Understanding Web Services Policy

July 6, 2006

Version 1.0

Authors

Asir S Vedamuthu (Editor), Microsoft

Daniel Roth, Microsoft

Summary

Understanding Web Services Policy 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 Framework and Web Services Policy Attachment specifications provide the complete normative description of the Web Services Policy language.

Contents

1. Introduction

This document, *Understanding Web Services Policy*, 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.

Section 2 - Basic Concepts: Policy Expression - 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.

Section 3 - Advanced Concepts I: Policy Expression – this is the first 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.

Section 4 - Advanced Concepts II: Policy Assertion Design - this is the second advanced section that walks through the dimensions of a policy assertion for assertion authors. This section describes the role of policy assertions, parts of a policy assertion, when to design policy assertions, outlines 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. Appendix C lists all the XML 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 requestors 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.

SOAP Message

```
<soap:Envelope>
  <soap:Header>
    <wsa:To>http://stock.contoso.com/realquote</wsa:To>
    <wsa:Action>http://stock.contoso.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. Appendix C lists all the XML 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. Tony is a Web service developer. He is building a client application that retrieves real time stock quote information from Contoso, Ltd. Contoso supplies real time data using Web services. Tony has Contoso's advertised WSDL description of these Web services. Contoso requires the use of addressing headers for messaging. Just the WSDL description is not sufficient for Tony to enable the interaction between his 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, Tony may have to read any related documentation, call someone at Contoso 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, Contoso may augment the service WSDL description with a policy that requires the use of addressing. Tony can use a policy-aware client that understands this policy and engages addressing automatically.

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

Policy Expression

```
<Policy>
  <wsap:UsingAddressing />
</Policy>
```

The policy expression in the above example consists of a `Policy` main element and a child element `wsap:UsingAddressing`. Child elements of the `Policy` element are policy assertions. Contoso attaches the above policy expression to a WSDL binding description.

The `wsap:UsingAddressing` element is a policy assertion. (The prefix `wsap` is used here to denote the Web Services Addressing – WSDL Binding 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, Contoso requires the use of transport-level security for protecting messages.

Secure Message

```
<soap:Envelope>
  <soap:Header>
    <wss:Security soap:mustUnderstand="1" >
      <wsu:Timestamp u: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://real.contoso.com/quote</wsa:To>
    <wsa:Action>http://real.contoso.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. Contoso 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, Contoso 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.

Addressing and Security Policy Expression

```
<Policy>
  <wsap:UsingAddressing />
  <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.

Tony 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 Contoso 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. Contoso 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 Contoso, 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, like Tony, 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
- Web Services Reliable Messaging Policy
- Web Services Atomic Transaction
- Web Services Business Activity Framework
- Devices Profile for Web Services
- A Technical Reference for Windows CardSpace
- …

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.

Addressing and Security Policy Expression

```
<All>
  <wsap:UsingAddressing />
  <sp:TransportBinding>...</sp:TransportBinding>
</All>
```

In addition to requiring the use of addressing, Contoso 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, Contoso uses the `sp:AsymmetricBinding` policy assertion (see the example below).

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, Contoso 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.

Transport- or Message-Level Security Policy Expression

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

Contoso 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.

Addressing and Transport- OR Message-Level Security Policy Expression

```
<All>
  <wsap:UsingAddressing />
  <ExactlyOne>
     <sp:TransportBinding>...</sp:TransportBinding>
     <sp:AsymmetricBinding>...</sp:AsymmetricBinding >
  </ExactlyOne>
</All>
```

Using this policy expression, Contoso gives the choice of mechanisms for protecting messages to clients (or requestors).

2.6 Optional Policy Assertion

Through a customer survey program, Contoso 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. Contoso 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, Contoso uses the `mtom:OptimizedMimeSerialization` policy assertion (see the example below).

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 for 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 Contoso'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.

Optional MIME Serialization, Addressing and Transport- OR Message-Level Security Policy Expression

```
<All>
  <mtom:OptimizedMimeSerialization wsp:Optional=”true”/>
  <wsap:UsingAddressing />
  <ExactlyOne>
     <sp:TransportBinding>...</sp:TransportBinding>
     <sp:AsymmetricBinding>...</sp:AsymmetricBinding >
  </ExactlyOne>
</All>
```

Contoso is able to meet their customer needs by adding optional support for the Optimized MIME Serialization. An optional policy assertion represents a behavior that may be engaged.

2.7 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. Contoso 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 Contoso's Web services, Tony 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, Contoso 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).

Transport Security Policy Assertion

```
<sp:TransportBinding>
  <Policy>
    <sp:TransportToken>
      <Policy>
        <sp:HttpsToken RequireClientCertificate="false" />
      </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, like Tony, 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.

2.8 Referencing Policy Expressions

Contoso 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 Contoso's customers, Contoso supports multiple WSDL bindings for these Web services. Contoso 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.

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

Common Policy Expression

```
<Policy wsu:Id=”common”>
  <mtom:OptimizedMimeSerialization wsp:Optional=”true”/>
  <wsap:UsingAddressing />
</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 URI for referencing this policy expression (within the same document) is #common. If the policy document URI is `http://real.contoso.com/policy.xml` then the absolute URI for referencing this policy expression is `http://real.contoso.com/policy.xml#common.` (The absolute URI is formed by combining the document URI, # and the value of the `wsu:Id` attribute.)

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.

PolicyReference to Common Policy Expression

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

For referencing a policy expression within the same XML document, Contoso uses the `wsu:Id` attribute for identifying a policy expression and a URI 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.

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 URI 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 URI of this policy expression is `http://real.contoso.com/policy/common`.

Common Policy Expression

```
<Policy Name=”http://real.contoso.com/policy/common”>
  <mtom:OptimizedMimeSerialization wsp:Optional=”true”/>
  <wsap:UsingAddressing />
</Policy>
```

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

PolicyReference to Common Policy Expression

```
<PolicyReference URI=”http://real.contoso.com/policy/common”/>
```

2.9 Attaching Policy Expressions to WSDL

A majority of Contoso’s customers use WSDL for building their client applications. Contoso 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, Contoso 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`.

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, Contoso 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`). Contoso does not require the use of security for these services, but requires the use of addressing and allows the use of optimization.

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, Contoso 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 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 (other than 'no claims') about the absence of policy expressions.

Service providers, like Contoso, 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, like Tony, 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.

2.10 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 Contoso, use policy expressions to represent combinations of capabilities and requirements. Web service developers, like Tony, 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 I: Policy Expression

In Section 2, we covered the basics of Web Services Policy language. This is the first 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?

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 descendent 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 Contoso's policy expression (see below) from the previous section.

Contoso's Secure Policy Expression

```
<Policy>
  <All>
    <mtom:OptimizedMimeSerialization wsp:Optional=”true”/>
    <wsap:UsingAddressing />
    <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:

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.

