace of this WSDL document.) To require the use of security for these offerings, Company-X attaches the secure policy expression in the previous section to this binding description. The WSDL `binding` element is a common policy attachment point. The secure policy expression attached to the `SecureBinding` WSDL binding description applies to any message exchange associated with any `port` that supports this binding description. This includes all the message exchanges described by operations in the `RealTimeDataInterface`.

Example 2-21. Secure Policy Expression Attached to WSDL Binding

```
<wsdl:binding name="SecureBinding" type="tns:RealTimeDataInterface" >
  <PolicyReference URI="#secure" />
  <wsdl:operation name="GetRealQuote">…</wsdl:operation>
  …
</wsdl:binding>
```

In addition to providing real-time quotes and book information on securities, Company-X provides other kinds of data through Web services such as quotes delayed by 20 minutes and security symbols through Web services (for example `GetDelayedQuote`, `GetDelayedQuotes`, `GetSymbol` and `GetSymbols`). Company-X does not require the use of security for these services, but requires the use of addressing and allows the use of optimization.

Example 2-22. Open Policy Expression Attached to WSDL Binding

```
<wsdl:binding name="OpenBinding" type="tns:DelayedDataInterface" >
  <PolicyReference URI="#common" />
  <wsdl:operation name="GetDelayedQuote">…</wsdl:operation>
  …
</wsdl:binding>
```

In the example above, the `OpenBinding` WSDL binding description defines a binding for an interface that provides other kinds of data such as quotes delayed by 20 minutes and security symbols. To require the use of addressing and allow the use of optimization, Company-X attaches the common policy expression in the previous section to this binding description. As we have seen in the `SecureBinding` case, the common policy expression attached to the `OpenBinding` WSDL binding description applies to any message exchange associated with any `port` that supports this binding description. This includes all the message exchanges described by operations in the `DelayedDataInterface`.

As mentioned earlier, providers have the option to convey requirements, such as the use of addressing or security, through word-of-mouth and documentation – as they always have. The absence of policy expressions, for example, in a WSDL document does not indicate anything about the capabilities and requirements of a service. The service may have capabilities and requirements that can be expressed as policy expressions, such as the use of addressing, security and optimization. Or, the service may not have such capabilities and requirements. A policy aware client should not conclude anything about the absence of policy expressions.

Service providers, like Company-X, can preserve and leverage their investments in WSDL and represent the capabilities and requirements of a Web service as policies. A WSDL document may specify varying behaviors across Web service endpoints. Web service developers can use a policy-aware client that recognizes these policy expressions in WSDL documents and engages behaviors automatically for each of these endpoints. Any complexity of varying behaviors across Web service endpoints is absorbed by a policy-aware client or tool and hidden from these Web service developers.

2.12 Policy Automates Web Services Interaction

As you have seen, Web Services Policy is a simple language that has four elements - `Policy, All`, `ExactlyOne` and `PolicyReference` - and one attribute - `wsp:Optional`. In practice, service providers, like Company-X, use policy expressions to represent combinations of capabilities and requirements. Web service developers use policy-aware clients that understand policy expressions and engage the behaviors represented by providers automatically. A sizable amount of complexity is absorbed by policy-aware clients (or tools) and is invisible to these Web service developers.

Web Services Policy extends the foundation on which to build interoperable Web services, hides complexity from developers and automates Web service interactions.

3. Advanced Concepts: Policy Expression

In **2. Basic Concepts: Policy Expression** [p.4] , we covered the basics of Web Services Policy language. This is the advanced section that provides more in-depth materials for Web Services Policy implementers and assertion authors. This section covers the following topics:

- What is a policy expression?
- What is the normal form of a policy expression and how to normalize policy expressions?
- What is the policy data model?
- How to select a compatible policy alternative?
- How to attach policy expressions to WSDL constructs?
- How to combine policies?
- What are the extensibility points?
- What are the parts of a policy assertion?

3.1 Policy Expression

A policy expression is the XML representation and interoperable form of a Web Services Policy. A policy expression consists of a `Policy` wrapper element and a variety of child and descendant elements. Child and descendent elements from the policy language are `Policy, All`, `ExactlyOne` and `PolicyReference`. Other child elements of `Policy`, `All` and `ExactlyOne` are policy assertions. (The `Policy` element plays two roles: wrapper element and operator.) Policy assertions can contain a nested policy expression. Policy assertions can also be marked optional to represent behaviors that may be engaged (capabilities) for an interaction. The optional marker is the `wsp:Optional` attribute which is placed on a policy assertion element.

Let us take a closer look at Company-X's policy expression (see below) from the previous section.

Example 3-1. Company-X's Secure Policy Expression

```
<Policy>
  <All>
    <mtom:OptimizedMimeSerialization wsp:Optional="true"/>
    <wsam:Addressing>…</wsam:Addressing>
    <ExactlyOne>
     <sp:TransportBinding>…</sp:TransportBinding>
     <sp:AsymmetricBinding>…</sp:AsymmetricBinding >
    </ExactlyOne>
  </All>
</Policy>
```

The `Policy` element is the wrapper element. The `All` and `ExactlyOne` elements are the policy operators. All other child elements of the `All` and `ExactlyOne` elements are policy assertions from domains such as messaging, addressing, security, reliability and transactions.

3.2 Normal Form for Policy Expressions

Web Services Policy language defines two forms of policy expressions: compact and normal form. Up to this point, we have used the compact form. The compact form is less verbose than the normal form. The compact form is useful for authoring policy expressions. The normal form is an intuitive representation of the policy data model. We will look into the policy data model in the next section.

The normal form uses a subset of constructs used in the compact form and follows a simple outline for its XML representation:

Example 3-2. Normal Form for Policy Expressions

```
<Policy>
  <ExactlyOne>
    <All>
     <x:AssertionA>…</x:AssertionA>
     <y:AssertionB>…</y:AssertionB>
       …
    </All>
    <All>
     <x:AssertionA>…</x:AssertionA>
     <z:AssertionC>…</z:AssertionC>
       …
    </All>
   …
  </ExactlyOne>
<Policy/>
```

The normal form consists of a `Policy` wrapper element and has one child `ExactlyOne` element. This `ExactlyOne` element has zero or more `All` child elements. Each of these `All` elements has zero or more policy assertions. The `PolicyReference` element and `wsp:Optional` attribute are not used in the normal form. And, a nested policy expression in the normal form has at most one policy alternative.

The normal form represents a policy as a collection of policy alternatives and a policy alternative as a collection of policy assertions in a straight-forward manner.

The example below is a policy expression in the normal form. This expression contains two policy alternatives: one that requires the use of transport-level security and the other that requires the use of message-level security for protecting messages.

Example 3-3. Transport- or Message-Level Security Policy Expression in Normal Form

```
<Policy>
  <ExactlyOne>
    <All>
     <sp:TransportBinding>…</sp:TransportBinding>
    </All>
    <All>
     <sp:AsymmetricBinding>…</sp:AsymmetricBinding >
    </All>
  </ExactlyOne>
</Policy>
```

A policy expression in the compact form can be converted to the normal form. Web Services Policy language describes the algorithm for this conversion.

Let us re-consider Company-X’s policy expression (see the example below). Company-X requires the use of addressing and either transport- or message-level security and allows the use of optimization. This policy expression is in the compact form and has four policy alternatives for requesters:

1. Requires the use of addressing and transport-level security
2. Requires the use of addressing and message-level security
3. Requires the use of optimization, addressing and transport-level security and
4. Requires the use of optimization, addressing and message-level security.

Example 3-4. Company-X’s Secure Policy Expression in Compact Form

```
<Policy wsu:Id=”secure”>
  <All>
    <PolicyReference URI=”#common”/>
    <ExactlyOne>
     <sp:TransportBinding>…</sp:TransportBinding>
     <sp:AsymmetricBinding>…</sp:AsymmetricBinding >
    </ExactlyOne>
  </All>
</Policy>

<Policy wsu:Id=”common”>
  <mtom:OptimizedMimeSerialization wsp:Optional="true"/>
  <wsam:Addressing>…</wsam:Addressing>
</Policy>
```

Let us look at the normal form for this policy expression. The example below is Company-X's policy expression in the normal form. As you can see, the compact form is less verbose than the normal form. The normal form represents a policy as a collection of policy alternatives. Each of the `All` operators is a policy alternative. There are four policy alternatives in the normal form. These alternatives map to bullets (a) through (d) above.

Example 3-5. Company-X's Policy Expression in Normal Form

```
<Policy>
  <ExactlyOne>
    <All> <!-- - - - - - - - - - - - - - - Policy Alternative (a) -->
       <wsam:Addressing>…</wsam:Addressing>
       <sp:TransportBinding>…</sp:TransportBinding>
    </All>
    <All> <!-- - - - - - - - - - - - - - - Policy Alternative (b) -->
      <wsam:Addressing>…</wsam:Addressing>
      <sp:AsymmetricBinding>…</sp:AsymmetricBinding >
    </All>
    <All> <!-- - - - - - - - - - - - - - - Policy Alternative (c) -->
      <mtom:OptimizedMimeSerialization />
       <wsam:Addressing>…</wsam:Addressing>
       <sp:TransportBinding>…</sp:TransportBinding>
    </All>
    <All> <!-- - - - - - - - - - - - - - - Policy Alternative (d) -->
      <mtom:OptimizedMimeSerialization />
      <wsam:Addressing>…</wsam:Addressing>
      <sp:AsymmetricBinding>…</sp:AsymmetricBinding>
    </All>
  </ExactlyOne>
</Policy>
```

The `wsp:Optional` attribute, nested policy expression and `PolicyReference` element are converted to their corresponding normal form. The `wsp:Optional` attribute converts to two alternatives, one with and the other without the assertion. A policy alternative containing an assertion with a nested policy expression that has multiple policy alternatives converts to multiple policy alternatives where the assertion contains a nested policy expression that has at most one policy alternative.

The `PolicyReference` element is replaced with its referenced policy expression. See section **3.6 Policy Retrieval** [p.29] for more details on how to retrieve referenced policy expressions.