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 Contoso’s policy expression (see the example below). Contoso 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 requestors:

(a) Requires the use of addressing and transport-level security
(b) Requires the use of addressing and message-level security
(c) Requires the use of optimization, addressing and transport-level security and
(d) Requires the use of optimization, addressing and message-level security.

Contoso’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”/>
  <wsap:UsingAddressing />
```

```
</Policy>
```

Let us look at the normal form for this policy expression. The example below is Contoso'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.

Contoso's Policy Expression in Normal Form

```
<Policy>
  <ExactlyOne>
    <All> <!-- - - - - - - - - - - - - - Policy Alternative (a) -->
       <wsap:UsingAddressing />
       <sp:TransportBinding>...</sp:TransportBinding>
    </All>
    <All> <!-- - - - - - - - - - - - - - Policy Alternative (b) -->
      <wsap:UsingAddressing />
      <sp:AsymmetricBinding>...</sp:AsymmetricBinding >
    </All>
    <All> <!-- - - - - - - - - - - - - - Policy Alternative (c) -->
       <mtom:OptimizedMimeSerialization />
       <wsap:UsingAddressing />
       <sp:TransportBinding>...</sp:TransportBinding>
    </All>
    <All> <!-- - - - - - - - - - - - - - Policy Alternative (d) -->
       <mtom:OptimizedMimeSerialization />
       <wsap:UsingAddressing />
       <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. 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:

o Do nothing. A policy expression with the referenced URI is already known to be available in a local cache or chip (embedded systems).

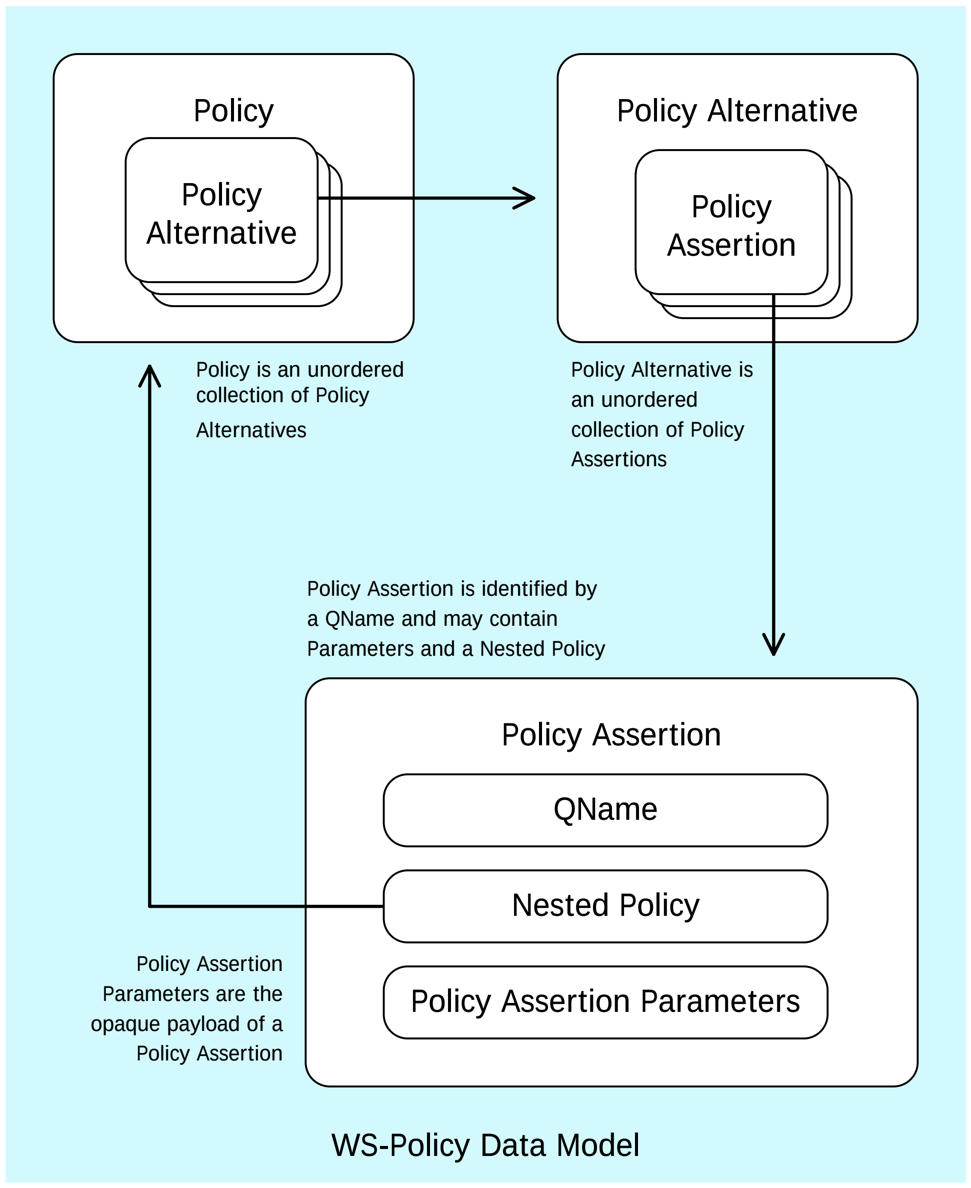
- Use the referenced URI and retrieve an existing policy expression from the containing XML document: a policy element with an XML ID.
- Use the referenced URI and retrieve a policy expression from some policy repository (local or remote) or catalog. Policy tools may use any protocols (say Web Services Metadata Exchange) for such metadata retrieval. These protocols may require additional out of band information.
- Attempt to resolve the referenced URI 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) attribute and referenced using a URI reference to this XML ID value.

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.

Contoso uses a policy to convey the conditions for an interaction. Policy-aware clients, like the one used by Tony (as explained earlier in Section 2), 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.