3.3 Policy Data Model

In the previous section, we considered the normal form for policy expressions. As we discussed, the normal form represents a policy as a collection of policy alternatives. In this section, let us look at the policy data model.

Company-X uses a policy to convey the conditions for an interaction. Policy-aware clients, like the one used by the developer in our example (as explained earlier in **2. Basic Concepts: Policy Expression** [p.4]), view policy as an unordered collection of zero or more policy alternatives. A policy alternative is an unordered collection of zero or more policy assertions. A policy alternative represents a collection of behaviors or requirements or conditions for an interaction. In simple words, each policy alternative represents a set of conditions for an interaction. The diagram below describes the policy data model.

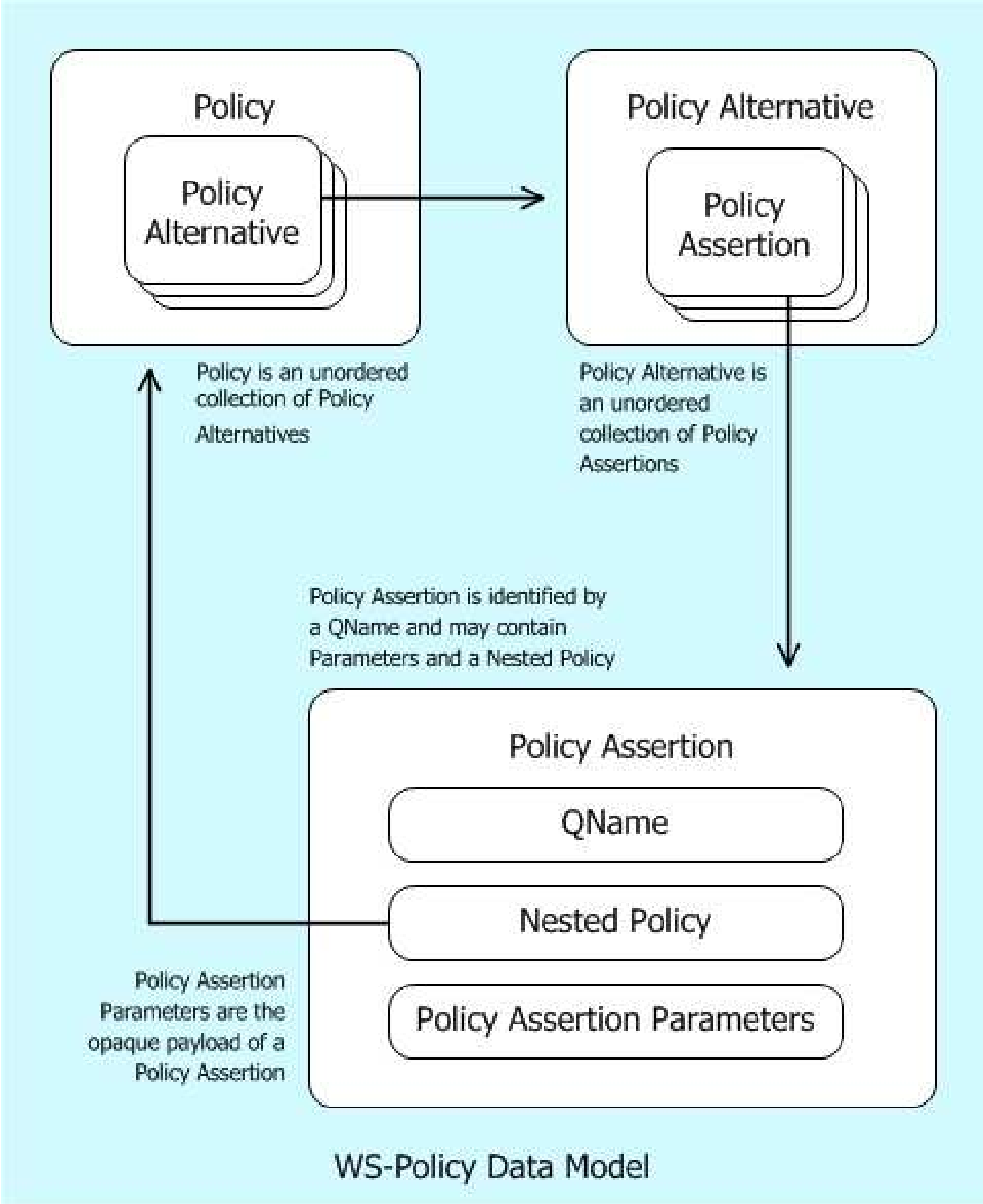


Figure 3-1. WS-Policy Data Model

A policy-aware client uses a policy to determine whether one of these policy alternatives (i.e. the conditions for an interaction) can be met in order to interact with the associated Web Service. Such clients may choose any of these policy alternatives and must choose exactly one of them for a successful Web service interaction. Clients may choose a different policy alternative for a subsequent interaction. It is important to understand that a policy is a useful piece of metadata in machine-readable form that enables tooling, yet is not required for a successful Web service interaction. Why? Web service developers could use the documentation, talk to the service providers, or look at message traces to infer these conditions for an interaction. Developers continue to have these options, as they always had.

As we discussed, a policy assertion identifies a domain specific behavior or requirement or condition. A policy assertion has a QName that identifies its behavior or requirement or condition. In the XML representation, the QName of the assertion element is the QName of the policy assertion. A policy assertion may contain assertion parameters and a nested policy.

The assertion parameters are the opaque payload of an assertion. Parameters carry additional useful pieces of information necessary for engaging the behavior described by an assertion. In the XML representation, the child elements and attributes of an assertion excluding the child elements and attributes from the WS-Policy language XML namespace name, are the assertion parameters. For example @wsp:Optional and @wsp:Ignorable are not assertion parameters.

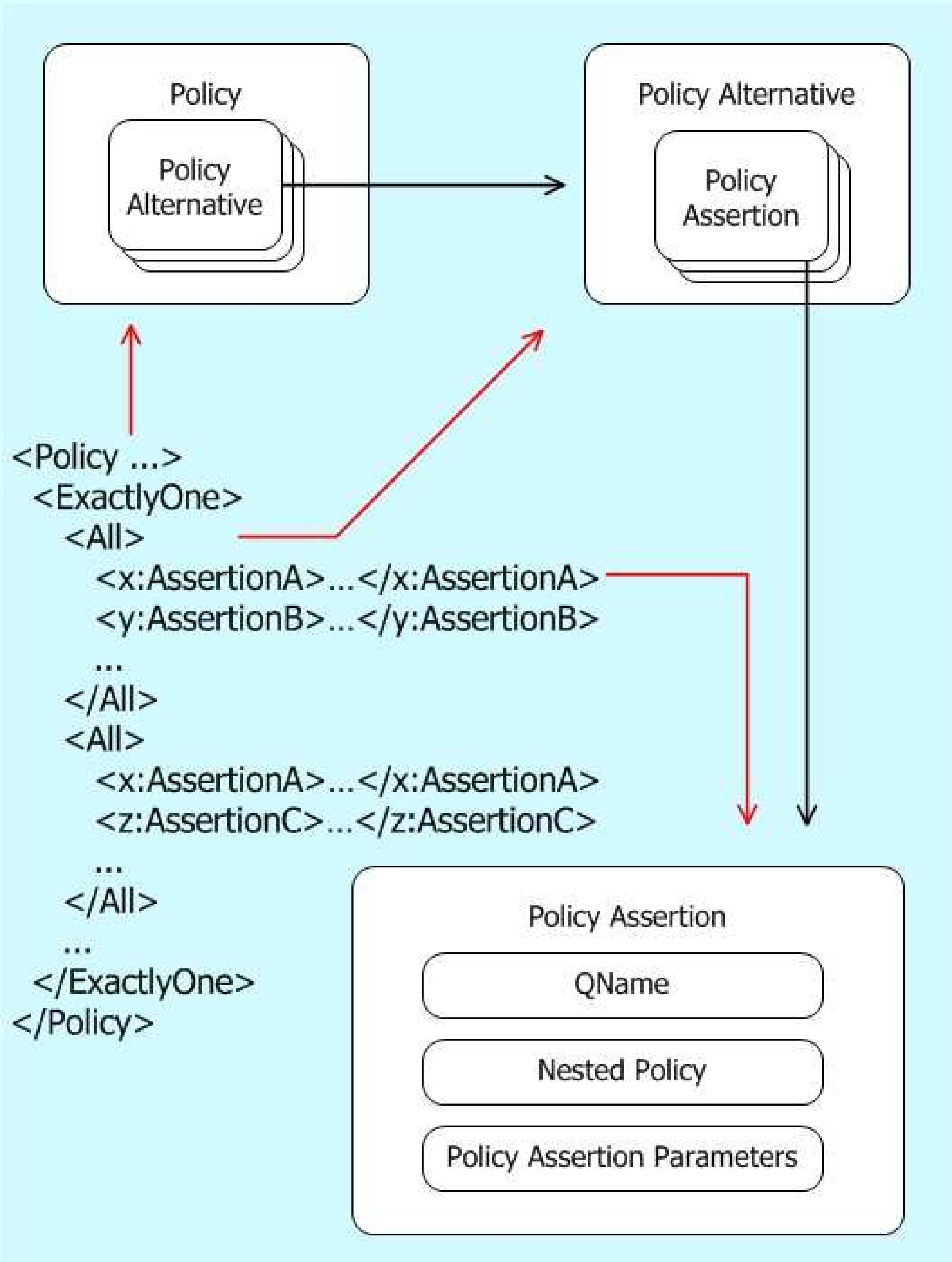
We considered nested policy expressions in the context of a security usage scenario. Let us look at its shape in the policy data model. In the normal form, a nested policy is a policy that has at most one policy alternative and is owned by its parent policy assertion. The policy alternative in a nested policy represents a collection of dependent behaviors or requirements or conditions that qualify the behavior of its parent policy assertion.