Policy
Policy Alternative
Policy Alternative
Policy Assertion
Policy is an unordered collection of Policy Alternatives
Policy Alternative is an unordered collection of Policy Assertions
Policy Assertion is identified by a QName and may contain Parameters and a Nested Policy
Policy Assertion
QName
Nested Policy
Policy Assertion Parameters
Policy Assertion Parameters are the opaque payload of a Policy Assertion
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, like Tony, 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 are the assertion parameters.

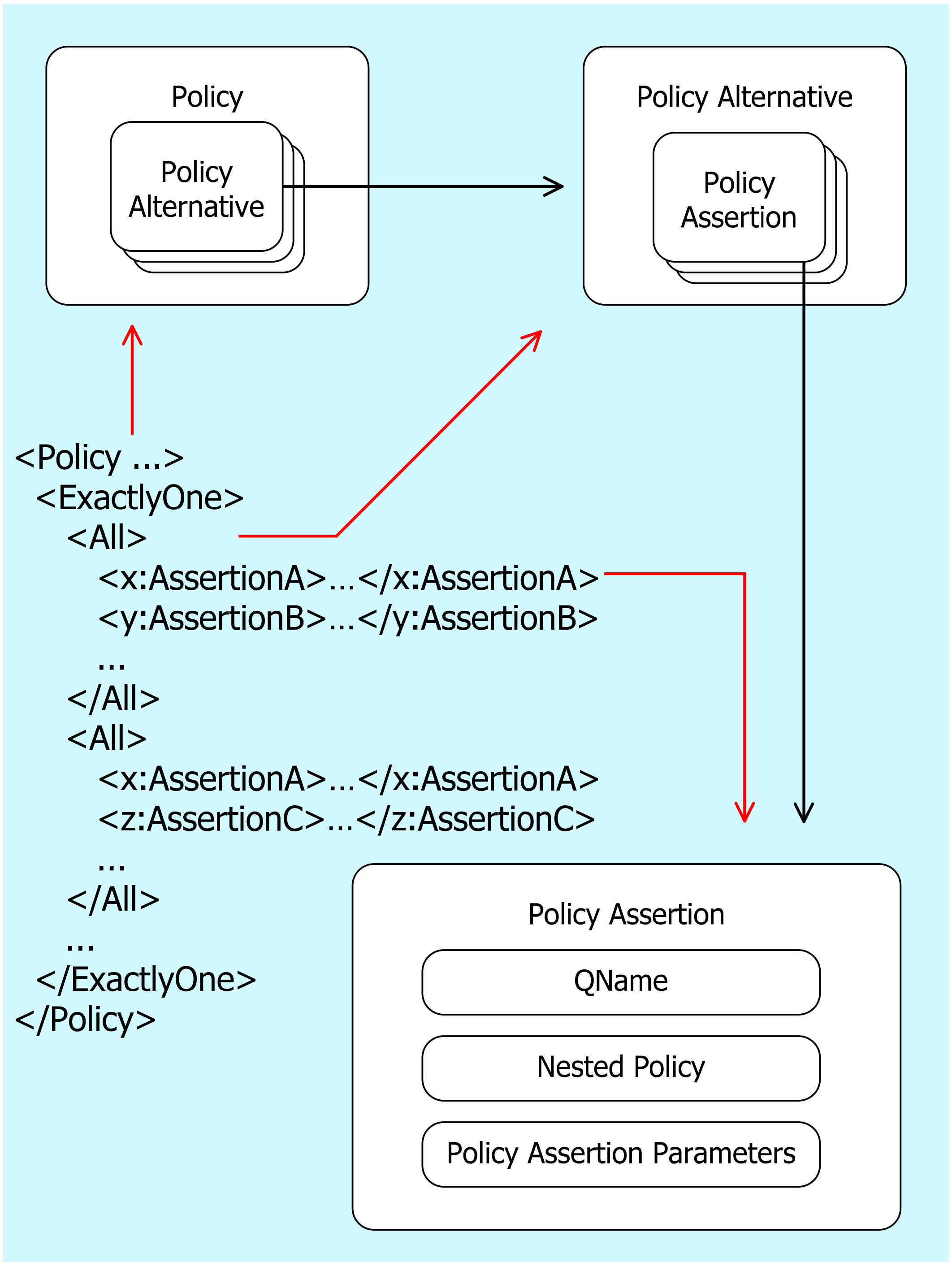
We considered nested policy expressions in the context of a security usage scenario. Let us look at its shape in the policy data model. In the normal form, a nested policy is a policy that has at most one policy alternative and is owned by its parent policy assertion. The policy alternative in a nested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

3.4 Compatible Policies

A provider, like Contoso, and a requestor, like Tony's policy-aware client, 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 Contoso's policy expression (from Section 3.2). As we saw before, Contoso offers four policy alternatives. Of them, one of the policy alternatives requires the use of addressing and transport-level security.

Contoso's Policy Expression

```
<Policy>
  <ExactlyOne>
    <All> <!-- - - - - - - - - - -   Contoso’s Policy Alternative (a) -->
       <!-- - - - - - - - - - - - - - - - - - - - - Policy Assertion (c1) -->
       <wsap:UsingAddressing />
       <!-- - - - - - - - - - - - - - - - - - - - - Policy Assertion (c2) -->
       <sp:TransportBinding>...</sp:TransportBinding>
    </All>
    …
  </ExactlyOne>
</Policy>
```

Tony's organization requires the use of addressing and transport-level security for any interaction with Contoso's Web services. Tony 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.

Tony's Policy Expression in Normal Form

```
<Policy>
  <ExactlyOne>
    <All> <!-- - - - - - - - - - - - - - -  Tony’s Policy Alternative -->
       <!-- - - - - - - - - - - - - - - - - - - - - Policy Assertion (t1) -->
      <sp:TransportBinding>...</sp:TransportBinding>
       <!-- - - - - - - - - - - - - - - - - - - - - Policy Assertion (t2) -->
      <wsap:UsingAddressing />
    </All>
  </ExactlyOne>
</Policy>
```

Tony lets his policy-aware client select a compatible policy alternative in Contoso's policy. How does this client select a compatible policy alternative? It is simple – it uses the policy intersection. That is, Tony's policy-aware client uses these two policy expressions (Tony's and Contoso'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 Contoso's policy alternative are compatible with policy assertions (t2) and (t1) in Tony's policy alternative. Contoso's policy alternative (a) and Tony'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, Contoso's policy alternative (a) is compatible with Tony's policy alternative. Contoso's policy and Tony's policy are compatible because one of Contoso's policy alternative is compatible with Tony's policy alternative.