A policy-aware client supports a policy assertion if the client engages the behavior or requirement or condition indicated by the assertion. A policy-aware client supports a policy alternative if the client engages the behaviors represented by all the assertions in the alternative. A policy-aware client supports a policy if the client engages the behaviors represented by at least one of the policy alternatives.

In the previous section, we saw how the normal form of a policy expression represents a policy as a collection of policy alternatives. By policy language design, the normal form of a policy expression directly maps to the policy data model:

- Each child element of `Policy/ExactlyOne/All` maps to a policy assertion.
- Each `Policy/ExactlyOne/All` element and policy assertions which correspond to its children map to a policy alternative.
- The `Policy/ExactlyOne` element maps to a collection of policy alternatives.
- The `Policy` wrapper element and policy alternatives which correspond to the `Policy/ExactlyOne` element map to a policy.

The diagram below describes this mapping from the normal form of a policy expression to the policy data model.



Mapping from Normal Form to Policy Data Model

Figure 3-2. Mapping from Normal Form to Policy Data Model

3.4 Compatible Policies

A provider, like Company-X, and a requester, like the policy-aware client used in our example, may represent their capabilities and requirements for an interaction as policies and want to limit their message exchanges to mutually compatible policies. Web Services Policy defines an intersection mechanism for selecting compatible policy alternatives when there are two or more policies.

The example below is a copy of Company-X's policy expression (from **3.2 Normal Form for Policy Expressions** [p.18]). As we saw before, Company-X offers four policy alternatives. Of them, one of the policy alternatives requires the use of addressing and transport-level security.

Example 3-6. Company-X's Policy Expression

```
<Policy>
  <ExactlyOne>
    <All> <!-- - - - - - - - - - -  Company-X's Policy Alternative (a) -->
       <!-- - - - - - - - - - - - - - - - - - - Policy Assertion (c1) -->
       <wsam:Addressing>…</wsam:Addressing>
       <!-- - - - - - - - - - - - - - - - - - - Policy Assertion (c2) -->
      <sp:TransportBinding>…</sp:TransportBinding>
    </All>
    …
  </ExactlyOne>
</Policy>
```

The client application developer's organization requires the use of addressing and transport-level security for any interaction with Company-X's Web services. The developer represents these behaviors using a policy expression illustrated in the example below in normal form. This policy expression contains one policy alternative that requires the use of addressing and transport-level security.

Example 3-7. The Client Application's Policy Expression in Normal Form

```
<Policy>
  <ExactlyOne>
    <All> <!-- - - - - - - - - - - - - - -  Client's Policy Alternative -->
       <!-- - - - - - - - - - - - - - - - - - - Policy Assertion (t1) -->
     <sp:TransportBinding>…</sp:TransportBinding>
       <!-- - - - - - - - - - - - - - - - - - - Policy Assertion (t2) -->
      <wsam:Addressing>…</wsam:Addressing>
    </All>
  </ExactlyOne>
</Policy>
```

The developer lets her policy-aware client select a compatible policy alternative in Company-X's policy. How does this client select a compatible policy alternative? It is simple – it uses the policy intersection. That is, the policy-aware client uses these two policy expressions (the client's and Company-X's) and the policy intersection to select a compatible policy alternative for this interaction. Let us look at the details of policy intersection.

For two policy assertions to be compatible they must have the same QName. And, if either assertion has a nested policy, both assertions must have a nested policy and the nested policies must be compatible. For example, policy assertions (c2) and (t1) have the same QName, `sp:TransportBinding`. For this discussion, let us assume that these two assertions have compatible nested policies. These two assertions are compatible because they have the same QName and their nested policies are compatible.

Two policy alternatives are compatible if each policy assertion in one alternative is compatible with a policy assertion in the other and vice-versa. For example, policy assertions (c1) and (c2) in Company-X's policy alternative are compatible with policy assertions (t2) and (t1) in tje client's policy alternative. Company-X's policy alternative (a) and the client's policy alternative are compatible because assertions in these two alternatives are compatible.

Two policies are compatible if a policy alternative in one is compatible with a policy alternative in the other. For example, Company-X's policy alternative (a) is compatible with the client's policy alternative. Company-X's policy and the client's policy are compatible because one of Company-X's policy alternative is compatible with the client's policy alternative.

For this interaction, the developer's policy-aware client can use policy alternative (a) to satisfy Company-X's conditions or requirements.

Similarly, policy intersection can be used to check if providers expose endpoints that conform to a standard policy. For example, a major retailer might require all their supplier endpoints to be compatible with an agreed upon policy.

3.4.1 Strict and Lax Policy Intersection

The previous sections outlined how the normal-form of a policy expression relate to the policy data model and how the compatibility of requester and provider policies may be determined. This section outlines how ignorable assertions may impact the process of determining compatibility.

In order to determine compatibility of its policy expression with a provider policy expression, a requester may use either a "lax" or "strict" mode of the intersection algorithm.

In the strict intersection mode two policy alternatives are compatible when each assertion in one is compatible with an assertion in the other, and vice versa. For this to be possible they must share the same policy alternative vocabulary. The strict intersection mode is the mode of intersection discussed in the previous sections of this document.

When using the strict intersection mode all assertions are part of the policy alternative vocabulary, including those marked with `wsp:Ignorable`. Thus the `wsp:Ignorable` attribute does not impact the intersection result even when its attribute value is "true".

If a requester wishes to ignore ignorable assertions in a provider's policy, then the requester should use the lax intersection mode. In the lax intersection mode all ignorable assertions (i.e. with the value "true" for the `wsp:Ignorable` attribute) are to be ignored by the intersection algorithm. Thus in the lax intersection mode two policy alternatives are compatible when each non-ignorable assertion in one is compatible with an assertion in the other, and vice versa. For this to be possible the two policy alternatives must share a policy alternative vocabulary for all "non-ignorable" assertions.

Regardless of the chosen intersection mode, ignorable assertions do not express any wire-level requirements on the behavior of consumers - in other words, a consumer could choose to ignore any such assertions that end up in the resulting policy after intersection, with no adverse effects on runtime interactions.

Domain-specific processing could take advantage of any information from the policy data model, such as the ignorable property of a policy assertion.

A requester can decide how to process a provider's policy to determine if and how the requester will interact with the provider. The requester can have its own policy that expresses its own capabilities and requirements, and can make one or more attempts at policy intersection in order to determine a compatible alternative and/or isolate the cause of an empty intersection result. The requester can use and analyze the result(s) of policy intersection to select a compatible alternative or trigger other domain-specific processing options. For example, a requester can at first attempt strict mode intersection, and then lax mode as another choice, if the previous attempt returns an empty intersection result.

3.5 Attaching Policy Expressions to WSDL

In **2. Basic Concepts: Policy Expression** [p.4] , we looked into how Company-X attached their policy expressions to the WSDL `binding` element. In addition to the WSDL `binding` element, a policy expression can be attached to other WSDL elements such as `service`, `port`, `operation` and `message`. These elements are the WSDL policy attachment points in a WSDL document.

The WSDL attachment points are partitioned (as illustrated below) into four policy subjects: message, operation, endpoint and service. When attached, capabilities and requirements represented by a policy expression apply to a message exchange or message associated with (or described by) a policy subject.

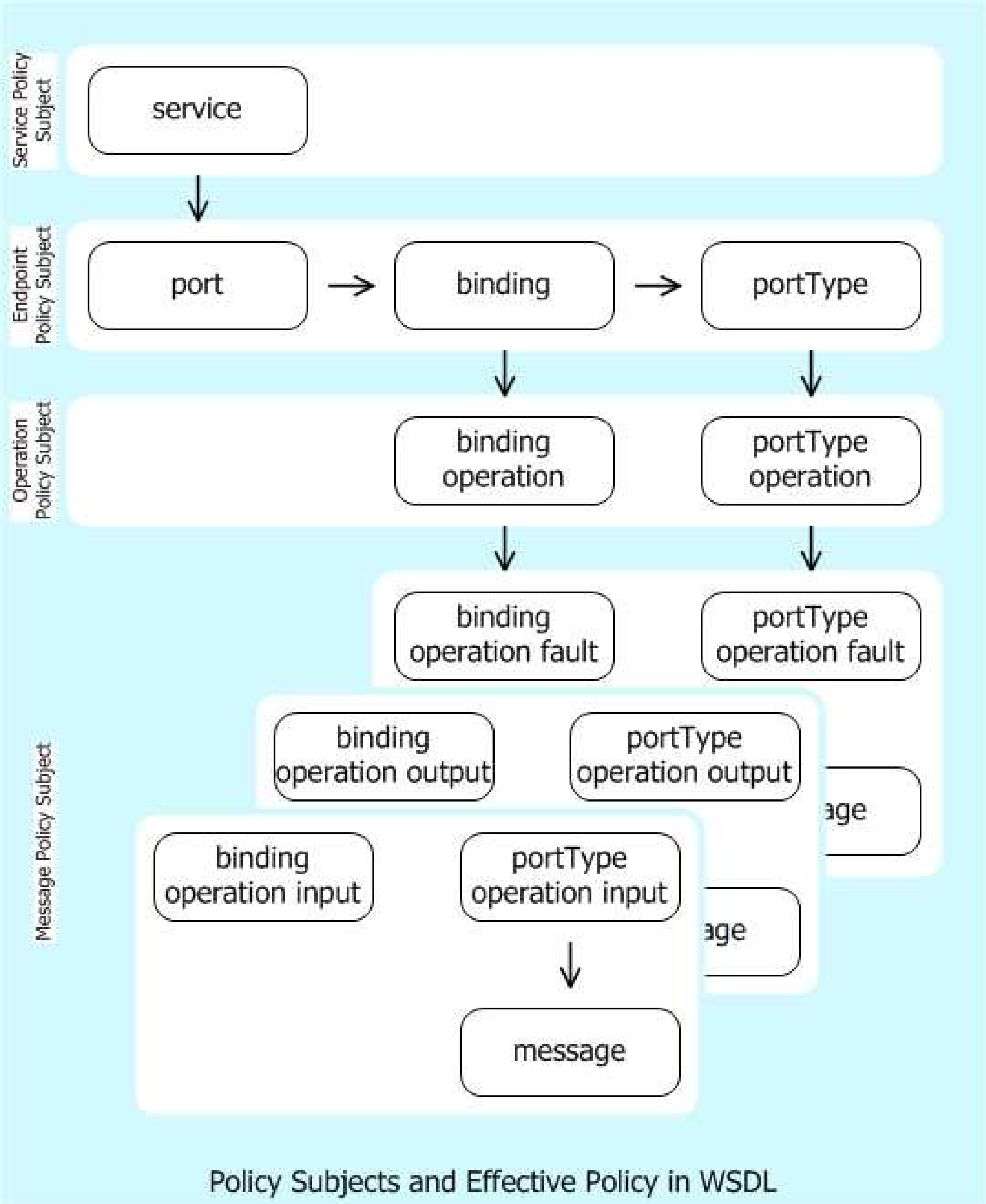


Figure 3-3. Policy Subjects and Effective Policy in WSDL

The WSDL `service` element represents the service policy subject. Policy expressions associated with a service policy subject apply to any message exchange using any of the endpoints offered by that service.