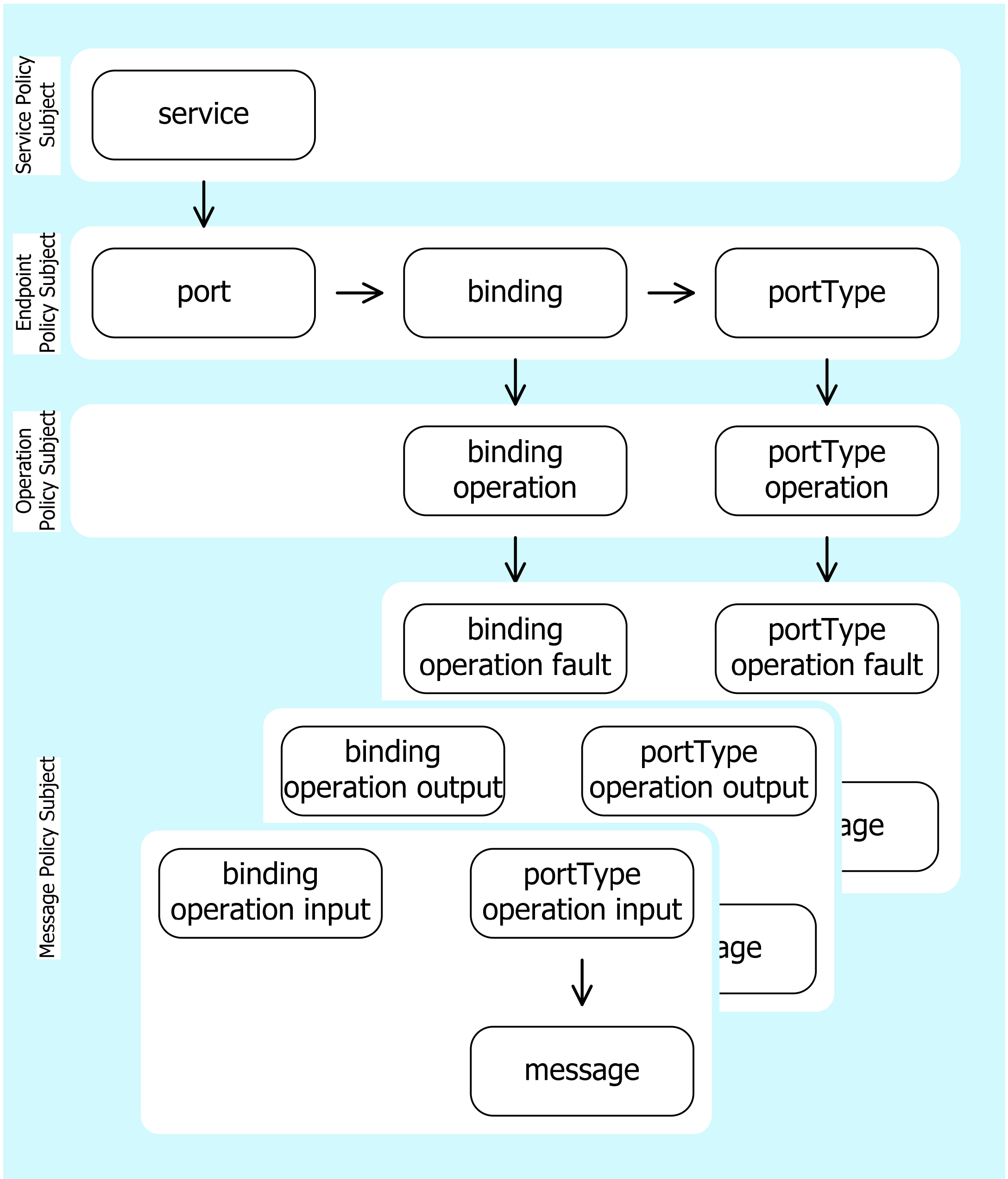
For this interaction, Tony's policy-aware client can use policy alternative (a) to satisfy Contoso's conditions or requirements.

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.

3.5 Attaching Policy Expressions to WSDL

In Section 2, we looked into how Contoso 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.



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.

Contoso'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 Combine Policies

Multiple policy expressions may be attached to WSDL constructs. Let us consider how Contoso 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.

Multiple Policy Expressions Attached to Endpoint Policy Subject

```
<Policy wsu:Id=”common2”>
  <mtom:OptimizedMimeSerialization wsp:Optional=”true”/>
  <wsap:UsingAddressing />
</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 Contoso's secure policy in Section 2 and has four policy alternatives. In other words, the combination of two policies is the cross product of alternatives in these two policies.

Effective Policy of the Endpoint Policy Subject in the Previous Example

```
<Policy>
  <All>
    <Policy>
      <mtom:OptimizedMimeSerialization wsp:Optional=”true”/>
      <wsap:UsingAddressing />
    </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.

3.7 Extensibility and Versioning

Web Services Policy language is an extensible language by design. The `Policy`, `ExactlyOne`, `All` and `PolicyReference` elements are extensible. The `Policy`, `ExactlyOne` and `All` elements allow child element and attribute extensibility. The `PolicyReference` element allows 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 elements as policy assertions.

Web Services Policy language enables simple versioning practices that allow requestors to continue the use of any older policy alternatives in a backward compatible manner. This allows service providers, like Contoso, to deploy new behaviors using additional policy assertions without breaking compatibility with clients that rely on any older policy alternatives.

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

Contoso's Version 1 Policy Expression

```
<Policy>
  <ExactlyOne>
```

```
    <All>
      <wsap:UsingAddressing />
      <sp:TransportBinding>...</sp:TransportBinding>
    </All>
  </ExactlyOne>
</Policy>
```

Over time, Contoso adds support for advanced behaviors: requiring the use of addressing and message-level security for protecting messages. They added this advanced support without breaking compatibility with requestors that rely on addressing and transport-level security. The example below is Contoso’s version 2 policy expression. In this version, Contoso’s 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 Contoso’s using the old policy alternative. Of course, these clients have the option to migrate from using old policy alternatives to new policy alternatives.

Contoso’s Version 2 Policy Expression

```
<Policy>
  <ExactlyOne>
    <All>
      <wsap:UsingAddressing />
      <sp:TransportBinding>...</sp:TransportBinding>
    </All>
    <All> <!-- - - - - - - - - - - - - - - - - - NEW Policy Alternative -->
      <wsap:UsingAddressing />
      <sp:AsymmetricBinding>...</sp: AsymmetricBinding >
    </All>
  </ExactlyOne>
</Policy>
```

When Contoso 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, policy can be 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 policy 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 Contoso, incrementally deploy advanced behaviors, some requestors may not recognize these new policy assertions. As discussed before, these requestors 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 requestors and providers. Today, 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.

4. Advanced Concepts II: Policy Assertion Design

In the previous section, we covered in-depth materials for Web Services Policy implementers. This is the second advanced section that walks through the dimensions of a policy assertion for assertion authors. This section covers the following topics:

- What is the role of policy assertions?
- What are the parts of a policy assertion?
- When to design policy assertions?
- What are the guidelines for designing policy assertions?
- What are the minimum requirements for describing policy assertions?

4.1 Role of Policy Assertions

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`. Policy is a flexible language to represent consistent combinations of behaviors using policy operators: `Policy`, `All` and `ExactlyOne`. Policy is an expressive language and capable of representing behaviors from a variety of domains. Let us look for the key parts that unlock this potential.

Service providers combine behaviors for an interaction from domains such as messaging, security, reliability and transactions. To enable clients to engage these behaviors, services require some way to advertise these behaviors. Providers require machine readable metadata pieces that identify these behaviors. A policy assertion is a machine-readable metadata piece that requires the use of a behavior identified by the assertion. Web service developers can use policy-aware clients that recognize these assertions and engage their corresponding behaviors automatically.

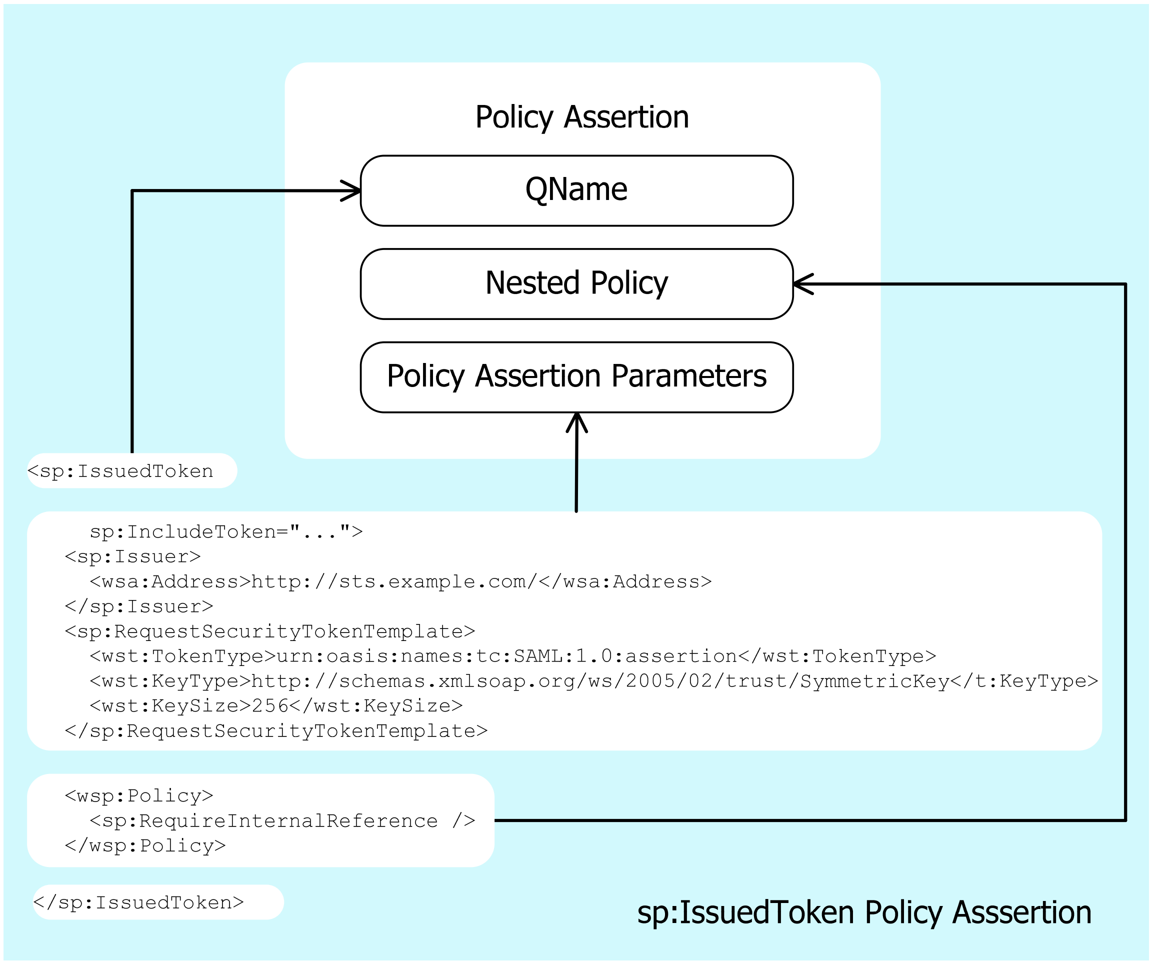
Policy assertions are the key parts and play a central role to unlock the potential offered by the Web Services Policy language. Assertions are defined by product designers, protocol authors, protocol implementers and Web service developers.

Policy assertion authors identify behaviors required for Web services interactions and represent these behaviors as policy assertions. By designing policy assertions, assertion authors make a significant contribution to automate Web services interactions and enable advanced behaviors.

4.2 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.



```
<sp:IssuedToken
    sp:IncludeToken="...">
  <sp:Issuer>
    <wsa:Address>http://sts.example.com/</wsa:Address>
  </sp:Issuer>
  <sp:RequestSecurityTokenTemplate>
    <wst:TokenType>urn:oasis:names:tc:SAML:1.0:assertion</wst:TokenType>
    <wst:KeyType>http://schemas.xmlsoap.org/ws/2005/02/trust/SymmetricKey</t:KeyType>
    <wst:KeySize>256</wst:KeySize>
  </sp:RequestSecurityTokenTemplate>
  <wsp:Policy>
    <sp:RequireInternalReference />
  </wsp:Policy>
</sp:IssuedToken>
```

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. Requestors 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, requestors 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 Contoso requires the use of a SAML token issued by a third party. Service providers, like Contoso, 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 Contoso's Web services.