The WSDL `port`, `binding` and `portType` elements collectively represent the endpoint policy subject. Policy expressions associated with an endpoint policy subject apply to any message exchange made using that endpoint.

The WSDL `binding/operation` and `portType/operation` elements collectively represent the operation policy subject. Policy expressions associated with an operation policy subject apply to the message exchange defined by that operation.

The WSDL `binding/operation/input`, `portType/operation/input`, and `message` element collectively represent the message policy subject for the input message. The WSDL `binding/operation/output`, `portType/operation/output`, and `message` element collectively represent the message policy subject for the output message. The WSDL `binding/operation/fault`, `portType/operation/fault`, and `message` element collectively represent the message policy subject for the fault message. Policy expressions associated with a message policy subject apply only to that message.

In the example below, the policy expression is attached to an endpoint policy subject.

Example 3-8. Company-X's Policy Expression Attached to WSDL binding Element

```
<wsdl:binding name="SecureBinding" type="tns:RealTimeDataInterface" >
  <PolicyReference URI="#secure" />
  <wsdl:operation name="GetRealQuote">…</wsdl:operation>
  …
</wsdl:binding>
```

If multiple policy expressions are attached to WSDL elements that collectively represent a policy subject then the effective policy of these policy expressions applies. The effective policy is the combination of the policy expressions that are attached to the same policy subject. For example, the effective policy of an endpoint policy subject is the combination of policy expressions attached to a WSDL `port` element, policy expressions attached to the `binding` element referenced by this port, and policy expressions attached to the `portType` element that is supported by this port. Let us consider how to combine policy expressions in the next section.

Most of the policy assertions are designated for the endpoint, operation or message policy subject. The commonly used WSDL attachment points are:

Policy Subject	Commonly used attachment point (s)
Endpoint	`binding` element
Operation	`binding/operation` element
Message	`binding/operation/input` and `binding/operation/output` elements

3.6 Policy Retrieval

Just as other service metadata languages, Web Services Policy does not mandate any specific policy retrieval mechanism. Any combination of any retrieval mechanisms in any order may be used for referencing policy expressions. Example retrieval mechanisms are:

- Do nothing. A policy expression with the referenced IRI is already known to be available in a local cache or chip (embedded systems).
- Use the referenced IRI and retrieve an existing policy expression from the containing XML document: a policy element with an XML ID.
- Use the referenced IRI and retrieve a policy expression from some policy repository (local or remote) or catalog. Policy tools may use any protocols (say Web Services Metadata Exchange) for such metadata retrieval. These protocols may require additional out of band information.
- Attempt to resolve the referenced IRI on the Web. This may resolve to a policy element or a resource that contains a policy element.

If the referenced policy expression is in the same XML document as the reference, then the policy expression should be identified using the `wsu:Id`|`xml:id` (XML ID) attribute and referenced using an IRI reference to this XML ID value.

WSDL 1.1 [*WSDL 1.1 [p.44]*] section 2.1 and WSDL 2.0 [*WSDL 2.0 Core Language [p.44]*] chapter 4 allow to import or include WSDL documents into another WSDL document with the wsdl11:import, wsdl20:import, and wsdl20:include statements. The importing and imported WSDL documents constitute separate XML documents each. If e.g. the importing WSDL document references a policy in the imported WSDL document, the rules for policy references between separate XML documents apply as described in **2.10 Referencing Policy Expressions** [p.13] .

3.7 Combine Policies

Multiple policy expressions may be attached to WSDL constructs. Let us consider how Company-X could have used multiple policy expressions in a WSDL document. In the example below, there are two policy expressions `#common2` and `#secure2` attached to the `SecureBinding` WSDL binding and `RealTimeDataPort` WSDL port descriptions.

Example 3-9. Multiple Policy Expressions Attached to Endpoint Policy Subject

```
<Policy wsu:Id=”common2”>
  <mtom:OptimizedMimeSerialization wsp:Optional="true"/>
  <wsam:Addressing>…</wsam:Addressing>
</Policy>
<Policy wsu:Id=”secure2”>
  <ExactlyOne>
   <sp:TransportBinding>…</sp:TransportBinding>
   <sp:AsymmetricBinding>…</sp:AsymmetricBinding >
  </ExactlyOne>
</Policy>
<wsdl:binding name="SecureBinding" type="tns:RealTimeDataInterface" >
```

```
  <PolicyReference URI="#secure2" />
  <wsdl:operation name="GetRealQuote">…</wsdl:operation>
  …
</wsdl:binding>
<wsdl:service name=”RealTimeDataService”>
  <wsdl:port name=”RealTimeDataPort” binding=”tns:SecureBinding”>
    <PolicyReference URI="#common2"/>
    …
  </wsdl:port>
</wsdl:service>
```

As we discussed before, the WSDL `port`, `binding` and `portType` elements collectively represent the endpoint policy subject. In the example above, the `#common2` and `#secure2` policy expressions attached to the `SecureBinding` WSDL binding and `RealTimeDataPort` WSDL port descriptions collectively apply to any message exchange associated with the `RealTimeDataPort` WSDL port.

As in the example above, multiple policy expressions may be attached to Web service constructs that collectively represent a single policy subject. When there are multiple policy expressions attached to the same policy subject then the effective policy or combination of these policy expressions apply to the associated policy subject.

The effective policy is the combination of two or more policy expressions attached to the same policy subject. The combination of two policy expressions, also known as the merged policy expression, is a new policy expression that combines these two policy expressions using the `All` policy operator.

The policy expression below is the combination of the two policy expressions attached to the `SecureBinding` WSDL binding and `RealTimeDataPort` WSDL port descriptions. The `#common2` policy expression has two policy alternatives. The `#secure2` policy expression has two policy alternatives. The combination of these two policies is equivalent to Company-X's secure policy in **2. Basic Concepts: Policy Expression** [p.4] and has four policy alternatives. In other words, the combination of two policies is the cross product of alternatives in these two policies.

Example 3-10. Effective Policy of the Endpoint Policy Subject in the Previous Example

```
<Policy>
  <All>
    <Policy>
     <mtom:OptimizedMimeSerialization wsp:Optional="true"/>
      <wsam:Addressing>…</wsam:Addressing>
    </Policy>
    <Policy>
      <ExactlyOne>
       <sp:TransportBinding>…</sp:TransportBinding>
       <sp:AsymmetricBinding>…</sp:AsymmetricBinding >
      </ExactlyOne>
    </Policy>
  </All>
</Policy>
```

Of course, the above policy expression can be normalized. There are four policy alternatives in the normal form. As we have seen in the policy data model, a policy is an unordered collection of policy alternatives. That is, the order of policy alternatives is insignificant. Therefore, the order of combining these policy expressions is insignificant.

3.8 Extensibility and Versioning

3.8.1 Policy Language

Web Services Policy language is an extensible language by design. The `Policy`, `ExactlyOne`, `All` and `wsp:PolicyReference` elements are extensible. The `Policy` element allows child element and attribute extensibility, while the `ExactlyOne` and `All` elements allow child element extensibility. The `PolicyReference` child element allows element and attribute extensibility. Extensions must not use the policy language XML namespace name. A consuming processor processes known attributes and elements, ignores unknown attributes and treats unknown children of the `Policy`, `ExactlyOne`, `All` elements as policy assertions. The child elements of `wsp:PolicyReference` are ignored.

The `PolicyReference` element allows element and attribute extensibility.

3.8.2 Policy Expressions

Services that use the Web Services Policy language for policy expression enable simple versioning practices that allow requesters to continue the use of older policy alternatives in a backward compatible manner. This versioning practice allows service providers, like Company-X, to deploy new behaviors using additional (or new) policy assertions without breaking compatibility with clients that rely on any older policy alternatives. We use examples below to illustrate how versioning might be done.

The example below represents a Company-X version 1 policy expression. This expression requires the use of addressing and transport-level security for protecting messages.

Example 3-11. Company-X's Version 1 Policy Expression

```
<Policy>
  <ExactlyOne>
    <All>
      <wsam:Addressing>…</wsam:Addressing>
     <sp:TransportBinding>…</sp:TransportBinding>
    </All>
  </ExactlyOne>
</Policy>
```

Over time, Company-X adds support for advanced behaviors: requiring the use of addressing and message-level security for protecting messages. They would like to add this advanced support without breaking compatibility with requesters that rely on addressing and transport-level security. The example below is Company-X's version 2 policy expression. In this version, Company-X adds a new policy alternative that requires the use of addressing and message-level security. The clients that rely on addressing and transport-level security may continue to interact with Company-X's using the old policy alternative. Of course, these clients have the option to migrate from using old policy alternatives to new policy alternatives.

Example 3-12. Company-X's Version 2 Policy Expression

```
<Policy>
  <ExactlyOne>
    <All>
      <wsam:Addressing>…</wsam:Addressing>
     <sp:TransportBinding>…</sp:TransportBinding>
    </All>
    <All> <!-- - - - - - - - - - - - - - - - NEW Policy Alternative -->
      <wsam:Addressing>…</wsam:Addressing>
      <sp:AsymmetricBinding>…</sp: AsymmetricBinding >
    </All>
  </ExactlyOne>
</Policy>
```

When Company-X added support for advanced behaviors, they spent time to plan for the continued support for existing clients, the smooth migration from using current to advanced behaviors, and the switch to use only the advanced behaviors in the near future (i.e. sun-setting current behaviors). In this versioning scenario, a policy expression with multiple alternatives was used to represent current and advanced behaviors in a non-disruptive manner: no immediate changes to existing clients are required and these clients can smoothly migrate to new functionality when they choose to. This level of versioning support in a policy expression enables the same class of versioning best practices built into WSDL constructs such as service, port and binding.

Let us look at tooling for unknown policy assertions. As service providers, like Company-X, incrementally deploy advanced behaviors, some requesters may not recognize these new policy assertions. As discussed before, these requesters may continue to interact using old policy alternatives. New policy assertions will emerge to represent new behaviors and slowly become part of everyday interoperable interaction between requesters and providers. For example, most tools use a practical tolerant strategy to process new or unrecognized policy assertions. These tools consume such unrecognized assertions and designate these for user intervention. As you would recognize, there is nothing new in this practice. This is similar to how a proxy generator that generates code from WSDL creates code for all the known WSDL constructs and allows Web service developers to fill in code for custom or unknown constructs in the WSDL.

3.8.3 Use of Ignorable attribute and an alternative Versioning Scenario

One potential use of the wsp:Ignorable attribute is to mark versioning related information by creating a new policy assertion within a policy expression. The new assertion is added to the original policy expression and then the service can update the assertion parameter values when the service expires.

One scenario that illustrates this is a service which will support a particular version of a service until a certain point in time. After that time, the service will not be supported. In this scenario, the expiry date and time of the service would be a new policy assertion [see Guidelines section 4] that the service provider defines . This hypothetical EndOfLife policy assertion is then included in the original policy expression, but it could be marked as ignorable. The service, in this case, wants to inform the consumers it does have an expiry time, and so it is useful to convey this information from the beginning to help smooth the versioning process.

Company-X could specify that one policy alternative will expire at a certain point in time using the hypothetical ignorable Company-X expiry assertion. The example below shows how Company-X can create a new version 2 policy expression with a second hypothetical ignorable EndOfLife Assertion with a different date and time.

Example 3-13. Company-X's Version 2 Policy Expression with hypothetical ignorable EndOfLife Assertion

```
<Policy>
  <ExactlyOne>
    <All>
      <company-x:EndOfLife wsp:Ignorable="true"/>Mar-31-2008</company-x:EndOfLife>
      <wsam:Addressing>…</wsam:Addressing>
      <sp:TransportBinding>…</sp:TransportBinding>
    </All>

    <!-- NEW Policy Alternative -->
    <All>
      <company-x:EndOfLife wsp:Ignorable="true">Mar-31-2999</company-x:EndOfLife>
        <wsam:Addressing>…</wsam:Addressing>
        <sp:AsymmetricBinding>…</sp:AsymmetricBinding>
      </All>
  </ExactlyOne>
</Policy>
```

In this variant of the versioning scenario, the use of ignorable allows versioning related information to be conveyed and used where understood.

In a scenario such as this, CompanyX is acting as both a policy assertion author and a policy expression author. As a policy expression author, when an assertion type is tagged as ignorable information, the use of strict or lax mode and presence or absence of the assertion type in the first version are important decisions.

3.8.4 Use of Ignorable and Optional attributes

If Company-X knows about the hypothetical EndOfLife Policy assertion, it may or may not mark that assertion with wsp:Optional="true" in the first version. If it does include the assertion, marks the assertion with wsp:Ignorable="true" and wsp:Optional="false", then a client that:

- does not know about the assertion and using lax intersection will produce an intersection.
- does not know about the assertion and using strict intersection will not produce an intersection.
- does know about the assertion and using strict or lax intersection will produce an intersection.

If it does include the assertion, marks the assertion with wsp:Ignorable="true" and wsp:Optional="true", then a client that:

- does or does not know about the assertion and using lax or strict intersection will produce an intersection.

The following table summarizes the requester assertion knowledge and intersection mode on the left vs provider ignorable and optional on the top

Requester \ Provider	Required	Required and Ignorable (for intersection)	Optional	Optional and Ignorable (for intersection)
does not know, lax	No	Yes	Yes	Yes
does not know, strict	No	No	Yes	Yes
does know, lax	Yes	Yes	Yes	Yes
does know, strict	Yes	Yes	Yes	Yes

If Company-X adds the hypothetical EndOfLife policy assertion type to a subsequent Alternative and does not mark the assertion with wsp:Optional="true", then after the policy expression has been deployed/used the same algorithm holds true, notably that a client using strict mode that does not understand the assertion will not intersect with the alternative. If CompanyX adds the hypothetical EndOfLife policy assertion with an ignorable attribute and does mark the assertion with wsp:Optional="true", then clients using strict mode who do not understand the hypothetical EndOfLife assertion with the ignorable information will still be compatible with the alternative that does not contain the hypothetical EndOfLife policy assertion as per the intersection rules. When wsp:Ignorable="true" is used, clients that are unaware of the hypothetical EndOfLife assertion may make more requests for expired services. This could result in servers generating Faults if the request is received after the expiry date. .

If Company-X knows about the hypothetical EndOfLife Policy assertion, it can guarantee that clients that know or don’t know about the hypothetical EndOfLife Policy Assertion can intersect under any mode by marking the assertion with wsp:Optional="true". Clients that know about the hypothetical EndOfLife Policy assertion and performing strict intersection can guarantee interaction with services that know or don’t know about the hypothetical EndOfLife Policy assertion by marking the assertion with wsp:Optional="true". Clients that know about the hypothetical EndOfLife Policy assertion and performing lax intersection can guarantee interaction with services that know or don’t know about the hypothetical EndOfLife Policy assertion by marking the assertion with wsp:Optional="true" or marking it with wsp:Ignorable="true".

Because the actual value of the date/time may not be known when the policy expression is first created, a value that is roughly infinitely in the future is used. A subsequent policy alternative could refine the value and domain specific processing of the assertion can differentiate the value. The advantage of adding the end of life information through a domain specific assertion is that some clients will have a machine processable way of knowing when the alternative will no longer be supported by evaluating the policy assertions in a policy expression. Without this information in a policy expression, the information must be conveyed in some other way or it will not be conveyed at all. This can usefully smooth the transition between versions of a service.

The disadvantage of adding the end of life information through a domain specific assertion is that clients need to understand the semantics of the hypothetical EndOfLife assertion in order to know whether a particular alternative is still valid. For example, a client that doesn't know what the parameter "Mar-31-2008" means, will not know that the service is no longer available on April 1, and may send messages to this service in April, and if the service enforces "end of life", these messages may fail.

3.9 Parts of a Policy Assertion

As we discussed, a policy assertion identifies a domain specific behavior or requirement or condition. A policy assertion has a QName that identifies its behavior or requirement or condition. A policy assertion may contain assertion parameters and a nested policy.

Let us look at the anatomy of a policy assertion from the security domain. The policy expression in the diagram below uses the `sp:IssuedToken` policy assertion. This assertion illustrates the use of assertion parameters and nested policy.