4.3 When to design policy assertions?

As we illustrated in the previous section, requiring the use of a security token issued by a third party is represented as a policy assertion. In simple words, a policy assertion identifies a domain specific behavior:

- That is a requirement
- That is relevant to an interoperable Web service interaction
- That is relevant to an interaction that involves two or more Web service participants
- That applies to its associated policy subject such as service, endpoint, operation and message.

Given that interoperability and automation are the motivations, policy assertions that represent opt-in, shared and visible behaviors are useful pieces of metadata. Such assertions enable tooling and improve interoperability. The key to understanding when to design policy assertions is to have clarity on the characteristics of a behavior represented by a useful policy assertion: opt-in, shared and visible.

Opt-in behavior

An opt-in behavior refers to a requirement that providers and requestors must deliberately choose to engage for a successful web service interaction. Examples of such behaviors are the use of optimization, message-level security, reliable messaging and atomic transaction. Policy assertions are not necessary to interoperate on widespread assumed behaviors. An example of an assumed behavior is the use of UTF-8 or UTF-16 text encoding for XML messages.

Shared behavior

A shared behavior refers to a requirement that is relevant to an interoperable Web service interaction and involves two or more participants. If an assertion only describes one participant's behavior (non-shared behavior) then the assertion is not relevant to an interoperable interaction. Non-shared behaviors do not add any value for tooling or interoperability. An example of a non-shared behavior is the use of logging or auditing by the provider.

Requestors may use the policy intersection to select a compatible policy alternative for a Web service interaction. If an assertion only describes one participant's behavior then this assertion will not be present in the other participants' policy and the policy intersection will unnecessarily produce false negatives.

Visible behavior

A visible behavior refers to a requirement that manifests on the wire. Web services provide interoperable machine-to-machine interaction among disparate systems. Web service interoperability is the capability of disparate systems to exchange data using common data formats and protocols such as messaging, security, reliability and transaction. Such data formats and protocols manifest on the wire. Providers and requestors only rely on these wire messages that conform to such formats and protocols for interoperability. If an assertion describes a behavior that does not manifest on the wire then the assertion is not relevant to an interoperable interaction.

For example, say an assertion describes the privacy notice information of a provider and there is an associated regulatory safeguard in place on the provider's side. Such assertions may represent business or regulatory level metadata but do not add any value to interoperability.

If an assertion has no wire- or message-level visible behavior, then the interacting participants may require some sort of additional non-repudiation mechanism to indicate compliance with the assertion. Introducing an additional non-repudiation mechanism adds unnecessary complexity to processing a policy assertion.

4.4 Guidelines for Designing Assertions

The policy language allows assertion authors to invent their own XML dialects to represent policy assertions. The policy language builds on natural XML nesting and leverages XML Schema validation. The policy language relies only on the QName of the policy assertion XML element. Everything else is left for the policy assertion authors to design. The policy language offers plenty of options to assertion authors such as independent assertions, dependent assertions, nested policies and assertion parameters.

The description of a policy assertion should identify a single domain specific behavior in an objective manner and answer the following questions:

- What is the behavior? (In the previous section, we discussed the characteristics of a behavior represented by a useful policy assertion.)
- What are the assertion parameters?
- Are there any dependent behaviors, and how are they represented?
- What is the assertion's QName and XML information set representation?
- What is the policy subject of this behavior?
- What are the attachment points?

As you would have expected, the policy assertion design is more than a technical design. Given that interoperability and automation are the motivations, policy assertion design is a business process to reach agreements with relevant stakeholders for interoperability and tooling. Setting aside the business aspects of the design, the rest of this section walks through a few tradeoffs or dimensions to consider and provides technical guidelines for designing policy assertions.

4.4.1 Optional Behaviors

A policy assertion identifies a domain specific behavior that is a requirement relevant to a Web Service interaction. Policy assertions can be marked optional using the `wsp:Optional` attribute marker to represent behaviors that may be engaged (capabilities) for an interaction. It is important to note that behavior (policy assertion) and optional representation (`wsp:Optional` attribute) are distinct ideas of the Web Services Policy language. Conflating these distinct ideas unnecessarily disrupts scenarios that depend on the policy intersection: if an assertion indicates an optional behavior and this assertion is not present in the other participants' policy then the policy intersection will unnecessarily produce false negatives.

Best practice: use the `wsp:Optional` attribute to indicate optional behaviors.

4.4.2 Assertion vs. assertion parameter

Policy assertion parameters are the opaque payload of an assertion. Parameters carry additional useful information for engaging the behavior described by an assertion and are preserved through policy processing such as normalize, merge and policy intersection. Requestors may use policy intersection to select a compatible policy alternative for an interaction. Assertion parameters do not affect the outcome of policy intersection.

In the example below, `sp:Body` and `sp:Header` elements are the two assertion parameters of the `sp:SignedParts` policy assertion (this assertion requires the parts of a message to be protected). These two parameters identify the parts of a wire message that should be protected. These parameters carry additional useful information for engaging the behavior that is irrelevant to compatibility tests.

Policy Assertion with Assertion Parameters

```
<Policy>
  <sp:SignedParts>
    <sp:Body />
    <sp:Header />
  </sp:SignedParts>
  …
</Policy>
```

Best practice: represent useful (or additional) information necessary for engaging the behavior represented by a policy assertion as assertion parameters.

4.4.3 Leveraging Nested Policy

As we have seen before, a nested policy expression further qualifies the dependent behaviors of its parent policy assertion. When we consider nested policy there is always two or more policy assertions involved. The following design questions below can help you to determine when to use nested policy expressions:

Are these assertions designed for the same policy subject?

Do these assertions represent dependent behaviors?

If the answers are yes to both of these questions then leveraging nested policy expressions is a good idea. Keep in mind that a nested policy expression participates in the policy intersection algorithm. If a requestor uses policy intersection to select a compatible policy alternative then the assertions in a nested policy expression play a first class role in the outcome. There is one caveat to watch out for: policy assertions with deeply nested policy can greatly increase the complexity of a policy and should be avoided when they are not needed.

Best practice: represent dependent behaviors that apply to the same policy subject using nested policy assertions.