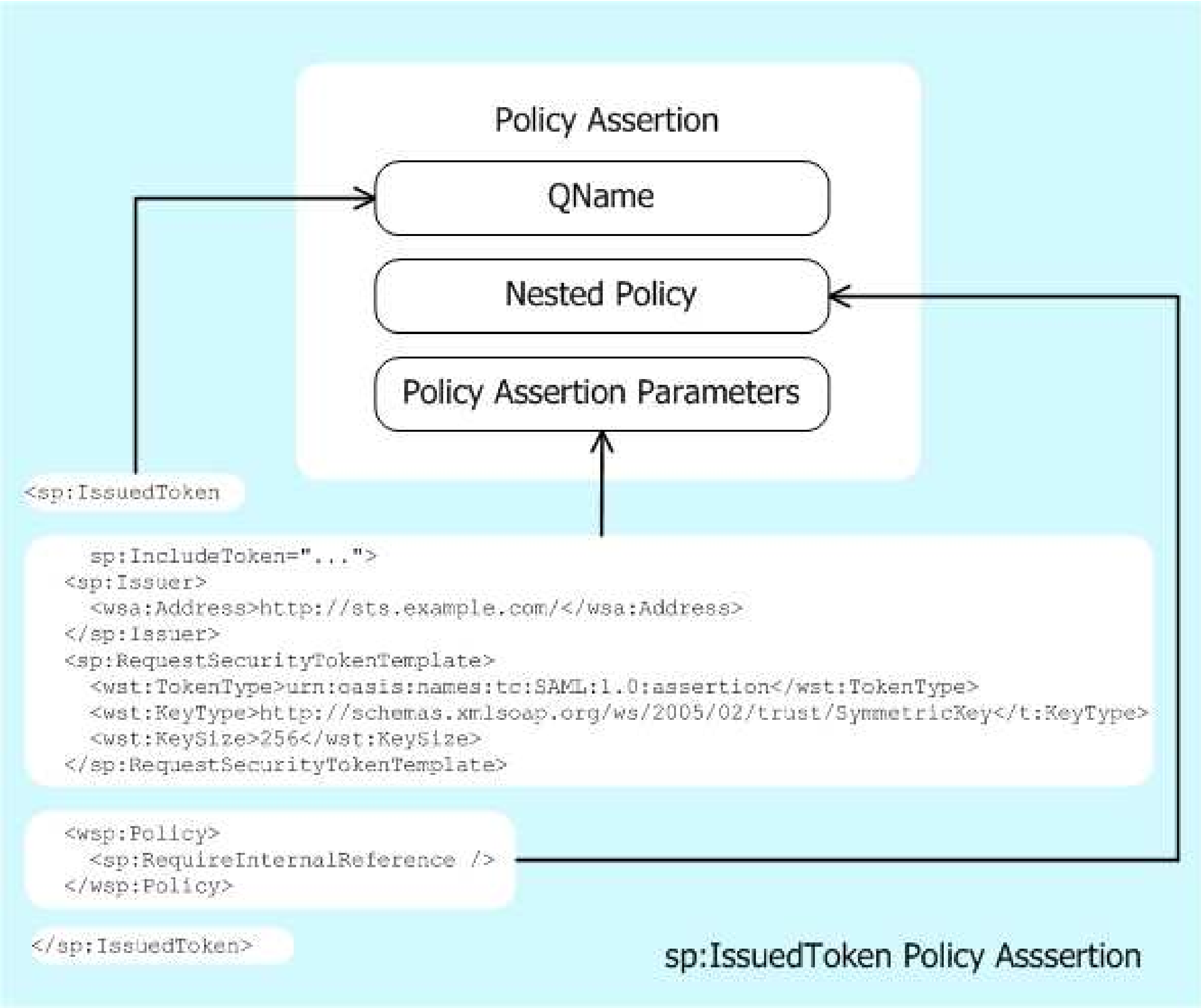


Figure 3-4. sp:IssuedToken Policy Assertion

The `sp:IssuedToken` element is a policy assertion that identifies the use of a security token – such as SAML token - issued by a third party for protecting messages. A policy assertion is an XML element. The QName of this element represents the behavior identified by this policy assertion.

The `sp:IssuedToken` policy assertion has three parameters: `@sp:IncludeToken`, `sp:Issuer` and `sp:RequestSecurityTokenTemplate`.

The `sp:IncludeToken` attribute is a parameter that contains information on whether a security token should be included in messages or an external reference to the key of this security token should be used. The `sp:Issuer` parameter is an endpoint reference to a security token issuer. The `sp:RequestSecurityTokenTemplate` parameter contains the necessary information to request a security token from the specified issuer. Parameters are the opaque payload of a Policy Assertion, carry useful information for engaging the behavior described by an assertion and are preserved through policy processing such as normalize, merge and intersection. requesters may use policy intersection to select a compatible policy alternative for an interaction. Assertion parameters do not affect the outcome of policy intersection.

For the `sp:Issuer` policy assertion parameter, the assertion author uses the natural XML structural relationships (the child elements and attributes) and encodes the relationship between an assertion and its parameters in a machine readable form. Assertion parameters may be represented as child XML elements or attributes of an assertion. The policy language allows assertion authors to strongly tie the relationship between an assertion and its parameters using the natural XML structural relationships.

The `sp:IssuedToken` policy assertion has a nested policy expression. The `sp:RequireInternalReference` element is a nested policy assertion of the `sp:IssuedToken` policy assertion. The `sp:RequireInternalReference` assertion requires the use of an internal reference for referencing the issued token. A nested policy assertion further qualifies a dependent behavior of its parent policy assertion. As mentioned earlier, requesters may use policy intersection to select a compatible policy alternative for an interaction. Nested policy assertions affect the outcome of policy intersection.

The `sp:IssuedToken` security policy assertion identifies a visible domain specific behavior: the use of a security token – such as SAML token - issued by a third party for protecting messages. This behavior is relevant to a Web service interaction. For the sake of discussion, let us assume that Company-X requires the use of a SAML token issued by a third party. Service providers, like Company-X, must convey this usage and all the necessary information to obtain this security token for Web service developers. This is a key piece of metadata for a successful interaction with Company-X's Web services.

4. Versioning Policy Language

Editorial note	
The WG is contemplating moving some or all of this material into a non-normative appendix of the framework or attachment document. User feedback is solicited	

Over time, the Policy WG or third parties can version or extend the Policy Language with new or modified constructs. These constructs may be compatible or incompatible with previous versions. Some of the possible new constructs that have been mentioned previously are: new operators, operator cardinality, policy identification, compact syntax, Policy Inclusion, security, referencing, attachment points, alternative priority, effective dating, negotiation.

WS-Policy provides extensibility points on 6 elements with a combination of attribute and/or element extensibility. The possible extensibility points are:

1. Policy: element from ##other namespace and any attribute
2. PolicyReference: any attribute and any element
3. ExactlyOne, All: element from ##other namespace, no attribute extensibility
4. PolicyAttachment: element from ##other namespace and any attribute
5. AppliesTo: any element and any attribute

4.1 Policy Framework

WS-Policy Framework 1.5 specifies that any child element that is not known inside a Policy, ExactlyOne or All will be treated as an assertion. The default value for wsp:Optional="false". After normalization, such an element will be inside an ExactlyOne/All operator.

Let us show an example with a hypothetical new operator that is a Choice with a minOccurs and a maxOccurs attributes, ala XSD:Choice, in a new namespace. We use the wsp16 prefix to indicate a hypothetical Policy Language 1.6 that is intended to be compatible with Policy Language 1.5:

Example 4-1. Policy containing 1.5 and 1.6 Policies.

```
<wsp:Policy>
  <wsp:ExactlyOne>
    <wsp16:Choice wsp16:minOccurs="1" wsp16:maxOccurs="2">
      ...
    </wsp16:Choice>
    <wsp:All>
       ...
    </wsp:All>
  </wsp:ExactlyOne>
</wsp:Policy>
```

The normalization rule for wsp:Optional="false" would be applied to the wsp16:Choice, yielding the following expression:

Example 4-2. Normalized Policy containing 1.5 and 1.6 Policies

```
<wsp:Policy>
  <wsp:ExactlyOne>
    <wsp:ExactlyOne>
      <wsp:All>
```

```
         <wsp16:Choice wsp16:minOccurs="1" wsp16:maxOccurs="2">
          ...
        </wsp16:Choice>
      </wsp:All>
    </wsp:ExactlyOne>
    <wsp:All>
       ...
    </wsp:All>
  </wsp:ExactlyOne>
</wsp:Policy>
```

Alternatively, the wsp:Optional could be set to "true" on the choice, as in:

Example 4-3. Policy containing explicit wsp:Optional="true"

```
<wsp:Policy>
  <wsp16:Choice wsp16:minOccurs="1" wsp16:maxOccurs="2"
wsp:Optional="true">
      ...
  </wsp16:Choice>
</wsp:Policy>
```

The normalized form will be:

Example 4-4. Normalized policy

```
<wsp:Policy>
  <wsp:ExactlyOne>
     <wsp:All>
         <wsp16:Choice wsp16:minOccurs="1" wsp16:maxOccurs="2">
          ...
        </wsp16:Choice>
      </wsp:All>
     <wsp:All/>
  </wsp:ExactlyOne>
</wsp:Policy>
```

Because the wsp16:Choice alternative isn't understood in either normalized form, it will not be chosen as one of the alternatives and will effectively be ignored. Policy intersection may be more difficult with such compatible extensions. For example, the previous will "look" like it has a wsp16:Choice typed assertion. To determine intersection with a Policy that does not have the wsp16:Choice type assertion, domain specific processing would have to be done. However, there is an alternative that does not have the wsp16:Choice, so intersection would yield the expected result.

Note: it is possible to add new names to the existing namespace, such as:

Example 4-5. Policy containing 1.5 and 1.6 Policies all in the 1.5 namespace

```
<wsp:Policy>
  <wsp:ExactlyOne>
    <wsp:Choice wsp:minOccurs="1" wsp:maxOccurs="2">
      ...
    </wsp:Choice>
    <wsp:All>
       ...
    </wsp:All>
  </wsp:ExactlyOne>
</wsp:Policy>
```

Notice that using a new namespace can result in backwards and forwards compatibility if normalization results in an optional alternative.

Best practice: insert new elements in an optional alternative or mark with wsp:Optional="true".

Incompatible versions of the Policy language may be indicated by a new namespace name for at least the new and/or incompatible elements or attributes. Imagine that the Choice operator is required by a future version of Policy, then there will be a new namespace for the Policy element. We use the wsp20 prefix to indicate a hypothetical Policy Language 2.0 that is intended to be incompatible with Policy Language 1.5:

Example 4-6. Policy containing 2.0 only Policies.

```
<wsp20:Policy>
  <wsp20:ExactlyOne>
    <wsp20:Choice wsp:minOccurs="1" wsp:maxOccurs="2">
      ...
    </wsp20:Choice>
    ...
  </wsp20:ExactlyOne>
</wsp20:Policy>
```

The new Policy operator could be embedded inside an existing Policy element:

Example 4-7. Policy containing 2.0 (incompatible with 1.5) Policies embedded in wsp 1.5 Policy.

```
<wsp:Policy>
    <wsp20:Choice wsp:minOccurs="1" wsp:maxOccurs="2">
      ...
    </wsp20:Choice>
    ...
</wsp20:Policy>
```

This will be treated as an Assertion for normalization and intersection computation. This will result in only one alternative that requires the wsp20:Choice, the intended behaviour for incompatible changes.

Best practice: use a new namespace for new incompatible construct and insert inside either: new Policy element OR existing All for future incompatible policy extensions.

A future version of WS-Policy could support the current operators in the existing namespace, such as:

Example 4-8. Policy containing 1.5 operator in 2.0 Policy

```
<wsp20:Policy>
  <wsp:ExactlyOne>
    <wsp20:Choice wsp:minOccurs="1" wsp:maxOccurs="2">
      ...
    </wsp20:Choice>
    ...
  </wsp:ExactlyOne>
</wsp20:Policy>
```

It is difficult to predict whether this functionality would be useful. The future version of WS-Policy doesn't appear to be precluded from doing this.

4.2 Policy Attachment

Policy attachment provides WSDL 1.1 and UDDI attachment points. It appears that exchange of Policy will be in the context of WSDL or UDDI. WRT WSDL, the policy model is an extension of the WSDL definition. As such, it is likely that future versions of Policy will be exchanged as multiple Policy expressions within a WSDL. One alternative is that there would be a separate WSDL for each version of Policy. The problem of how to specify and query for compound documents is very difficult, so it is more likely that each version of Policy will be exchanged within a WSDL.

We show an example of a new version of policy that allows QName reference to Policies in the PolicyReference:

Example 4-9. WSDL containing 1.5 and 2.0 (compatible with 2.0) Policy References.

```
<wsdl11:binding name="StockQuoteSoapBinding" type="fab:Quote" >
       <wsoap12:binding style="document"
          transport="http://schemas.xmlsoap.org/soap/http" />
        <wsp:Policy>
          <wsp:ExactlyOne>
                <wsp:All>
                <wsp:PolicyReference URI="#RmPolicy"
wsdl11:required="true" />
              <wsp:PolicyReference URI="#X509EndpointPolicy"
wsdl11:required="true" />
                </wsp:All>
                <wsp:All>
                <wsp:PolicyReferenceByQName ref="rmp:RMAssertion"
wsdl11:required="true" />
              <wsp:PolicyReferenceByQName ref="sp:AsymmetricBinding"
wsdl11:required="true" />
                </wsp:All>
         </wsp:ExactlyOne>
        </wsp:Policy>
  <wsdl11:operation name="GetLastTradePrice" > ....
  ...
```

The PolicyReference element is element or attribute extensible. One example of an addition is a list of backup URIs for the PolicyReference:

Example 4-10. WSDL containing 1.5 and 2.0 (compatible with 2.0) Policy References.

```
<wsdl11:binding name="StockQuoteSoapBinding" type="fab:Quote" >
       <wsoap12:binding style="document"
          transport="http://schemas.xmlsoap.org/soap/http" />
        <wsp:Policy>
          <wsp:ExactlyOne>
                <wsp:All>
                <wsp:PolicyReference URI="" wsp16:alternateURIs="URI*"
wsdl11:required="true" />
              <wsp:PolicyReference URI="" wsp16:alternateURIs="URI*"
wsdl11:required="true" />
                </wsp:All>
         </wsp:ExactlyOne>
        </wsp:Policy>
  <wsdl11:operation name="GetLastTradePrice" > ....
  ...
```

The policy framework specification says that any unknown attributes are ignored. A Policy 1.5 processor will not understand the wsp16:alternateURI attribute, it will be ignored. A Policy 1.6 processor will understand the alternate URIs so it won't be ignored.

PolicyAttachment and AppliesTo also have extensibility points. We choose not to illustrate these at this time.

5. Conclusion

Service providers use Web Services Policy to represent combinations of behaviors (capabilities and requirements). Web service developers use policy-aware clients that understand policy expressions and engage the behaviors represented by providers automatically. These behaviors may include security, reliability, transaction, message optimization, etc. Web Services Policy is a simple language, hides complexity from developers, automates Web service interactions, and enables secure, reliable and transacted Web Services.

A. Security Considerations

Security considerations are discussed in the *Web Services Policy Framework* *[p.44]* document.

B. XML Namespaces

The table below lists XML Namespaces that are used in this document. The choice of any namespace prefix is arbitrary and not semantically significant.

Table B-1. Prefixes and XML Namespaces used in this specification.

Prefix	XML Namespace	Specifications
mtom	http://schemas.xmlsoap.org/ws/2004/09/policy/optimizedmimeserialization	[*WS-MTOMPolicy [p.43]*]
soap	http://www.w3.org/2003/05/soap-envelope	[*SOAP 1.2 Messaging Framework [p.43]*]
sp	http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702	[*WS-SecurityPolicy [p.44]*]
wsa	http://www.w3.org/2005/08/addressing	[*WS-Addressing Core [p.43]*]
wsam	http://www.w3.org/2007/05/addressing/metadata	[*WS-Addressing Metadata [p.43]*]
wsdl	http://schemas.xmlsoap.org/wsdl/	[*WSDL 1.1 [p.44]*]
wsp	http://www.w3.org/ns/ws-policy	[*Web Services Policy Framework [p.44]* , *Web Services Policy Attachment [p.44]*]
wss	http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd	[*WS-Security 2004 [p.44]*]
wst	http://schemas.xmlsoap.org/ws/2005/02/trust	[*WS-Trust [p.44]*]
wsu	http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd	[*WS-Security 2004 [p.44]*]

C. References

[C14N 1.0 Note]

Known Issues with Canonical XML 1.0 (C14N/1.0), J. Kahan and K. Lanz, Editors. World Wide Web Consortium, 17 August 2006. Available at http://www.w3.org/2006/04/c14n-note/c14n-note.html.>

[MTOM]
SOAP Message Transmission Optimization Mechanism, M. Gudgin, N. Mendelsohn, M. Nottingham and H. Ruellan, Editors. World Wide Web Consortium, 25 January 2005. This version of the SOAP Message Transmission Optimization Mechanism Recommendation is http://www.w3.org/TR/2005/REC-soap12-mtom-20050125/. The latest version of SOAP Message Transmission Optimization Mechanism is available at http://www.w3.org/TR/soap12-mtom/.

[WS-MTOMPolicy]
MTOM Serialization Policy Assertion (WS-MTOMPolicy), C. Ferris, et al, Authors. International Business Machines Corporation and Microsoft Corporation, Inc., September 2006. Available at http://schemas.xmlsoap.org/ws/2004/09/policy/optimizedmimeserialization/

[SOAP 1.1]
Simple Object Access Protocol (SOAP) 1.1, D. Box, et al, Editors. World Wide Web Consortium, 8 May 2000. Available at http://www.w3.org/TR/2000/NOTE-SOAP-20000508/.

[SOAP 1.2 Messaging Framework]
SOAP Version 1.2 Part 1: Messaging Framework, M. Gudgin, M. Hadley, N. Mendelsohn, J-J. Moreau, H. Frystyk Nielsen, Editors. World Wide Web Consortium, 24 June 2003. This version of the SOAP Version 1.2 Part 1: Messaging Framework Recommendation is http://www.w3.org/TR/2003/REC-soap12-part1-20030624/. The latest version of SOAP Version 1.2 Part 1: Messaging Framework is available at http://www.w3.org/TR/soap12-part1/.

[SecSpecMaintWG]
XML Security Specifications Maintenance Working Group , See http://www.w3.org/2007/xmlsec.

[WS-Addressing Core]
Web Services Addressing 1.0 - Core, M. Gudgin, M. Hadley, and T. Rogers, Editors. World Wide Web Consortium, 9 May 2006. This version of the Web Services Addressing 1.0 - Core Recommendation is http://www.w3.org/TR/2006/REC-ws-addr-core-20060509/. The latest version of Web Services Addressing 1.0 - Core is available at http://www.w3.org/TR/ws-addr-core.

[WS-Addressing Metadata]
Web Services Addressing 1.0 - Metadata, M. Gudgin, M. Hadley, T. Rogers and Ü. Yalçinalp, Editors. World Wide Web Consortium, 16 May 2007. This version of the Web Services Addressing 1.0 - Metadata is http://www.w3.org/TR/2007/WD-ws-addr-metadata-20070516/. The latest version of Web Services Addressing 1.0 - Metadata is available at http://www.w3.org/TR/ws-addr-metadata.

[Web Services Atomic Transaction]
Web Services Atomic Transaction, L. P. Cabrera, et al, Authors. Arjuna Technologies, Inc., BEA Systems, Inc., Hitachi Software, Inc., IONA Technologies, Inc., International Business Machines Corporation, and Microsoft Corporation, February 2005. Available at http://schemas.xmlsoap.org/ws/2004/10/wsat/.

[Web Services Business Activity Framework]
Web Services Business Activity Framework, L. P. Cabrera, et al, Authors. Arjuna Technologies, Inc., BEA Systems, Inc., Hitachi Software, Inc., IONA Technologies, Inc., International Business Machines Corporation, and Microsoft Corporation, February 2005. Available at http://schemas.xmlsoap.org/ws/2004/10/wsba/.

[Devices Profile for Web Services]
Devices Profile for Web Services, S. Chan, et al, Authors. Intel Corporation, Lexmark, Inc., Microsoft Corporation, and Richo Software, Inc., February 2006. Available at http://schemas.xmlsoap.org/ws/2006/02/devprof/.

[WS-MetadataExchange]
Web Services Metadata Exchange (WS-MetadataExchange), K. Ballinger, et al, Authors. BEA Systems Inc., Computer Associates International, Inc., International Business Machines Corporation, Microsoft Corporation, Inc., SAP AG, Sun Microsystems, and webMethods, August 2006. Available at http://schemas.xmlsoap.org/ws/2004/09/mex/

[Web Services Policy Framework]
Web Services Policy 1.5 - Framework, A. S. Vedamuthu, D. Orchard, F. Hirsch, M. Hondo, P. Yendluri, T. Boubez and Ü. Yalçinalp, Editors. World Wide Web Consortium, 05, June 2007. This version of the Web Services Policy 1.5 - Framework specification is at http://www.w3.org/TR/ws-policy/. The latest version of Web Services Policy 1.5 - Framework is available at http://www.w3.org/TR/ws-policy/.

[Web Services Policy Attachment]
Web Services Policy 1.5 - Attachment, A. S. Vedamuthu, D. Orchard, F. Hirsch, M. Hondo, P. Yendluri, T. Boubez and Ü. Yalçinalp, Editors. World Wide Web Consortium, 05, June 2007. This version of the Web Services Policy 1.5 - Attachment specification is at http://www.w3.org/TR/ws-policy-attach. The latest version of Web Services Policy 1.5 - Attachment is available at http://www.w3.org/TR/ws-policy-attach/.