4.4.4 Minimal approach

How big should an assertion be? How many assertion parameters should the assertion enumerate? How many dependent behaviors should the assertion enumerate? It is always good to start with a simple working policy assertion that allows extensibility. As your design work progresses, you may add more parameters or nested policy assertions to meet your interoperability needs.

Best practice: start with a simple working assertion that allows extensibility.

4.4.5 QName and XML Information Set representation

As mentioned before, Web Services Policy language allows assertion authors to invent their own XML dialects to represent policy assertions. The policy language relies only on the policy assertion XML element QName. This QName is unique and identifies the behavior represented by a policy assertion. Assertion authors have the option to represent an assertion parameter as a child element (by leveraging natural XML nesting) or an attribute of an assertion. The general guidelines on when to use XML elements versus attributes apply.

The syntax of an assertion can be represented using an XML outline (plus an XML schema document). If the assertion has a nested policy expression then the assertion XML outline can enumerate the nested assertions that are allowed.

Best practice: use a unique QName to identify the behavior and provide an XML outline (plus an XML schema document) to specify the syntax of an assertion.

4.4.6 Policy subject and attachment points

A behavior identified by a policy assertion applies to the associated policy subject. If a policy assertion is to be used with WSDL, policy assertion authors must specify a WSDL policy subject. What is the policy subject of this behavior?

- If the behavior applies to any message exchange using any of the endpoints offered by a service then the subject is the service policy subject.
- If the behavior applies to any message exchange made using an endpoint then the subject is the endpoint policy subject.
- If the behavior applies to any message exchange defined by an operation then the subject is the operation policy subject.
- If the behavior applies to an input message then the subject is the message policy subject - similarly for output and fault message policy subjects.

For a given WSDL policy subject, there may be several attachment points. For example, there are three attachment points for the endpoint policy subject: the `port`, `binding` and `portType` element. Policy assertion authors should identify the relevant attachment point when defining a new assertion. To determine the relevant attachment points, authors should consider the scope of the attachment point. For example, an assertion should only be allowed in the `portType` element if the assertion reasonably applies to any endpoint that ever references that `portType`. Most of the known policy assertions are designed for the endpoint, operation or message policy subject. The commonly used attachment points for these policy subjects are outlined in Section 3.7 - Attaching Policy Expressions to WSDL.

The service policy subject is a collection of endpoint policy subjects. The endpoint policy subject is a collection of operation policy subjects and etc. As you can see, the WSDL policy subjects compose naturally. It is quite tempting to associate the identified behavior to a broader policy subject than to a fine granular policy subject. For instance, it is convenient to attach a supporting token assertion (defined by the Web Services Security Policy specification) to an endpoint policy subject instead of a message policy subject. For authoring convenience, an assertion author may allow the association of an assertion to multiple policy subjects. If an assertion is allowed to be associated with multiple policy subjects then the assertion author has the burden to describe the semantics of multiple instances of the same assertion attached to multiple policy subjects at the same time. The best practice is to choose the most granular policy subject that the behavior applies to.

Best practice: specify a policy subject, choose the most granular policy subject that the behavior applies to and specify a preferred attachment point.

4.4.7 Versioning behaviors

Over time, there may be multiple equivalent behaviors emerging in the Web Service interaction space. Examples of such multiple equivalent behaviors are WSS: SOAP Message Security 1.0 vs. 1.1 and WS-Addressing August 2004 version vs. WS-Addressing W3C Recommendation. These equivalent behaviors are mutually exclusive for an interaction. Such equivalent behaviors can be modeled as independent assertions. The policy expression in the example below requires the use of WSS: SOAP Message Security 1.0.

Message-level Security and WSS: SOAP Message Security 1.0

```
<Policy>
  <sp:Wss10>...</sp:Wss10>
</Policy>
```

The policy expression in the example below requires the use of WSS: SOAP Message Security 1.1. These are multiple equivalent behaviors and are represented using distinct policy assertions.

Message-level Security and WSS: SOAP Message Security 1.1

```
<Policy>
  <sp:Wss11>…</sp:Wss11>
</Policy>
```

Best practice: use independent assertions for modeling multiple equivalent behaviors.

4.5 Describing Policy Assertions

Thus far, we walked through the dimensions of a policy assertion and guidelines for authoring policy assertions. Let us look at what are the minimum requirements for describing policy assertions in specifications:

1. Description must clearly and completely specify the syntax (plus an XML Schema document) and semantics of a policy assertion.
2. If there is a nested policy expression, description must declare it and enumerate the nested policy assertions that are allowed.
3. A policy alternative may contain multiple instances of the same policy assertion. Description must specify the semantics of parameters and nested policy (if any) when there are multiple instances of a policy assertion in the same policy alternative.
4. If a policy assertion is to be used with WSDL, description must specify a WSDL policy subject – such as service, endpoint, operation and message.

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.

Appendix A – Security Considerations

This appendix describes the security considerations that service providers, requestors, policy authors, policy assertion authors, and policy implementers need to consider when exposing, consuming and designing policy expressions, authoring policy assertions or implementing policy.

Information Disclosure Threats

A policy is used to represent the capabilities and requirements of a Web Service. Policies may include sensitive information. Malicious consumers may acquire sensitive information, fingerprint the service and infer service vulnerabilities. These threats can be mitigated by requiring authentication for sensitive information, by omitting sensitive information from the policy or by securing access to the policy. For securing access to policy metadata, policy providers can use mechanisms from other Web Services specifications such as WS-Security and WS-MetadataExchange.

Spoofing and Tampering Threats

If a policy expression is unsigned it could be easily tampered with or replaced. To prevent tampering or spoofing of policy, requestors should discard a policy unless it is signed by the provider and presented with sufficient credentials. Requestors should also check that the signer is actually authorized to express policies for the given policy subject.

Downgrade Threats

A policy may offer several alternatives that vary from weak to strong set of requirements. An adversary may interfere and remove all the alternatives except the weakest one (say no security requirements). Or, an adversary may interfere and discard this policy and insert a weaker policy previously issued by the same provider. Policy authors or providers can mitigate these threats by sun-setting older or weaker policy alternatives. Requestors can mitigate these threats by discarding policies unless they are signed by the provider.