[Web Services Reliable Messaging Policy]
Web Services Reliable Messaging Policy Assertion (WS-RM Policy), D. David, A. Kamarkar, G. Pilz, and Ü. Yalçinalp, Editors. Organization for the Advancement of Structured Information Standards, 24 April 2006. Available at http://docs.oasis-open.org/ws-rx/wsrmp/200608/wsrmp-1.1-rddl-200608.html

[WSDL 1.1]
Web Services Description Language (WSDL) 1.1, E. Christensen, et al, Authors. World Wide Web Consortium, March 2001. Available at http://www.w3.org/TR/2001/NOTE-wsdl-20010315.

[WSDL 2.0 Core Language]
Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language, R. Chinnici, J. J. Moreau, A. Ryman, S. Weerawarana, Editors. World Wide Web Consortium, 27 March 2006. This version of the WSDL 2.0 specification is http://www.w3.org/TR/2006/CR-wsdl20-20060327. The latest version of WSDL 2.0 is available at http://www.w3.org/TR/wsdl20.

[WS-Security 2004]
Web Services Security: SOAP Message Security 1.0, A. Nadalin, C. Kaler, P. Hallam-Baker and R. Monzillo, Editors. Organization for the Advancement of Structured Information Standards, March 2004. Available at http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf.

[WS-SecurityPolicy]
WS-SecurityPolicy v1.0, A. Nadalin, M. Gudgin, A. Barbir, and H. Granqvist, Editors. Organization for the Advancement of Structured Information Standards, 8 December 2005. Available at http://www.oasis-open.org/committees/download.php/15979/oasis-wssx-ws-securitypolicy-1.0.pdf.

[WS-Trust]
Web Services Trust Language (WS-Trust), S. Anderson, et al, Authors. Actional Corporation, BEA Systems, Inc., Computer Associates International, Inc., International Business Machines Corporation, Layer 7 Technologies, Microsoft Corporation, Oblix Inc., OpenNetwork Technologies Inc., Ping Identity Corporation, Reactivity Inc., RSA Security Inc., and VeriSign Inc., February 2005. Available at http://schemas.xmlsoap.org/ws/2005/02/trust.

[XML ID]
xml:id Version 1.0, J. Marsh, D. Veillard and N. Walsh, Editors. World Wide Web Consortium, 9 September 2005. This version of the xml:id Version 1.0 Recommendation is http://www.w3.org/TR/2005/REC-xml-id-20050909/. The latest version of xml:id Version 1.0 is available at http://www.w3.org/TR/xml-id/.

[XMLID11]
Canonical XML 1.1, This is a work in progress. J. Boyer and G. Marcy Authors. W3C Working Draft, 20 December 2006. Available at http://www.w3.org/TR/xml-c14n11/.

[XOP]
XML-binary Optimized Packaging, M. Gudgin, N. Mendelsohn, M. Nottingham and H. Ruellan, Editors. World Wide Web Consortium, 25 January 2005. This version of the XML-binary Optimized Packaging Recommendation is http://www.w3.org/TR/2005/REC-xop10-20050125/. The latest version of XML-binary Optimized Packaging is available at http://www.w3.org/TR/xop10/.

D. Acknowledgements (Non-Normative)

This document is the work of the W3C Web Services Policy Working Group.

Members of the Working Group are (at the time of writing, and by alphabetical order): Dimitar Angelov (SAP AG), Abbie Barbir (Nortel Networks), Charlton Barreto (Adobe Systems Inc.), Sergey Beryozkin (IONA Technologies, Inc.), Vladislav Bezrukov (SAP AG), Toufic Boubez (Layer 7 Technologies), Symon Chang (BEA Systems, Inc.), Paul Cotton (Microsoft Corporation), Glen Daniels (Sonic Software), Doug Davis (IBM Corporation), Jacques Durand (Fujitsu Limited), Ruchith Fernando (WSO2), Christopher Ferris (IBM Corporation), William Henry (IONA Technologies, Inc.), Frederick Hirsch (Nokia), Maryann Hondo (IBM Corporation), Ondrej Hrebicek (Microsoft Corporation), Steve Jones (Layer 7 Technologies), Tom Jordahl (Adobe Systems Inc.), Paul Knight (Nortel Networks), Philippe Le Hégaret (W3C/MIT), Mark Little (JBoss Inc.), Mohammad Makarechian (Microsoft Corporation), Ashok Malhotra (Oracle Corporation), Jonathan Marsh (WSO2), Monica Martin (Sun Microsystems, Inc.), Arnaud Meyniel (Axway Software), Jeff Mischkinsky (Oracle Corporation), Dale Moberg (Axway Software), Anthony Nadalin (IBM Corporation), David Orchard (BEA Systems, Inc.), Sanjay Patil (SAP AG), Manjula Peiris (WSO2), Fabian Ritzmann (Sun Microsystems, Inc.), Daniel Roth (Microsoft Corporation), Tom Rutt (Fujitsu Limited), Sanka Samaranayake (WSO2), Felix Sasaki (W3C/Keio), Skip Snow (Citigroup), Yakov Sverdlov (CA), Mark Temple-Raston (Citigroup), Asir Vedamuthu (Microsoft Corporation), Sanjiva Weerawarana (WSO2), Ümit Yalçinalp (SAP AG), Prasad Yendluri (webMethods, Inc.).

Previous members of the Working Group were: Jeffrey Crump, Jong Lee, Bob Natale, Eugene Osovetsky, Bijan Parsia, Seumas Soltysik.

The people who have contributed to discussions on public-ws-policy@w3.org are also gratefully acknowledged.

E. Changes in this Version of the Document (Non-Normative)

A list of major editorial changes since the Working Draft dated 30 March, 2007 is below:

- Editorial changes to align with the OASIS WS-SecurityPolicy specification.
- Editorial changes to align with the W3C WS-Addressing Metadata specification.

F. Web Services Policy 1.5 - Primer Change Log (Non-Normative)

Date	Author	Description
20060816	ASV	Created first draft per action item 2 from the Austin F2F. This draft is based on a contribution from Microsoft.
20060829	ASV	Implemented the resolution for issue 3561: replaced URI with IRI.
20060919	DBO	Implemented the action 26 to add versioning material to primer.
20060924	TIB	Implemented the editorial action 35 to move the Security Considerations section to the Framework document.
20060924	TIB	Implemented the editorial action 36 to insert a reference to the Security Considerations section from the Framework document.
20060926	PY	Made a first pass at the changes to address issues reported by Paul Cotton.
20060928	PY	Completed making remaining changes to address issues reported by Paul Cotton. Fixing up the Acknowledgements is pending
20061020	PY	Implemented resolution for Issue 3827. Editors Action Item 56.
20061027	TIB	Implemented resolution for Issue 3815. Editors Action Item 55.
20061101	TIB	Implemented resolution for Issue 3815. Editors Action Item 68.
20061101	PY	Implemented the resolution for Issue 3791. Editors Action Item 67.
20061121	ASV	Implemented the resolution for issue 3809. Editors Action Item 79.
20061121	ASV	Implemented the resolution for issue 3966. Editors Action Item 81.
20061125	ASV	Reset Section **E. Changes in this Version of the Document** [p.45] .
20061125	ASV	Implemented the resolution for issue 3792. Editors Action Item 80: moved Sections 4.2 Parts of a Policy Assertion and 4.4.8 Versioning Policy Language into Section **3. Advanced Concepts: Policy Expression** [p.17] ; moved Section 4 Advanced Concepts II: Policy Assertion Design into the Guidelines document.
20061127	ASV	Added Frederick and Umit to the list of editors. Editors’ action 86.
20061207	FJH	Implemented the resolution for issue 3952 as outlined (with editorial correction replacing "for as" with "as"), Editors’ action 92.

20061213	TIB	Implemented the resolution for issue 3965 as outlined. Editors' action 94.
20070104	MH	Implemented the resolution for issue 4069 as outlined. Editors' action 110.
20070108	ASV	Reset Section **E. Changes in this Version of the Document** [p.45] .
20070118	FJH	Implemented the resolution for issue 4041 resolution corresponding to Editors' action 143.
20070122	PY	Completed action item: 118 Resolution for issue 4141
20070122	PY	Completed action item: 127 Resolution for issue 4197
20070131	FJH	Implemented resolution for issue 4270 as Resolved on 31 January 2007, closing editors action 151.
20070313	FJH	Applied resolution to issue 4379 with minor editorial revision (editors action 181). Updated references order.
20070314	FJH	Applied resolution to issue 4263 (editors action 195).
20070315	PY	Applied the resolution to issue 4339 (editors action 194).
20070315	PY	Applied the resolution to issue 4262 (editors action 201).
20070315	FJH	Applied resolution to issue 4255 (editors action 192).
20070315	ASV	Implemented the resolution for issue 4288. Editors' action 196.
20070315	ASV	Implemented the resolution for issue 3979. Editors' action 198.
20070315	FJH	Applied resolution to issue 4253 (editors action 191).
20070319	MH	Implemented the resolution for issue 4213 as outlined. Editors' action 189.
20070319	PY	Implemented the resolution for issue 4103 as outlined. Editors' action 193.
20070320	ASV	Implemented the resolution for issue 4300. Editors' action 190.
20070321	ASV	Updated section **E. Changes in this Version of the Document** [p.45] .
20070321	ASV	Formatted the example in **3.8.3 Use of Ignorable attribute and an alternative Versioning Scenario** [p.32] .
20070322	ASV	Deleted residual text in **4. Versioning Policy Language** [p.36] ; `s/The possible extensibility points with their current extensibility - including some outstanding issues related to extensibility - are:/The possible extensibility points are:/` ; `s/PolicyReference: any attribute and a proposal to add any element/PolicyReference: any attribute and any element/`.

20070426	PY	Editorial changes to align with the OASIS WS-SecurityPolicy specification. For issue 4318. Editors' action 244.
20070430	TIB	Editorial changes for issue 4393. Editors' action 239.
20070501	ASV	Reset Section **E. Changes in this Version of the Document** [p.45] .
20070502	TIB	Further changes for issue 4393. Editors' action 239.
20070502	DBO	Finished changes for issue 4414. Editors' action 239.
20070524	DBO	Finished changes for issue 4559. Editors' action 281, and issue 4375. Editors' action 282