Repudiation Threats

Malicious providers may include policy assertions in its policy whose behavior cannot be verified by examining the wire message from the provider to requestor. In general, requestors have no guarantee that a provider will behave as described in the provider’s policy expression. The provider may not and perform a malicious activity. For example, say the policy assertion is privacy notice information and the provider violates the semantics by disclosing private information. Requestors can mitigate this threat by discarding policy alternatives which include assertions whose behavior cannot be verified by examining the wire message from the provider to requestor. Assertion authors can mitigate this threat by not designing assertions whose behavior cannot be verified using wire messages.

Denial of Service Threats

Malicious providers may provide a policy expression with a large number of alternatives, a large number of assertions in alternatives, deeply nested policy expressions or chains of `PolicyReference` elements that expand exponentially (see the chained sample below; this is similar to the well-known DTD entity expansion attack). Policy implementers need to anticipate these rogue providers and use a configurable bound with defaults on number of policy alternatives, number of assertions in an alternative, depth of nested policy expressions, etc.

Chained Policy Reference Elements

```
<Policy wsu:Id=”p1”>
  <PolicyReference URI=”#p2”/ >
  <PolicyReference URI=”#p2”/>
</Policy>

<Policy wsu:Id=”p2” >
  <PolicyReference URI=”#p3”/>
  <PolicyReference URI=”#p3”/>
</Policy>

<Policy wsu:Id=”p3” >
  <PolicyReference URI=”#p4”/>
  <PolicyReference URI=”#p4”/>
</Policy>

<!-- Policy/@wsu:Id p4 through p99 -->

<Policy wsu:Id=”p100” >
  <PolicyReference URI=”#p101”/>
  <PolicyReference URI=”#p101”/>
</Policy>

<Policy wsu:Id=”p101” >
  <mtom:OptimizedMimeSerialization />
</Policy>
```

Malicious providers may provide a policy expression that includes multiple `PolicyReference` elements that use a large number of different internet addresses. These may require the consumers to establish a large number of TCP connections. Policy implementers need to anticipate such rogue providers and use a configurable bound with defaults on number of `PolicyReference` elements per policy expression.

General XML Considerations

Implementers of Web Services policy language should be careful to protect their software against general XML threats like deeply nested XML or XML that contains malicious content.

Appendix B – Acknowledgements

This document has been developed as a result of joint work with many individuals and teams, including: Paul Cotton, Colleen Evans, Kirill Gavrylyuk, Martin Gudgin, Ram Jeyaraman, Andrew Layman, Jonathan Marsh, Jeffrey Schlimmer, Jorgen Thelin, and Kyle Young.

Appendix C – XML Namespaces

Table 1 lists XML namespaces that are used in this document. The choice of any namespace prefix is arbitrary and not semantically significant.

Table 1: Prefixes and XML Namespaces used in this specification.

Prefix	XML Namespace	Specification(s)
	http://schemas.xmlsoap.org/ws/2004/09/policy	[WS-Policy, WS-PolicyAttachment]
mtom	http://schemas.xmlsoap.org/ws/2004/09/policy/optimizedmimeserialization	[WS-OptimizedSerializationPolicy]
soap	http://www.w3.org/2003/05/soap-envelope	[SOAP 1.2]
sp	http://schemas.xmlsoap.org/ws/2005/07/securitypolicy	[WS-SecurityPolicy]
wsa	http://www.w3.org/2005/08/addressing	[WS-Addressing]
wsap	http://www.w3.org/2006/05/addressing/wsdl	[WS-AddressingPolicy]
wsdl	http://schemas.xmlsoap.org/wsdl/	[WSDL 1.1]
wsp	http://schemas.xmlsoap.org/ws/2004/09/policy	[WS-Policy, WS-PolicyAttachment]
wss	http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd	[WS-Security 2004]
wst	http://schemas.xmlsoap.org/ws/2005/02/trust	[WS-Trust]
wsu	http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd	[WS-Security 2004]

Appendix D – References

[MTOM]
M. Gudgin, et al, "SOAP Message Transmission Optimization," January 2005. (See http://www.w3.org/TR/2005/REC-soap12-mtom-20050125/.)

[SOAP 1.1]
D. Box, et al, "Simple Object Access Protocol (SOAP) 1.1," May 2000. (See http://www.w3.org/TR/2000/NOTE-SOAP-20000508.)

[SOAP 1.2]
M. Gudgin, et al, "SOAP Version 1.2 Part 1: Messaging Framework," June 2003. (See http://www.w3.org/TR/2003/REC-soap12-part1-20030624/.)

[XOP]
M. Gudgin, et al, "XML-binary Optimized Packaging," January 2005. (See http://www.w3.org/TR/2005/REC-xop10-20050125/.)

[WS-Addressing]

M. Gudgin, et al, "Web Services Addressing 1.0 - Core," May 2006. (See http://www.w3.org/TR/2006/REC-ws-addr-core-20060509/.)

[WS-AddressingPolicy]
M. Gudgin, et al, "Web Services Addressing 1.0 – WSDL Binding," May 2006. (See http://www.w3.org/TR/2006/CR-ws-addr-wsdl-20060529/. The latest version of this specification is available at http://www.w3.org/TR/ws-addr-wsdl/.)

[WS-MetadataExchange]
K. Ballinger, et al, "Web Services Metadata Exchange (WS-Metadata Exchange)," September 2004. (See http://schemas.xmlsoap.org/ws/2004/09/mex/.)

[WSDL 1.1]
E. Christensen, et al, "Web Services Description Language (WSDL) 1.1," March 2001. (See http://www.w3.org/TR/2001/NOTE-wsdl-20010315.)

[WS-Policy]
S. Bajaj, et al, "Web Services Policy Framework (WS-Policy)," March 2006. (See http://schemas.xmlsoap.org/ws/2004/09/policy.)

[WS-PolicyAttachment]
S. Bajaj, et al, "Web Services Policy Attachment (WS-PolicyAttachment)," March 2006. (See http://schemas.xmlsoap.org/ws/2004/09/policy.)

[WS-Security 2004]
A. Nadalin, et al, "Web Services Security: SOAP Message Security 1.0 (WS-Security 2004)," March 2004. (See http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf.)

[WS-SecurityPolicy]
G. Della-Libera, et al, "Web Services Security Policy Language (WS-SecurityPolicy)," July 2005. (See http://schemas.xmlsoap.org/ws/2005/07/securitypolicy.)

[WS-Trust]
S.Anderson, et al, "Web Services Trust Language (WS-Trust)," February 2005. (See http://schemas.xmlsoap.org/ws/2005/02/trust.)